aws

Guia do usuário

Amazon Aurora DSQL

Amazon Aurora DSQL: Guia do usuário

Table of Contents

O que é o Amazon Aurora DSQL?

O Amazon Aurora DSQL é um serviço de banco de dados relacional distribuído e sem servidor otimizado para workloads transacionais. O Aurora DSQL oferece escala praticamente ilimitada e não exige que você gerencie a infraestrutura. A arquitetura ativa-ativa altamente disponível oferece 99,99% de disponibilidade em uma única região e 99,999% em várias regiões.

Quando usar o Amazon Aurora DSQL

O Aurora DSQL é otimizado para workloads transacionais que se beneficiam das transações ACID e de um modelo de dados relacional. Por ser sem servidor, o Aurora DSQL é ideal para padrões de aplicação de arquiteturas de microsserviços, sem servidor e orientadas a eventos. O Aurora DSQL é compatível com o PostgreSQL, de modo que você pode usar drivers conhecidos, mapeamentos de objetos relacionais (ORMs), frameworks e recursos de SQL.

O Aurora DSQL gerencia automaticamente a infraestrutura do sistema e escala computação, E/S e armazenamento com base na workload. Como não há servidores para provisionar ou gerenciar, você não precisa se preocupar com o tempo de inatividade de manutenção relacionado a provisionamento, aplicação de patches ou atualizações de infraestrutura.

O Aurora DSQL ajuda você a criar e manter aplicações empresariais que estão sempre disponíveis em qualquer escala. O design ativo-ativo sem servidor automatiza a recuperação de falhas para que você não precise se preocupar com o failover de banco de dados tradicional. As aplicações se beneficiam da disponibilidade multi-AZ e multirregional, e você não precisa se preocupar com a consistência final ou com a falta de dados relacionados a failovers.

Principais recursos do Aurora DSQL

Os seguintes recursos principais ajudam você a criar um banco de dados distribuído sem servidor para atender a aplicações de alta disponibilidade:

Arquitetura distribuída

O Aurora DSQL é constituído dos seguintes componentes multilocatário:

- Retransmissão e conectividade
- Computação e bancos de dados

- Log de transações, controle de simultaneidade e isolamento
- Armazenamento

Um ambiente de gerenciamento coordena os componentes anteriores. Cada componente oferece redundância em três zonas de disponibilidade (AZs), com ajuste de escala automático de clusters e autorrecuperação em caso de falhas nos componentes. Para saber mais sobre como essa arquitetura atende à alta disponibilidade, consulte Resiliência no Amazon Aurora DSQL.

Clusters de região única e multirregionais

Os clusters do Aurora DSQL oferecem os seguintes benefícios:

- Replicação de dados síncrona
- Operações de leitura consistentes
- Recuperação automática de falhas
- Garantia de consistência de dados em várias AZs ou regiões

Se um componente de infraestrutura falhar, o Aurora DSQL encaminhará automaticamente as solicitações a uma infraestrutura íntegra sem intervenção manual. O Aurora DSQL fornece transações de atomicidade, consistência, isolamento e durabilidade (ACID) com alta consistência, isolamento de snapshots, atomicidade e durabilidade entre AZs e entre regiões.

Os clusters emparelhados multirregionais oferecem a mesma resiliência e conectividade que os clusters de região única. Mas eles melhoram a disponibilidade oferecendo dois endpoints regionais, um em cada região de cluster emparelhado. Ambos os endpoints de um cluster emparelhado apresentam um único banco de dados lógico. Eles estão disponíveis para operações simultâneas de leitura e gravação e oferecem alta consistência de dados. Você pode criar aplicações que são executadas em várias regiões ao mesmo tempo para melhorar o desempenho e a resiliência, e saiba que os leitores sempre veem os mesmos dados.

Compatibilidade com bancos de dados PostgreSQL

A camada de banco de dados distribuído (computação) no Aurora DSQL é baseada em uma versão principal atual do PostgreSQL. Você pode se conectar ao Aurora DSQL com drivers e ferramentas conhecidos do PostgreSQL, como o `psql`. No momento, o Aurora DSQL é compatível com o PostgreSQL versão 16 e aceita um subconjunto de recursos, expressões e tipos de dados do PostgreSQL. Para ter mais informações sobre os recursos com suporte, consulte Compatibilidade com recursos SQL no Aurora DSQL.

Disponibilidade de regiões para o Aurora DSQL

Com o Amazon Aurora DSQL, você pode implantar instâncias de banco de dados em várias Regiões da AWS para atender a aplicações globais e aos requisitos de residência de dados. A disponibilidade de regiões determina onde você pode criar e gerenciar clusters de banco de dados do Aurora DSQL. Administradores de banco de dados e arquitetos de aplicações que precisam projetar sistemas de banco de dados altamente disponíveis e globalmente distribuídos em geral precisam saber qual região atende às suas workloads. Os casos de uso comuns incluem configurar a recuperação de desastres entre regiões, atender usuários por meio de instâncias de banco de dados geograficamente mais próximas para reduzir a latência e manter cópias de dados em locais específicos para fins de conformidade.

A tabela a seguir mostra as Regiões da AWS em que o Aurora MySQL está disponível no momento e o endpoint para cada Região da AWS.

Regiões da AWS e endpoints com suporte

Nome da região	Região	Endpoint	Protocolo
Leste dos EUA (Norte da Virgínia)	us-east-1	dsql.us-east-1.api.aws	HTTPS
Leste dos EUA (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
Oeste dos EUA (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS
Europa (Londres)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europa (Irlanda)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europa (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
Ásia-Pacífico (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS
Ásia-Pacífico (Tóquio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS

> **Note**
>
> No momento, é possível usar clusters multirregionais do Aurora DSQL nas três seguintes Regiões da AWS:
>
> - Leste dos EUA (Norte da Virgínia)
> - Leste dos EUA (Ohio)
> - Oeste dos EUA (Oregon)

Preços do Aurora DSQL

Para obter informações sobre custos, consulte Aurora DSQL pricing.

Próximas etapas

Para ter informações sobre os componentes principais do Aurora DSQL e começar a usar o serviço, consulte o seguinte:

- Conceitos básicos do Aurora DSQL
- Compatibilidade com recursos SQL no Aurora DSQL
- Acessar o Aurora DSQL
- Aurora DSQL e PostgreSQL

Conceitos básicos do Aurora DSQL

O Amazon Aurora DSQL é um banco de dados relacional distribuído e sem servidor, otimizado para workloads transacionais. Nas seções a seguir, você aprenderá a criar clusters do Aurora DSQL de região única e multirregionais, conectar-se a eles e criar e carregar um esquema de exemplo. Você acessará clusters com o AWS Management Console e interagirá com o banco de dados usando o utilitário `psql`. Ao final, você terá um cluster do Aurora DSQL funcional configurado e pronto para uso em workloads de teste ou produção.

Tópicos

Pré-requisitos

Antes de começar a usar o Aurora DSQL, verifique se você atende aos seguintes pré-requisitos.

- Sua identidade do IAM deve ter permissão para fazer login no AWS Management Console.
- Sua identidade do IAM deve atender a um dos seguintes critérios:
  - Ter acesso para executar qualquer ação em qualquer recurso em sua Conta da AWS.
  - A permissão `iam:CreateServiceLinkedRole` do IAM e a possibilidade de obter acesso à ação de política do IAM `dsql:*`.
- Se você usar a AWS CLI em um ambiente semelhante ao Unix, o Python versão 3.8+ e o `psql` versão 14+ devem estar instalados. Para verificar as versões de sua aplicação, execute os comandos a seguir.

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

Se você usar a AWS CLI em um ambiente diferente, configure manualmente o Python versão 3.8+ e o `psql` versão 14+.

- Se você pretende acessar o Aurora DSQL usando o AWS CloudShell, o Python versão 3.8+ e o `psql` versão 14+ são fornecidos sem configuração adicional. Para ter mais informações sobre o AWS CloudShell, consulte What is AWS CloudShell?.
- Se você pretende acessar o Aurora DSQL usando uma GUI, use o DBeaver ou o JetBrains DataGrip. Para obter mais informações, consulte Acessar o Aurora DSQL com o DBeaver e Acessar o Aurora DSQL com o JetBrains DataGrip.

Acessar o Aurora DSQL

Você pode acessar o Aurora DSQL por meio das técnicas a seguir. Para saber como usar a CLI, as APIs e os SDKs, consulte Acessar o Aurora DSQL.

Tópicos

- Acessar o Aurora DSQL por meio do AWS Management Console
- Acessar o Aurora DSQL usando clientes SQL
- Usar o protocolo do PostgreSQL com o Aurora DSQL

Acessar o Aurora DSQL por meio do AWS Management Console

Você pode acessar o AWS Management Console para Aurora DSQL em https://console.aws.amazon.com/dsql.

Acessar o Aurora DSQL usando clientes SQL

O Aurora DSQL usa o protocolo padrão do PostgreSQL. Use o cliente interativo de sua preferência fornecendo um token de autenticação do IAM assinado como senha ao se conectar ao cluster. Um token de autenticação é uma string exclusiva de caracteres que o Aurora DSQL gera dinamicamente usando o AWS Signature Version 4.

O Aurora DSQL usa o token somente para autenticação. O token não afeta a conexão depois que ela é estabelecida. Se você tentar se reconectar usando um token expirado, a solicitação de conexão será negada. Para obter mais informações, consulte Gerar um token de autenticação no Amazon Aurora DSQL.

Tópicos

Acessar o Aurora DSQL com psql (terminal interativo do PostgreSQL)

O utilitário `psql` é uma interface de linha de comandos para o PostgreSQL. Ele permite que você digite consultas interativamente, as envie ao PostgreSQL e veja os resultados da consulta. Para melhorar os tempos de resposta das consultas, use o cliente PostgreSQL versão 17.

Baixe o instalador do seu sistema operacional na página PostgreSQL Downloads. Para ter mais informações sobre o `psql`, consulte https://www.postgresql.org/docs/current/app-psql.htm.

Se você já tem a AWS CLI instalada, use o exemplo a seguir para se conectar ao cluster. Você pode usar o AWS CloudShell, que vem com o `psql` pré-instalado, ou pode instalar o `psql` diretamente.

```
# 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
```

Acessar o Aurora DSQL com o DBeaver

DBeaver é uma ferramenta de banco de dados de código aberto baseada em GUI. Você pode usá-lo para se conectar e gerenciar seu banco de dados. Para baixar o DBeaver, consulte a página de

download no site da comunidade do DBeaver. As etapas a seguir explicam como se conectar ao seu cluster usando o DBeaver.

Como configurar uma nova conexão do Aurora DSQL no DBeaver

1. Escolha Nova conexão de banco de dados.
2. Na janela Nova conexão de banco de dados, escolha PostgreSQL.
3. Na guia Configurações de conexão/Principal, escolha Conectar por: host e insira as informações a seguir.

   - Host: use o endpoint do cluster.

     Banco de dados: insira `postgres`.

     Autenticação: escolha `Database Native`.

     Nome de usuário: insira `admin`.

     Senha: gere um token de autenticação. Copie o token gerado e use-o como senha.
4. Ignore quaisquer avisos e cole o token de autenticação no campo Senha do DBeaver.

> **Note**
>
> Você deve definir o modo SSL nas conexões de cliente. O Aurora DSQL permite o `SSLMODE=require`. O Aurora DSQL impõe a comunicação SSL do lado do servidor e rejeita conexões que não sejam SSL.

5. Para começar a executar instruções SQL, você deve conectar-se ao cluster.

> **Important**
>
> Os recursos administrativos fornecidos pelo DBeaver para os bancos de dados PostgreSQL (como o Gerenciador de Sessões e o Gerenciador de Bloqueios) não se aplicam a um banco de dados devido à sua arquitetura exclusiva. Embora acessíveis, essas telas não fornecem informações confiáveis sobre a integridade ou o status do banco de dados.

Credenciais de autenticação expiradas para o DBeaver

As sessões estabelecidas permanecem autenticadas por no máximo uma hora ou até que o DBeaver desconecte-se ou atinja o tempo limite. Para estabelecer novas conexões, forneça um token de autenticação válido no campo Senha de Configurações de conexão. Ao tentar abrir uma nova sessão (por exemplo, para listar novas tabelas ou um novo console do SQL), uma nova tentativa de autenticação é forçada. Se o token de autenticação configurado nas configurações em Configurações de conexão não for mais válido, essa nova sessão falhará e o DBeaver invalidará todas as sessões abertas anteriormente. Tenha isso em mente ao escolher a duração do token de autenticação do IAM com a opção `expires-in`.

Acessar o Aurora DSQL com o JetBrains DataGrip

O JetBrains DataGrip é um IDE multiplataforma para trabalhar com SQL e bancos de dados, incluindo o PostgreSQL. O DataGrip oferece uma interface gráfica robusta e um editor SQL inteligente. Para baixar o DataGrip, acesse a página de download no site da JetBrains.

Como configurar uma nova conexão do Aurora DSQL no JetBrains DataGrip

1. Escolha Nova fonte de dados e selecione PostgreSQL.
2. Na guia Fontes de Dados/Geral, insira as seguintes informações:
   - Host: use o endpoint do cluster.

     Porta: o Aurora DSQL usa o padrão do PostgreSQL: `5432`.

     Banco de dados: o Aurora DSQL usa o padrão do PostgreSQL: `postgres`.

     Autenticação: escolha `User & Password` .

     Nome de usuário: insira `admin`.

     Senha: gere um token e cole-o neste campo.

     URL: não modifique este campo. Ele será preenchido automaticamente com base nos outros campos.
3. Senha: forneça-a gerando um token de autenticação. Copie a saída resultante do gerador de token e cole-a no campo de senha.

> **Note**
>
> Você deve definir o modo SSL nas conexões de cliente. O Aurora DSQL permite o `PGSSLMODE=require`. O Aurora DSQL impõe a comunicação SSL do lado do servidor e rejeita conexões não SSL.

4. Para começar a executar instruções SQL, você deve conectar-se ao cluster.

> **Important**
>
> Algumas visualizações fornecidas pelo DataGrip para os bancos de dados PostgreSQL (como Sessões) não se aplicam a um banco de dados devido à sua arquitetura exclusiva. Embora acessíveis, essas telas não fornecem informações confiáveis sobre as sessões reais conectadas ao banco de dados.

Credenciais de autenticação expiradas

As sessões estabelecidas permanecem autenticadas por no máximo 1 hora ou até que ocorra uma desconexão explícita ou um tempo limite do lado do cliente. Se for necessário estabelecer novas conexões, um novo token de autenticação deverá ser gerado e fornecido no campo Senha de Propriedades da fonte de dados. Ao tentar abrir uma nova sessão (por exemplo, para listar novas tabelas ou um novo console do SQL), uma nova tentativa de autenticação é forçada. Se o token de autenticação configurado nas configurações de Conexão não for mais válido, essa nova sessão falhará e todas as sessões abertas anteriormente se tornarão inválidas.

Usar o protocolo do PostgreSQL com o Aurora DSQL

O PostgreSQL usa um protocolo baseado em mensagens para comunicação entre clientes e servidores. O protocolo é compatível com TCP/IP e também com sockets de domínio Unix. A tabela a seguir mostra como o protocolo do PostgreSQL pode ser usado com o Aurora DSQL.

PostgreSQL	Aurora DSQL	Observações
Perfil (também conhecido como usuário ou grupo)	Perfil de banco de dados	O Aurora DSQL cria um perfil chamado `admin` para você. Ao criar perfis de banco de dados personalizados, você deve usar

PostgreSQL	Aurora DSQL	Observações
		o perfil `admin` para associá-los aos perfis do IAM para autenticação ao se conectar ao cluster. Para ter mais informações, consulte Configure custom database roles.
Host (também conhecido como nome de host ou hostspec)	Endpoint do cluster	Os clusters de região única do Aurora DSQL oferecem um único endpoint gerenciado e redirecionam automaticamente o tráfego se houver indisponibilidade na região.
Porta	N/D: use o padrão `5432`.	Esse é o padrão do PostgreSQL.
Banco de dados (dbname)	Use `postgres`.	O Aurora DSQL cria esse banco de dados para você quando você cria o cluster.
Modo SSL	O SSL está sempre habilitado do lado do servidor.	No Aurora DSQL, o Aurora DSQL é compatível com o modo SSL `require`. Conexões sem SSL são rejeitadas pelo Aurora DSQL.
Senha	Token de autenticação	O Aurora DSQL exige tokens de autenticação temporários em vez de senhas de longa duração. Para saber mais, consulte Gerar um token de autenticação no Amazon Aurora DSQL.

Etapa 1: criar um cluster do Aurora DSQL de região única

A unidade básica do Aurora DSQL é o cluster, onde você armazena seus dados. Nesta tarefa, você cria um cluster em uma única Região da AWS.

Como criar um cluster de região única no Aurora DSQL

1. Faça login no AWS Management Console e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.

2. Escolha Criar cluster e Região única.
3. (Opcional) Em Configurações de cluster, selecione qualquer uma das seguintes opções:
   - Selecione Configurações de criptografia personalizadas (avançado) para escolher ou criar uma AWS KMS key.
   - Selecione Habilitar proteção contra exclusão para impedir que o cluster de banco de dados seja excluído. Por padrão, a proteção contra exclusão fica selecionada.
4. (Opcional) Em Tags, escolha ou insira uma tag para o cluster.
5. Selecione Criar cluster.

Etapa 2: conectar-se a um cluster do Aurora DSQL

Um endpoint de cluster é gerado automaticamente quando você cria um cluster do Aurora DSQL com base no ID do cluster e na região. O formato de nomenclatura é `clusterid.dsql.region.on.aws`. Um cliente usa o endpoint para criar uma conexão de rede com seu cluster.

Como a autenticação é gerenciada por meio do IAM, você não precisa armazenar credenciais no banco de dados. Um token de autenticação é uma string exclusiva de caracteres que é gerada dinamicamente. O token é usado apenas para autenticação e não afetará a conexão depois que ela for estabelecida. Antes de tentar se conectar, sua identidade do IAM deve ter a permissão `dsql:DbConnectAdmin`, conforme descrito em Pré-requisitos.

> **Note**
>
> Para otimizar a velocidade de conexão do banco de dados, use o cliente PostgreSQL versão 17 e defina `PGSSLNEGOTIATION` como direct: `PGSSLNEGOTIATION=direct`.

Como se conectar ao cluster com um token de autenticação

1. Selecione o cluster de banco de dados do Aurora MySQL ao qual você deseja se conectar.
2. Selecione Conectar.
3. Em Endpoint (host), copie o endpoint.
4. A opção Conectar-se como administrado deve estar selecionada na seção Token de autenticação (senha).

5. Gere o token de autenticação. O token é válido por 15 minutos.
6. Na linha de comando, use o comando a seguir para iniciar o `psql` e se conectar ao cluster. Substitua *`your_cluster_endpoint`* pelo endpoint do cluster que você copiou anteriormente.

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

   Quando for solicitada uma senha, insira o token de autenticação que você copiou anteriormente. Se você tentar se reconectar usando um token expirado, a solicitação de conexão será negada. Para obter mais informações, consulte Gerar um token de autenticação no Amazon Aurora DSQL.
7. Pressione Enter. Você verá um prompt do PostgreSQL.

```
postgres=>
```

   Se você receber um erro de acesso negado, confirme se sua identidade do IAM tenha a permissão `dsql:DbConnectAdmin`. Se você tiver a permissão e continuar recebendo erros de negação de acesso, consulte Solucionar problemas do IAM e Como solucionar erros de acesso negado ou operação não autorizada em uma política do IAM?.

Etapa 3: executar exemplos de comando SQL no Aurora DSQL

Teste o cluster do Aurora DSQL executando instruções SQL. Os exemplos de instrução a seguir exigem os arquivos de dados chamados `department-insert-multirow.sql` e `invoice.csv`, que você pode baixar do repositório aws-samples/aurora-dsql-samples no GitHub.

Como executar exemplos de comando SQL no Aurora DSQL

1. Crie um esquema chamado `example`.

```
CREATE SCHEMA example;
```

2. Crie uma tabela de faturas que use um UUID gerado automaticamente como chave primária.

```
CREATE TABLE example.invoice(
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
    created timestamp,
    purchaser int,
    amount float);
```

3. Crie um índice secundário que use a tabela vazia.

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

4. Crie uma tabela de departamentos.

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

5. Use o comando `psql \include` para carregar o arquivo chamado `department-insert-multirow.sql` que você baixou do repositório aws-samples/aurora-dsql-samples no GitHub. Substitua *my-path* pelo caminho para sua cópia local.

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

6. Use o comando `psql \copy` para carregar o arquivo chamado `invoice.csv` que você baixou do repositório aws-samples/aurora-dsql-samples no GitHub. Substitua *my-path* pelo caminho para sua cópia local.

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

7. Consulte os departamentos e classifique-os pelo total de vendas.

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

O exemplo de saída a seguir mostra que o Departamento Três tem o maior volume de vendas.

```
           name           |     sum_amount
--------------------------+--------------------
 Example Department Three |  54061.67752854594
 Example Department Seven |  53869.65965365204
 Example Department Eight |  52199.73742066634
 Example Department One   | 52034.078869900826
```

```
 Example Department Six   |  50886.15556256385
 Example Department Two   |  50589.98422247931
 Example Department Five  | 49549.852635496005
 Example Department Four  |  49266.15578027619
(8 rows)
```

Etapa 4: criar um cluster multirregional

Ao criar um cluster multirregional, você deve especificar as seguintes regiões:

Região remota

Essa é a região na qual você cria um segundo cluster. Você cria um segundo cluster nessa região e o conecta ao cluster inicial. O Aurora DSQL replica todas as gravações no cluster inicial para o cluster remoto. Você pode ler e gravar em qualquer cluster.

Região testemunha

Essa região recebe todos os dados gravados no cluster multirregional. Entretanto, as regiões testemunha não hospedam endpoints de clientes e não oferecem acesso aos dados do usuário. Uma janela limitada de logs de transações criptografados é mantida nas regiões testemunha. Isso facilita a recuperação e permite um quórum transacional caso uma região fique indisponível.

O exemplo a seguir mostra como criar um cluster inicial, criar um segundo cluster em uma região diferente e, em seguida, conectar os dois clusters para criar um cluster multirregional. Demonstra também a replicação de gravação entre regiões e leituras consistentes de ambos os endpoints regionais.

Como criar um cluster multirregional

1. Faça login no AWS Management Console e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.
2. No painel de navegação, escolha Clusters.
3. Escolha Criar cluster e Multirregião.
4. (Opcional) Em Configurações de cluster, selecione qualquer uma das seguintes opções para o cluster inicial:

   - Selecione Configurações de criptografia personalizadas (avançado) para escolher ou criar uma AWS KMS key.

- Selecione Habilitar proteção contra exclusão para impedir que o cluster de banco de dados seja excluído. Por padrão, a proteção contra exclusão fica selecionada.

5. Em Configurações multirregionais, escolha as seguintes opções para o cluster inicial:

   - Em Região testemunha, escolha uma região. No momento, somente regiões baseadas nos EUA são aceitas para regiões testemunha em clusters multirregionais.
   - (Opcional) Em ARN do cluster da região remota, insira um ARN para um cluster existente em outra região. Se não houver nenhum cluster para servir como segundo cluster no cluster multirregional, conclua a configuração depois de criar o cluster inicial.

6. (Opcional) Escolha tags para o cluster inicial.
7. EEscolha Criar cluster para criar o cluster inicial. Se você não inseriu um ARN na etapa anterior, o console mostrará a notificação Configuração do cluster pendente.
8. Na notificação Configuração do cluster pendente, escolha Concluir configuração do cluster multirregional. Essa ação inicia a criação de um segundo cluster em outra região.
9. Escolha uma das seguintes opções para o segundo cluster:

   - Adicionar ARN do cluster da região remota: escolha essa opção se houver um cluster e você quiser que ele seja o segundo no cluster multirregional.
   - Criar cluster em outra região: escolha essa opção para criar um segundo cluster. Em Região remota, escolha a região para esse segundo cluster.

10. Escolha Criar cluster na ***your-second-region***, em que *your-second-region* é o local do segundo cluster. O console abre na segunda região.
11. (Opcional) Escolha as configurações de cluster para o segundo cluster. Por exemplo, você pode escolher uma AWS KMS key.
12. Escolha Criar cluster para criar o segundo cluster.
13. Escolha Emparelhar na ***initial-cluster-region***, em que *initial-cluster-region* é a região que hospeda o primeiro cluster que você criou.
14. Quando solicitado, selecione Sim para confirmar. Essa etapa conclui a criação do cluster multirregional.

Como se conectar e usar o cluster

1. Abra o console do Aurora DSQL e escolha a região para o segundo cluster.
2. Escolha Clusters.

3. Selecione a linha para o segundo cluster no cluster multirregional.
4. Em Ações, escolha Abrir no CloudShell.
5. Escolha Conectar-se como administrador.
6. Escolha Inicie o CloudShell.
7. Escolha Executar.
8. Crie um esquema de exemplo seguindo as etapas em Etapa 3: executar exemplos de comando SQL no Aurora DSQL.

   Exemplos de transação

   Example

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

9. Use comandos `copy` e `include` do `psql` para carregar dados de amostra. Para obter mais informações, consulte Etapa 3: executar exemplos de comando SQL no Aurora DSQL.

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

Como consultar dados no segundo cluster na região que hospeda o cluster inicial

1. No console do Aurora DSQL, escolha a região do cluster inicial.
2. Escolha Clusters.
3. Selecione a linha para o segundo cluster no cluster multirregional.
4. Em Ações, escolha Abrir no CloudShell.
5. Escolha Conectar-se como administrador.
6. Escolha Inicie o CloudShell.
7. Escolha Executar.
8. Consulte os dados que você inseriu no segundo cluster.

Example

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

Autenticação e autorização para o Aurora DSQL

O Aurora DSQL usa perfis e políticas do IAM para autorização de clusters. Você associa perfis do IAM a perfis do banco de dados PostgreSQL para autorização do banco de dados. Essa abordagem combina os benefícios do IAM com os privilégios do PostgreSQL. O Aurora DSQL usa esses recursos para oferecer uma política abrangente de autorização e acesso para o cluster, o banco de dados e os dados.

Gerenciar clusters usando o IAM

Para gerenciar clusters, use o IAM para autenticação e autorização:

Autenticação do IAM

Para autenticar sua identidade do IAM ao gerenciar clusters do Aurora DSQL, você deve usar o IAM. Você pode fornecer autenticação usando o AWS Management Console, a AWS CLI ou o SDK da AWS.

Autorização do IAM

Para gerenciar clusters do Aurora DSQL, conceda autorização usando ações do IAM para o Aurora DSQL. Por exemplo, para criar um cluster, sua identidade do IAM deve ter permissões para a ação `dsql:CreateCluster` do IAM, como no exemplo de ação de política a seguir.

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

Para obter mais informações, consulte Usar ações de política do IAM para gerenciar clusters.

Conectar-se a um cluster usando o IAM

Para se conectar ao seu cluster, use o IAM para autenticação e autorização:

Autenticação do IAM

Gere um token de autenticação temporário usando uma identidade do IAM com autorização para conexão com o cluster. Para saber mais, consulte Gerar um token de autenticação no Amazon Aurora DSQL.

Autorização do IAM

Conceda as seguintes ações de política do IAM à identidade do IAM que você está usando para estabelecer a conexão com o endpoint do cluster:

- Use `dsql:DbConnectAdmin` se você estiver usando o perfil `admin`. O Aurora DSQL cria e gerencia esse perfil para você. O exemplo de ação de política do IAM a seguir permite que o `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` se você estiver utilizando um perfil de banco de dados personalizado. Você cria e gerencia esse perfil usando comandos SQL em seu banco de dados. O exemplo de ação de política do IAM a seguir permite que um perfil de banco de dados personalizado fique conectado a *`my-cluster`* por até uma hora.

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

Depois de estabelecer uma conexão, seu é perfil tem autorização para ficar conectado por até uma hora.

Interagir com o banco de dados usando perfis de banco de dados do PostgreSQL e perfis do IAM

O PostgreSQL gerencia as permissões de acesso ao banco de dados usando o conceito de perfis. Um perfil pode ser considerado como um usuário ou um grupo de usuários do banco de dados,

dependendo de como ele está configurado. Você cria perfis do PostgreSQL usando comandos SQL. Para gerenciar a autorização em nível de banco de dados, conceda permissões do PostgreSQL aos seus perfis de banco de dados do PostgreSQL.

O Aurora DSQL permite dois tipos de perfil de banco de dados: um perfil `admin` e perfis personalizados. O Aurora DSQL cria automaticamente um perfil `admin` predefinido para você no cluster do Aurora DSQL. Não é possível modificar o perfil `admin`. Ao se conectar ao banco de dados como `admin`, você pode emitir SQL para criar perfis no nível do banco de dados e associá-los a seus perfis do IAM. Para permitir que os perfis do IAM se conectem ao banco de dados, associe seus perfis de banco de dados personalizados aos seus perfis do IAM.

Autenticação

Use o perfil `admin` para se conectar ao seu cluster. Depois de conectar seu banco de dados, use o comando `AWS IAM GRANT` para associar um perfil de banco de dados personalizado à identidade do IAM autorizada a se conectar ao cluster, como no exemplo a seguir.

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

Para saber mais, consulte Autorizar perfis de banco de dados a se conectarem ao cluster.

Autorização

Use o perfil `admin` para se conectar ao seu cluster. Execute comandos SQL para configurar perfis de banco de dados personalizados e conceder permissões. Para saber mais, consulte as funções de banco de dados do PostgreSQL e os privilégios do PostgreSQL na documentação do PostgreSQL.

Usar ações de política do IAM com o Aurora DSQL

A ação de política do IAM que você usa depende do perfil usado para se conectar ao cluster: um perfil `admin` ou um perfil de banco de dados personalizado. A política também depende das ações do IAM necessárias para esse perfil.

Usar ações de política do IAM para se conectar a clusters

Ao se conectar ao cluster com um perfil de banco de dados padrão de `admin`, use uma identidade do IAM com autorização para realizar a ação de política do IAM a seguir.

```
"dsql:DbConnectAdmin"
```

Ao se conectar ao cluster com um perfil de banco de dados personalizado, primeiro associe o perfil do IAM ao perfil de banco de dados. A identidade do IAM que você usa para se conectar ao cluster deve ter autorização para realizar a ação de política do IAM a seguir.

```
"dsql:DbConnect"
```

Para saber mais sobre perfis de banco de dados personalizados, consulte Usar perfis de banco de dados e autenticação do IAM.

Usar ações de política do IAM para gerenciar clusters

Ao gerenciar clusters do Aurora DSQL, especifique ações de política somente para as ações que seu perfil precisa realizar. Por exemplo, se seu perfil precisar apenas obter informações sobre o cluster, você pode limitar as permissões de perfil somente às permissões `GetCluster` e `ListClusters`, como no exemplo de política a seguir

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

O exemplo de política a seguir mostra todas as ações de política do IAM disponíveis para gerenciar clusters.

```
{
"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" : "*"
    }
  ]
}
```

Revogar a autorização usando o IAM e o PostgreSQL

Você pode revogar as permissões de seus perfis do IAM para acessar seus perfis no nível do banco de dados:

Revogar a autorização do administrador para se conectar aos clusters

Para revogar a autorização para se conectar ao cluster com um perfil `admin`, revogue o acesso da identidade do IAM a `dsql:DbConnectAdmin`. Edite a política do IAM ou desanexe a política da identidade.

Depois de revogar a autorização de conexão da identidade do IAM, o Aurora DSQL rejeita todas as novas tentativas de conexão dessa identidade do IAM. A autorização de conexões ativas que estejam usando a identidade do IAM pode ser mantida pelo tempo da conexão. Para obter mais informações sobre a duração das conexões, consulte Cotas e limites.

Revogar a autorização do perfil personalizado para se conectar aos clusters

Para revogar o acesso a outros perfis de banco de dados além de `admin`, revogue o acesso da identidade do IAM a `dsql:DbConnect`. Edite a política do IAM ou desanexe a política da identidade.

Você também pode remover a associação entre um perfil do banco de dados e o IAM usando o comando `AWS IAM REVOKE` no banco de dados. Para saber mais sobre a revogação do acesso de perfis de banco de dados, consulte Revogar a autorização do banco de dados de um perfil do IAM.

Não é possível gerenciar as permissões do perfil de banco de dados `admin` predefinido. Para saber como gerenciar permissões para perfis de banco de dados personalizados, consulte os privilégios do PostgreSQL. As modificações nos privilégios entrarão em vigor na próxima transação depois que o Aurora DSQL confirmar com sucesso a transação de modificação.

Gerar um token de autenticação no Amazon Aurora DSQL

Para se conectar ao Amazon Aurora DSQL com um cliente SQL, gere um token de autenticação para usar como senha. Esse token é usado somente para autenticar a conexão. Depois que a conexão é estabelecida, a conexão permanece válida mesmo que o token de autenticação expire.

Se você criar um token de autenticação usando o console da AWS, ele expirará automaticamente em uma hora por padrão. Se você usar a AWS CLI ou SDKs para criar o token, o padrão será 15 minutos. O máximo é 604.800 segundos, o que equivale a uma semana. Para se conectar novamente ao Aurora DSQL por meio do seu cliente, você pode usar o mesmo token de autenticação, caso ele não tenha expirado, ou pode gerar um novo.

Para começar a gerar um token, crie uma política do IAM e um cluster no Aurora DSQL. Em seguida, use o console da AWS, a AWS CLI ou os SDKs da AWS para gerar um token.

No mínimo, você deve ter as permissões do IAM listadas em Conectar-se a um cluster usando o IAM, dependendo do perfil de banco de dados que você usa para se conectar.

Tópicos

- Usar o console da AWS para gerar um token de autenticação no Aurora DSQL
- Usar a AWS CloudShell para gerar um token de autenticação no Aurora DSQL
- Usar a AWS CLI para gerar um token de autenticação no Aurora DSQL
- Usar os SDKs para gerar um token no Aurora DSQL

Usar o console da AWS para gerar um token de autenticação no Aurora DSQL

O Aurora DSQL autentica os usuários com um token, em vez de uma senha. Você pode gerar o token no console.

Gerar token de autenticação

1. Faça login no AWS Management Console e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.
2. Escolha o ID do cluster para o qual deseja gerar um token de autenticação. Se você ainda não criou um cluster, siga as etapas em Etapa 1: criar um cluster do Aurora DSQL de região única ou Etapa 4: criar um cluster multirregional.
3. Escolha Connect e, em seguida, selecione Get Token.
4. Escolha se você deseja se conectar como `admin` ou com um perfil de banco de dados personalizado.
5. Copie o token de autenticação gerado e use-o para Acessar o Aurora DSQL usando clientes SQL.

Para saber mais sobre perfis de banco de dados personalizados e o IAM no Aurora DSQL, consulte Autenticação e autorização.

Usar a AWS CloudShell para gerar um token de autenticação no Aurora DSQL

Antes de gerar um token de autenticação usando o AWS CloudShell, faça o seguinte:

- Crie um cluster do Aurora DSQL.
- Adicione uma permissão para executar a operação do Amazon S3 `get-object` e recuperar objetos de uma Conta da AWS fora da sua organização. Para obter mais informações, consulte o Guia do usuário do Amazon S3.

Como gerar token de autenticação usando o AWS CloudShell

1. Faça login no AWS Management Console e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.
2. Na parte inferior esquerda do Console da AWS, escolha AWS CloudShell.
3. Siga Instalar ou atualizar a versão mais recente da AWS CLI para instalar a AWS CLI.

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

4. Execute o comando a seguir para gerar um token de autenticação para o perfil `admin`. Substitua *`us-east-1`* pela sua região e *`cluster_endpoint`* pelo endpoint do seu cluster.

> **Note**
>
> Se você não estiver se conectando como `admin`, use `generate-db-connect-auth-token` em vez disso.

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

Se você tiver problemas, consulte Solucionar problemas do IAM e Como solucionar erros de acesso negado ou operação não autorizada em uma política do IAM?.

5. Use o comando a seguir para utilizar o `psql` para iniciar uma conexão com o cluster.

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

6. Você verá uma solicitação para fornecer uma senha. Copie o token que você gerou e não inclua espaços ou caracteres adicionais. Cole-o no prompt a seguir do `psql`.

```
Password for user admin:
```

7. Pressione Enter. Você verá um prompt do PostgreSQL.

```
postgres=>
```

Se você receber um erro de acesso negado, confirme se sua identidade do IAM tenha a permissão `dsql:DbConnectAdmin`. Se você tiver a permissão e continuar recebendo erros de negação de acesso, consulte Solucionar problemas do IAM e Como solucionar erros de acesso negado ou operação não autorizada em uma política do IAM?.

Para saber mais sobre perfis de banco de dados personalizados e o IAM no Aurora DSQL, consulte Autenticação e autorização.

Usar a AWS CLI para gerar um token de autenticação no Aurora DSQL

Quando seu cluster estiver `ACTIVE`, você poderá gerar um token de autenticação na CLI usando o comando `aws dsql`. Use uma das seguintes técnicas:

- Se você estiver se conectando com o perfil `admin`, use a opção `generate-db-connect-admin-auth-token`.
- Se você estiver se conectando com um perfil de banco de dados personalizado, use a opção `generate-db-connect-auth-token`.

O exemplo a seguir usa os atributos abaixo para gerar um token de autenticação para o perfil `admin`.

- *`your_cluster_endpoint`*: o endpoint do cluster. Ele segue o formato *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, como no exemplo `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*: a Região da AWS, como `us-east-2` ou `us-east-1`.

Os exemplos a seguir definem que o token deve expirar em 3.600 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
```

Usar os SDKs para gerar um token no Aurora DSQL

Você pode gerar um token de autenticação para o cluster quando ele estiver no status `ACTIVE`. Os exemplos de SDK usam os seguintes atributos para gerar um token de autenticação para o perfil `admin`:

- *`your_cluster_endpoint`* (ou *`yourClusterEndpoint`*): o endpoint do cluster do Aurora DSQL. O formato de nomenclatura é *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, como no exemplo `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`* (ou *`RegionEndpoint`*): a Região da AWS em que o cluster está localizado, como `us-east-2` ou `us-east-1`.

Python SDK

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `generate_db_connect_admin_auth_token`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `generate_connect_auth_token`.

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

C++ SDK

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `GenerateDBConnectAdminAuthToken`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `GenerateDBConnectAuthToken`.

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

using namespace Aws;
using namespace Aws::DSQL;

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

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

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

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

JavaScript SDK

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `getDbConnectAdminAuthToken`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `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

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `generateDbConnectAdminAuthToken`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `generateDbConnectAuthToken`.

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

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

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

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

Rust SDK

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `db_connect_admin_auth_token`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `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

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `generate_db_connect_admin_auth_token`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `generate_db_connect_auth_token`.

```
require 'aws-sdk-dsql'

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

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

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

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

.NET

> **Note**
>
> O SDK oficial para .NET não inclui uma chamada de API integrada para gerar um token de autenticação para o Aurora DSQL. Em vez disso, você deve usar `DSQLAuthTokenGenerator`, que é uma classe de utilitário. O exemplo de código a seguir mostra como gerar o token de autenticação para .NET.

Você pode gerar o token das seguintes maneiras:

- Se você estiver se conectando com um perfil `admin`, use `DbConnectAdmin`.
- Se você estiver se conectando com um perfil de banco de dados personalizada, use `DbConnect`.

O exemplo a seguir usa a classe utilitária `DSQLAuthTokenGenerator` para gerar o token de autenticação para um usuário com o perfil `admin`. Substitua *`insert-dsql-cluster-endpoint pelo endpoint do cluster`*.

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

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

AWSCredentials credentials = FallbackCredentialsFactory.GetCredentials();

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

Console.WriteLine(token);
```

Golang

> **Note**
>
> O SDK do Golang não fornece um método integrado para gerar um token pré-assinado. Você deve construir a solicitação assinada manualmente, conforme mostrado no exemplo de código a seguir.

No exemplo de código a seguir, especifique o `action` com base no usuário do PostgreSQL:

- Se você estiver se conectando com um perfil `admin`, use a ação `DbConnectAdmin`.
- Se você estiver se conectando com um perfil de banco de dados personalizado, use a ação `DbConnect`.

Além de *`yourClusterEndpoint`* e *`region`*, o exemplo a seguir usa *`action`*. Especifique a *`ação`* com base no usuário do PostgreSQL.

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

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

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

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

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

 return url, nil
}
```

Usar perfis de banco de dados e autenticação do IAM

O Aurora DSQL permite autenticação usando perfis do IAM e usuários do IAM. Você pode usar qualquer um dos métodos para autenticar e acessar bancos de dados do Aurora DSQL.

Perfis do IAM

Um perfil do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas, mas não está associada a uma pessoa específica. Ao usar perfis do IAM, você tem acesso a credenciais de segurança temporárias. Você pode assumir temporariamente um perfil do IAM de várias maneiras:

- Trocando de perfil no AWS Management Console.
- Chamando uma operação da AWS CLI ou da API da AWS.
- Usando um URL personalizado.

Depois de assumir um perfil, você pode acessar o Aurora DSQL usando as respectivas credenciais temporárias. Para ter mais informações sobre métodos para o uso de perfis, consulte Identidades do IAM no Guia do usuário do IAM.

Usuários do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas e está associada a uma única pessoa ou aplicação. Os usuários do IAM têm credenciais de longo prazo, como senhas e chaves de acesso, que podem ser usadas para acessar o Aurora DSQL.

> **Note**
>
> Para executar comandos SQL com a autenticação do IAM, você pode usar ARNs de perfil do IAM ou ARNs de usuário do IAM nos exemplos abaixo.

Autorizar perfis de banco de dados a se conectarem ao cluster

Crie um perfil do IAM e conceda autorização de conexão com a seguinte ação de política do IAM: `dsql:DbConnect`.

A política do IAM também deve conceder permissão de acesso aos recursos do cluster. Use um curinga (*) ou siga as instruções em Usar chaves de condição do IAM com o Amazon Aurora DSQL.

Autorizar perfis de banco de dados a usar SQL no banco de dados

Você deve usar um perfil do IAM com autorização para se conectar ao seu cluster.

1. Conecte-se ao seu cluster do Aurora DSQL usando um utilitário do SQL.

   Use um perfil de banco de dados `admin` com uma identidade do IAM autorizada para a ação `dsql:DbConnectAdmin` do IAM para se conectar ao cluster.
2. Crie um perfil de banco de dados, certificando-se de especificar a opção `WITH LOGIN`.

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

3. Associe o perfil de banco de dados ao ARN do perfil do IAM.

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

4. Conceder permissões em nível de banco de dados ao perfil de banco de dados

   Os exemplos a seguir usam o comando `GRANT` para fornecer autorização no banco de dados.

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

Para ter mais informações, consulte PostgreSQL `GRANT` e os PostgreSQL Privileges na documentação do PostgreSQL.

Revogar a autorização do banco de dados de um perfil do IAM

Para revogar a autorização do banco de dados, use a operação `AWS IAM REVOKE`.

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

Para saber mais sobre como revogar a autorização, consulte Revogar a autorização usando o IAM e o PostgreSQL.

Aurora DSQL e PostgreSQL

O Aurora DSQL é um banco de dados relacional distribuído, compatível com PostgreSQL, projetado para workloads transacionais. O Aurora DSQL usa os principais componentes do PostgreSQL, como o analisador, planejador, otimizador e sistema de tipos.

O design do Aurora DSQL garante que a sintaxe completa compatível do PostgreSQL apresente um comportamento compatível e produza resultados de consulta idênticos. Por exemplo, o Aurora DSQL fornece conversões de tipo, operações aritméticas e precisão e escala numéricas que são idênticas às do PostgreSQL. Quaisquer desvios são documentados.

O Aurora DSQL também introduz recursos avançados, como controle otimista de simultaneidade e gerenciamento de esquemas distribuídos. Com esses recursos, você pode usar as ferramentas conhecidas do PostgreSQL, bem como se beneficiar do desempenho e da escalabilidade essenciais para aplicações modernas e nativas da nuvem.

Destaques da compatibilidade com o PostgreSQL

Atualmente, o Aurora DSQL baseia-se na versão 16 do PostgreSQL. As principais compatibilidades incluem:

Protocolo de comunicação

O Aurora DSQL usa o protocolo de comunicação padrão PostgreSQL v3. Isso permite a integração com clientes, drivers e ferramentas padrão do PostgreSQL. Por exemplo, o Aurora DSQL é compatível com `psql`, `pgjdbc` e `psycopg`.

Compatibilidade com SQL

O Aurora DSQL oferece suporte a uma ampla variedade de expressões e funções padrão do PostgreSQL comumente usadas em workloads transacionais. As expressões SQL compatíveis produzem resultados idênticos aos do PostgreSQL, incluindo os seguintes:

- Manipulação de nulos
- Comportamento da ordem de classificação
- Escala e precisão para operações numéricas
- Equivalência para operações de string

Para obter mais informações, consulte Compatibilidade com recursos SQL no Aurora DSQL.

Gerenciamento de transações

O Aurora DSQL preserva as características principais do PostgreSQL, como transações ACID e um nível de isolamento equivalente ao Repeatable Read do PostgreSQL. Para obter mais informações, consulte Controle de simultaneidade no Aurora DSQL.

Principais diferenças de arquitetura

O design distribuído e sem compartilhamento do Aurora DSQL resulta em algumas diferenças fundamentais em relação ao PostgreSQL tradicional. Essas diferenças são intrínseca à arquitetura do Aurora DSQL e oferecem muitos benefícios de desempenho e escalabilidade. As principais diferenças incluem:

Controle de simultaneidade otimista (OCC)

O Aurora DSQL usa um modelo de controle de simultaneidade otimista. Essa abordagem sem bloqueios evita que as transações bloqueiem umas às outras, elimina deadlocks e permite a execução paralela de alto throughput. Esses recursos tornam o Aurora DSQL particularmente valioso para aplicações que exigem desempenho consistente em escala. Para obter mais exemplos, consulte Controle de simultaneidade no Aurora DSQL.

Operações assíncronas de DDL

O Aurora DSQL executa operações de DDL de forma assíncrona, o que permite leituras e gravações ininterruptas durante alterações de esquema. Sua arquitetura distribuída permite que o Aurora DSQL faça o seguinte:

- Execute operações de DDL como tarefas em segundo plano, minimizando interrupções.
- Coordene alterações do catálogo como transações distribuídas altamente consistentes. Isso garante visibilidade atômica em todos os nós, mesmo durante falhas ou operações concorrentes.
- Opere de forma totalmente distribuída e sem líderes em várias zonas de disponibilidade com camadas de computação e armazenamento desacopladas.

Para obter mais informações, consulte DDL e transações distribuídas no Aurora DSQL.

Compatibilidade com recursos SQL no Aurora DSQL

O Aurora DSQL e o PostgreSQL exibem resultados idênticos para todas as consultas SQL. Observe que o Aurora DSQL difere do PostgreSQL sem uma cláusula `ORDER BY`. Nas seções a seguir, saiba mais sobre a compatibilidade do Aurora DSQL com tipos de dados e comandos SQL do PostgreSQL.

Tópicos

Tipos de dados compatíveis no Aurora DSQL

O Aurora DSQL aceita um subconjunto dos tipos comuns do PostgreSQL.

Tópicos

Tipos de dados numéricos

O Aurora DSQL aceita os tipos de dados numéricos a seguir do PostgreSQL.

Name	Aliases	Intervalo e precisão	Tamanho de armazenamento	Suporte a índice
`smallint`	`int2`	-32768 a +32767	2 bytes	Sim
`integer`	`int`, `int4`	-2147483648 a +2147483647	4 bytes	Sim

Name	Aliases	Intervalo e precisão	Tamanho de armazenamento	Suporte a índice
`bigint`	`int8`	-9223372036854775808 a +9223372036854775807	8 bytes	Sim
`real`	`float4`	Precisão de 6 dígitos decimais	4 bytes	Sim
`double precision`	`float8`	Precisão de 15 dígitos decimais	8 bytes	Sim

`numeric [ (p, s) ]`	`decimal [ (p, s) ]`
	
	`dec[ (p,s)]`
		Numérico exato com precisão selecionável. A precisão máxima é 38 e a escala máxima é 37[2] O padrão é `numeric (18,6)`.	8 bytes + 2 bytes por dígito de precisão. O tamanho máximo para é 27 bytes.	Não

[2] Se você não especificar explicitamente um tamanho ao executar `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, o Aurora DSQL aplicará os padrões. O Aurora DSQL aplica limites quando você executa instruções `INSERT` ou `UPDATE`.

Tipos de dados de caracteres

O Aurora DSQL aceita os tipos de dados de caracteres a seguir do PostgreSQL.

Name	Aliases	Descrição	Limite do Aurora DSQL	Tamanho de armazenamento	Suporte a índice
`character [ (n) ]`	`char [ (n) ]`	String de caracteres com comprimento fixo	4.096 bytes[1]	Variável até 4.100 bytes	Sim

Name	Aliases	Descrição	Limite do Aurora DSQL	Tamanho de armazenamento	Suporte a índice
`character varying [ (n) ]`	`varchar [ (n) ]`	String de caracteres de comprimento variável	65.535 bytes[1]	Variável até 65.539 bytes	Sim
`bpchar [ (n) ]`		Se for de tamanho fixo, esse é um alias para `char`. Se o comprimento for variável, esse é um alias para `varchar`, em que os espaços à direita são semanticamente insignificantes.	4.096 bytes[1]	Variável até 4.100 bytes	Sim
`text`		String de caracteres de comprimento variável	1 MiB[1]	Variável até 1 MiB	Sim

[1]: se você não especificar explicitamente um tamanho ao executar `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, o Aurora DSQL aplicará os padrões. O Aurora DSQL aplica limites quando você executa instruções `INSERT` ou `UPDATE`.

Tipos de dados de data e hora

O Aurora DSQL é compatível com os tipos de dados de data e hora a seguir do PostgreSQL.

Name	Aliase	Descrição	Intervalo	Resolução	Taman de armaze ento	Suporte a índice
`date`		Data de calendário (ano, mês, dia)	4713 a.C.-5874897 d.C	1 dia	4 bytes	Sim
`time [ (p) ] [ without time zone ]`	`time`	Hora do dia sem fuso horário	0-1	1 microsseg undo	8 bytes	Sim
`time [ (p) ] with time zone`	`timet`	Hora do dia, incluindo fuso horário	00:00:00+1559 a 24:00:00-1559	1 microsseg undo	12 byte	Não
`timestamp [ (p) ] [ without time zone ]`		Data e hora, sem fuso horário	4713 a.C.-294276 d.C	1 microsseg undo	8 bytes	Sim
`timestamp [ (p) ] with time zone`	`times tz`	Data e hora, incluindo fuso horário	4713 a.C.-294276 d.C	1 microsseg undo	8 bytes	Sim
`interval [ fields ] [ (p) ]`		Intervalo de tempo	-178.000.000 anos a 178.000.000 anos	1 microsseg undo	16 byte	Não

Tipos de dados diversos

O Aurora DSQL é compatível com os tipos de dados diversos a seguir do PostgreSQL.

Name	Aliases	Descrição	Limite do Aurora DSQL	Tamanho de armazenamento	Suporte a índice
`boolean`	`bool`	Booleanos lógicos (verdadeiro/falso)		1 byte	Sim
`bytea`		Dados binários (“byte array”)	1 MiB[1]	Variável até o limite de 1 MiB	Não
`UUID`		Identificador universal único		16 bytes	Sim

[1]: se você não especificar explicitamente um tamanho ao executar `CREATE TABLE` ou `ALTER TABLE ADD COLUMN`, o Aurora DSQL aplicará os padrões. O Aurora DSQL aplica limites quando você executa instruções `INSERT` ou `UPDATE`.

Tipos de dados de tempo de execução de consulta

Os tipos de dados de tempo de execução de consulta são tipos de dados internos usados no momento da execução da consulta. Esses tipos são distintos dos tipos compatíveis com PostgreSQL, como `varchar` e `integer`, que você define em seu esquema. Em vez disso, esses tipos são representações de tempo de execução que o Aurora DSQL usa ao processar uma consulta.

Os seguintes tipos de dados são aceitos somente durante o tempo de execução da consulta:

Tipo matriz

O Aurora DSQL aceita matrizes dos tipos de dados compatíveis. Por exemplo, você pode ter uma matriz de números inteiros. A função `string_to_array` divide uma string em uma matriz no

estilo PostgreSQL usando o delimitador de vírgula (`,`) tal como mostrado no exemplo a seguir. Você pode usar matrizes em expressões, saídas de funções ou cálculos temporários durante a execução da consulta.

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

A função retorna uma resposta semelhante à seguinte:

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

Tipo inet

O tipo de dados representa endereços de host IPv4 e IPv6 e as respectivas sub-redes. Esse tipo é útil ao analisar logs, filtrar em sub-redes IP ou fazer cálculos de rede em uma consulta. Para ter mais informações, consulte inet na documentação do PostgreSQL.

SQL compatível com o Aurora DSQL

O Aurora DSQL é compatível com uma ampla variedade de recursos principais do PostgreSQL. Nas seções a seguir, você pode obter informações sobre a compatibilidade geral com expressões do PostgreSQL. Essa lista não é exaustiva.

Comando **SELECT**

O Aurora DSQL é compatível com as cláusulas a seguir do comando `SELECT`.

Cláusula primária	Cláusulas compatíveis
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	

Cláusula primária	Cláusulas compatíveis
HAVING	
USING	
WITH (expressões de tabela comuns)	
INNER JOIN	ON
OUTER JOIN	LEFT, RIGHT, FULL, ON
CROSS JOIN	ON
UNION	ALL
INTERSECT	ALL
EXCEPT	ALL
OVER	RANK (), PARTITION BY
FOR UPDATE	

Idioma de definição de dados (DDL)

O Aurora DSQL é compatível com os comandos de DDL do PostgreSQL a seguir.

Command	Cláusula primária	Cláusulas compatíveis
CREATE	TABLE	Para ter informações sobre a sintaxe compatível do comando CREATE TABLE, consulte CREATE TABLE.
ALTER	TABLE	Para ter informações sobre a sintaxe compatível do comando ALTER TABLE, consulte ALTER TABLE.

Command	Cláusula primária	Cláusulas compatíveis
DROP	TABLE	
CREATE	[UNIQUE] INDEX ASYNC	Você pode fazer esse comando com os seguintes parâmetros: ON, NULLS FIRST e NULLS LAST. Para ter informações sobre a sintaxe compatível do comando CREATE INDEX ASYNC, consulte Índices assíncronos no Aurora DSQL.
DROP	INDEX	
CREATE	VIEW	Para ter mais informações sobre a sintaxe compatível do comando CREATE VIEW, consulte CREATE VIEW.
ALTER	VIEW	Para ter informações sobre a sintaxe compatível do comando ALTER VIEW, consulte ALTER VIEW.
DROP	VIEW	Para ter informações sobre a sintaxe compatível do comando DROP VIEW, consulte DROP VIEW.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

DML (Data Manipulation Language)

O Aurora DSQL é compatível com os comandos de DML do PostgreSQL a seguir.

Command	Cláusula primária	Cláusulas compatíveis
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Linguagem de controle de dados (DCL)

O Aurora DSQL é compatível com os comandos de DCL do PostgreSQL a seguir.

Command	Cláusulas compatíveis
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Linguagem de controle de transação (TCL)

O Aurora DSQL é compatível com os comandos de TCL do PostgreSQL a seguir.

Command	Cláusulas compatíveis
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

Comandos do utilitário

O Aurora DSQL é compatível com os comandos do utilitário do PostgreSQL a seguir.

- `EXPLAIN`
- `ANALYZE` (somente nome da relação)

Subconjuntos de comandos SQL compatíveis no Aurora DSQL

Essa seção do PostgreSQL fornece informações detalhadas sobre as expressões aceitas, concentrando-se em comandos com extensos conjuntos de parâmetros e subcomandos. Por exemplo, CREATE TABLE no PostgreSQL oferece várias cláusulas e parâmetros. Essa seção descreve todos os elementos da sintaxe do PostgreSQL que o Aurora DSQL aceita para esses comandos.

Tópicos

`CREATE TABLE`

`CREATE TABLE` define uma nova tabela.

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

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression )|
  DEFAULT default_expr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
```

```
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:

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

and like_option is:

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

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

ALTER TABLE

ALTER TABLE altera a definição de uma tabela de banco de dados.

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

where action is one of:

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

CREATE VIEW

CREATE VIEW define uma nova visualização persistente. O Aurora DSQL não aceita visualizações temporárias. Somente visualizações permanentes.

Sintaxe compatível

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

Descrição

`CREATE VIEW` define a visualização de uma consulta. A visualização não é materializada fisicamente. Em vez disso, a consulta é executada sempre que a visualização é consultada em uma consulta.

`CREATE or REPLACE VIEW` é semelhante, mas, se já houver uma visualização com o mesmo nome, ela será substituída. A nova consulta deve gerar as mesmas colunas que foram geradas pela consulta de visualização existente (ou seja, os mesmos nomes de coluna na mesma ordem e com os mesmos tipos de dados), mas existe a possibilidade de outras colunas serem adicionadas ao final da lista. Os cálculos que dão origem às colunas de saída podem ser diferentes.

Se um nome de esquema (como `CREATE VIEW myschema.myview ...`) for fornecido, a visualização será criada no esquema especificado. Do contrário, a visualização será criada no esquema atual.

O nome da visualização deve ser diferente do nome de qualquer outra relação (tabela, índice e visualização) no mesmo esquema.

Parâmetros

`CREATE VIEW` aceita vários parâmetros para controlar o comportamento de visualizações atualizáveis automaticamente.

`RECURSIVE`

Cria uma visualização recursiva. A sintaxe: `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` é equivalente a `CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;`.

É necessário especificar uma lista de nomes de coluna de visualização para uma visualização recursiva.

name

O nome da visualização a ser criada, que pode ser opcionalmente qualificada para o esquema. É necessário especificar uma lista de nomes para uma visualização recursiva.

column_name

Uma lista opcional de nomes a serem usados para as colunas da visualização. Se não for fornecido, o nome da coluna será deduzido da consulta.

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

Essa cláusula especifica parâmetros opcionais para uma visualização. Os parâmetros a seguir são aceitos.

- `check_option (enum)`: esse parâmetro pode ser `local` ou `cascaded`, e é equivalente a especificar `WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)`: deve ser usado se a visualização for destinada a oferecer segurança em nível de linha. No momento, o Aurora DSQL não oferece suporte à segurança em nível de linha, mas essa opção ainda farão com que as condições `WHERE` da visualização (e quaisquer condições usando operadores marcados como `LEAKPROOF`) sejam avaliadas primeiro.
- `security_invoker (boolean)`: esta opção faz com que as relações básicas subjacentes sejam cotejadas como os privilégios do usuário da visualização e não do proprietário da visualização. Consulte as observações abaixo para obter todos os detalhes.

Todas as opções acima podem ser alteradas nas visualizações existentes usando `ALTER VIEW`.

query

Um comando `SELECT` ou `VALUES` que fornecerá as colunas e linhas da visualização.

- `WITH [ CASCADED | LOCAL ] CHECK OPTION`: esta opção controla o comportamento das visualizações atualizáveis automaticamente. Quando ela for especificada, os comandos `INSERT` e `UPDATE` na visualização serão verificados para garantir que as novas linhas satisfaçam a condição de definição da visualização (ou seja, as novas linhas são verificadas para garantir que possam ser vistas na visualização). Do contrário, a atualização será rejeitada. Se `CHECK OPTION` não for especificada, os comandos `INSERT` e `UPDATE` na visualização poderão criar linhas que não são visíveis na visualização. As opções de verificação a seguir são aceitas.

- `LOCAL`: as novas linhas só são verificadas em relação às condições definidas diretamente na própria visualização. Quaisquer condições definidas nas visualizações básicas subjacentes não são verificadas (a menos que também especifiquem a `CHECK OPTION`).
- `CASCADED`: as novas linhas são verificadas em relação às condições da visualização e de todas as visualizações básicas subjacentes. Se `CHECK OPTION` for especificada, mas `LOCAL` e `CASCADED` não forem especificadas, a opção `CASCADED` será adotada.

> **Note**
>
> `CHECK OPTION` não pode ser usada com visualizações `RECURSIVE`. `CHECK OPTION` é aceita apenas em visualizações que são atualizáveis automaticamente.

Observações

Use a instrução `DROP VIEW` para descartar visualizações.

Os nomes e tipos de dados das colunas da visualização devem ser cuidadosamente considerados. Por exemplo, “CREATE VIEW vista AS SELECT ‘Hello World’;” não é recomendado porque o nome da coluna é padronizado como `?column?;`. Além disso, o tipo de dados padrão da coluna é `text`, que talvez não seja o que você queria.

Uma abordagem mais adequada é especificar explicitamente o nome da coluna e o tipo de dados, como `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

Por padrão, o acesso às relações básicas subjacentes referidas na visualização é determinado pelas permissões do proprietário da visualização. Em alguns casos, isso pode ser usado para fornecer acesso seguro, mas restrito, às tabelas subjacentes. Entretanto, nem todas as visualizações estão protegidas contra adulteração.

- Se a visualização tiver a propriedade `security_invoker` definida como verdadeira, o acesso às relações básicas subjacentes será determinado pelas permissões do usuário que está executando a consulta, e não pelo proprietário da visualização. Portanto, o usuário de uma visualização do tipo security_invoker deve ter as permissões relevantes na visualização e nas respectivas relações básicas subjacentes.
- Se alguma das relações básicas subjacentes for uma visualização do tipo security_invoker, ela será tratada como se tivesse sido acessada diretamente da consulta original. Desse modo, uma visualização do tipo security_invoker sempre verificará suas relações básicas subjacentes

usando as permissões do usuário atual, mesmo que seja acessada de uma visualização sem a propriedade `security_invoker`.

- As funções chamadas na visualização são tratadas da mesma forma que se tivessem sido chamadas diretamente da consulta usando a visualização. Portanto, o usuário de uma visualização deve ter permissões para chamar todas as funções usadas pela visualização. As funções na visualização são executadas com os privilégios do usuário que está executando a consulta ou do proprietário da função, dependendo se as funções estão definidas como `SECURITY INVOKER` ou `SECURITY DEFINER`. Por exemplo, chamar `CURRENT_USER` diretamente em uma visualização sempre retornará o usuário que está invocando, não o proprietário da visualização. Isso não é afetado pela configuração `security_invoker` da visualização e, portanto, uma visualização com `security_invoker` definida como falsa não é equivalente a uma função `SECURITY DEFINER`.
- O usuário que está criando ou substituindo uma visualização deve ter privilégios `USAGE` em todos os esquemas mencionados na consulta da visualização para pesquisar os objetos referidos nesses esquemas. Observe, no entanto, que essa pesquisa só ocorre quando a visualização é criada ou substituída. Portanto, o usuário da visualização precisa apenas do privilégio `USAGE` no esquema que contém a visualização, não nos esquemas mencionados na respectiva consulta, mesmo para uma visualização security_invoker.
- Quando `CREATE OR REPLACE VIEW` é usada em uma visualização existente, somente a regra de definição `SELECT` da visualização, mais qualquer parâmetro `WITH ( ... )` e a respectiva `CHECK OPTION`, é alterada. Outras características da visualização, como propriedade, permissões e regras não SELECT, permanecem inalteradas. Para substituir uma visualização, ela deve pertencer a você (isso inclui ser membro do perfil de proprietário).

Visualizações atualizáveis

As visualizações simples são atualizáveis automaticamente: o sistema permitirá que as instruções `INSERT`, `UPDATE` e `DELETE` sejam usadas na visualização da mesma forma que em uma tabela normal. Uma visualização é atualizável automaticamente se satisfizer todas as seguintes condições:

- A visualização deve ter exatamente uma entrada na respectiva lista `FROM`, que deve ser uma tabela ou outra visualização atualizável.
- A definição da visualização não deve conter cláusulas `WITH`, `DISTINCT`, `GROUP BY`, `HAVING`, `LIMIT` ou `OFFSET` no nível superior.
- Ela não deve conter operações de conjunto (`UNION`, `INTERSECT` ou `EXCEPT`) no nível superior.

- A lista de seleção da visualização não deve conter agregados, funções de janela ou funções de retorno de conjuntos.

Uma visualização atualizável automaticamente pode conter uma combinação de colunas atualizáveis e não atualizáveis. Uma coluna é atualizável se for uma referência simples a uma coluna atualizável da relação básica subjacente. Caso contrário, a coluna será somente leitura e ocorrerá um erro se uma instrução `INSERT` ou `UPDATE` tentar atribuir um valor a ela.

Para visualizações atualizáveis automaticamente, o sistema converte qualquer instrução `INSERT`, `UPDATE` ou `DELETE` presente na visualização na instrução correspondente presente na relação básica subjacente. As instruções `INSERT` com uma cláusula `ON CONFLICT UPDATE` são totalmente aceitas.

Se uma visualização atualizável automaticamente contiver uma condição `WHERE`, essa condição restringirá quais linhas da relação básica estão disponíveis para modificação por instruções `UPDATE` e `DELETE` na visualização. No entanto, o comando `UPDATE` pode alterar uma linha para que ela não satisfaça mais a condição `WHERE`, tornando-a invisível na visualização. Da mesma forma, existe a possibilidade de que o comando `INSERT` insira linhas da relação básica que não satisfazem a condição `WHERE`, tornando-as invisíveis na visualização. `ON CONFLICT UPDATE` pode afetar de maneira similar uma linha existente não visível na visualização.

Você pode usar `CHECK OPTION` para evitar que os comandos `INSERT` e `UPDATE` criem linhas que não fiquem visíveis na visualização.

Se uma visualização atualizável automaticamente for marcada com a propriedade security_barrier, todas as condições `WHERE` da visualização (e quaisquer condições usando operadores marcados como `LEAKPROOF`) serão sempre avaliadas antes de qualquer condição que um usuário da visualização tenha adicionado. Observe que, por causa disso, as linhas que, em última análise, não são exibidas (porque não atendem às condições `WHERE` do usuário) ainda assim podem acabar sendo bloqueadas. Você pode usar `EXPLAIN` para ver quais condições são aplicadas no nível da relação (e, portanto, não bloqueiam linhas) e quais não são.

Por padrão, uma visualização mais complexa que não satisfaça todas essas condições é somente leitura: o sistema não permite inserção, atualização ou exclusão na visualização.

> **Note**
>
> O usuário que executa a inserção, atualização ou exclusão na visualização deve ter o privilégio correspondente de inserção, atualização ou exclusão na visualização. Por

padrão, o proprietário da visualização deve ter os privilégios relevantes nas relações básicas subjacentes, enquanto o usuário que executa a atualização não precisa de nenhuma permissão nas relações básicas subjacentes. Entretanto, se a visualização tiver security_invoker definido como verdadeiro, o usuário que executa a atualização, em vez do proprietário da visualização, deverá ter os privilégios relevantes nas relações básicas subjacentes.

Exemplos

Para criar uma visualização composta de todos os filmes de comédia.

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

Isso criará uma visualização que contém as colunas que estão na tabela `film` no momento da criação da visualização. Embora se tenha usado * para criar a visualização, as colunas adicionadas posteriormente à tabela não farão parte da visualização.

Crie uma visualização com `LOCAL CHECK OPTION`.

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

Isso criará uma visualização que verifica `kind` e `classification` das linhas novas.

Crie uma as linhas com uma combinação de colunas atualizáveis e não atualizáveis.

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

```
    WHERE f.kind = 'Comedy';
```

Essa as linhas aceitará `INSERT`, `UPDATE` e `DELETE`. Todas as colunas da tabela de filmes serão atualizáveis, enquanto as colunas `country` e `avg_rating` computadas serão somente leitura.

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

> **Note**
>
> Embora o nome da visualização recursiva seja qualificado para o esquema `CREATE`, sua autorreferência interna não é qualificada para o esquema. Isso ocorre porque o nome da expressão de tabela comum (CTE) criada implicitamente não pode ser qualificado para o esquema.

Compatibilidade

`CREATE OR REPLACE VIEW` é uma extensão da linguagem PostgreSQL. A cláusula `WITH ( ... )` também é uma extensão, assim como as visualizações de barreiras de segurança e as visualizações security_invoker. O Aurora DSQL é compatível com essas extensões de linguagem.

ALTER VIEW

A instrução `ALTER VIEW` permite alterar várias propriedades de uma visualização existente, e o Aurora DSQL é compatível com a sintaxe completa do PostgreSQL referente a esse comando.

Sintaxe compatível

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

Descrição

`ALTER VIEW` altera várias propriedades auxiliares de uma visualização. (Se você quiser modificar a consulta de definição da visualização, use `CREATE OR REPLACE VIEW`.) É necessário ter a propriedade da visualização para usar `ALTER VIEW`. Para alterar o esquema de uma visualização, você também deve ter privilégios `CREATE` no novo esquema. Para alterar o proprietário, você deve ser capaz de aplicar `SET ROLE` ao novo perfil de propriedade e esse perfil deve ter privilégios `CREATE` no esquema da visualização. Essas restrições determinam que a alteração do proprietário não faz nada que você não possa fazer ao descartar e recriar a visualização.

Parâmetros

Parâmetros do `ALTER VIEW`

`name`

O nome (opcionalmente qualificado para o esquema) de uma visualização existente.

`column_name`

Novo nome para uma coluna existente.

`IF EXISTS`

Não gera um erro se a visualização não existir. Um aviso é emitido nesse caso.

`SET/DROP DEFAULT`

Esses formulários definem ou removem o valor padrão de uma coluna. O valor padrão de uma coluna da visualização é substituído em qualquer comando `UPDATE` ou `INSERT` no qual o destino seja a visualização. O padrão da visualização terá precedência sobre quaisquer valores padrão das relações subjacentes.

new_owner

O nome de usuário do novo proprietário da visualização.

new_name

O nome da nova visualização.

new_schema

O novo esquema para a visualização.

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

Define ou redefine uma opção de visualização. As opções a seguir são aceitas.

- `check_option (enum)`: altera a opção de verificação da visualização. O valor deve ser `local` ou `cascaded`.
- `security_barrier (boolean)`: altera a propriedade de barreira de segurança da visualização. O valor deve ser um booliano, como `true` ou `false`.
- `security_invoker (boolean)`: altera a propriedade de barreira de segurança da visualização. O valor deve ser um booliano, como `true` ou `false`.

Observações

Por motivos históricos do PostgreSQL, `ALTER TABLE` também pode ser usado com visualizações; mas as únicas variantes de `ALTER TABLE` permitidas com visualização são equivalentes às mostradas anteriormente.

Exemplos

Mude o nome da visualização `foo` para `bar`.

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

Anexar um valor de coluna padrão a uma visualização atualizável.

```
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
```

Compatibilidade

`ALTER VIEW` é uma extensão do PostgreSQL do padrão SQL que o Aurora DSQL aceita.

DROP VIEW

A instrução `DROP VIEW` remove uma visualização existente. O Aurora DSQL é compatível com a sintaxe completa do PostgreSQL referente a esse comando.

Sintaxe compatível

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

Descrição

`DROP VIEW` descarta uma visualização existente. Para executar esse comando, a visualização deve pertencer a você.

Parâmetros

`IF EXISTS`

Não gera um erro se a visualização não existir. Um aviso é emitido nesse caso.

`name`

O nome (opcionalmente qualificado para o esquema) da visualização a ser removida.

`CASCADE`

Descarta automaticamente os objetos que dependem da visualização (como outras visualizações) e, por sua vez, todos os objetos que dependem desses objetos.

`RESTRICT`

Recusa-se a remover a visualização se qualquer objeto depender dela. Esse é o padrão.

Exemplos

```
DROP VIEW kinds;
```

Compatibilidade

Esse comando segue o padrão SQL, com a exceção de que o padrão permite que apenas uma visualização seja eliminada por comando, além da opção `IF EXISTS`, que é uma extensão do PostgreSQL que o Aurora DSQL aceita.

Recursos do PostgreSQL não compatíveis no Aurora DSQL

O Aurora DSQL é compatível com o PostgreSQL. Isso significa que o Aurora DSQL é compatível com os principais recursos relacionais, como transações ACID, índices secundários, junções,

inserções e atualizações. Para obter uma visão geral dos recursos SQL compatíveis, consulte as expressões SQL compatíveis.

As seções a seguir destacam quais recursos do PostgreSQL não podem ser usados no momento no Aurora DSQL.

Objetos não compatíveis

Os objetos não aceitos pelo Aurora DSQL incluem o seguinte:

- Vários bancos de dados em um único cluster do Aurora DSQL
- Tabelas temporárias
- Acionadores
- Tipos (suporte parcial)
- Tablespaces
- Funções escritas em linguagens diferentes do SQL
- Sequências
- Partições

Restrições não compatíveis

- Chaves externas
- Restrições de exclusão

Comandos incompatíveis

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

> **Note**
>
> O Aurora DSQL não requer limpeza. O sistema mantém estatísticas e gerencia a otimização do armazenamento automaticamente sem comandos vacuum manuais.

Extensões não compatíveis

O Aurora DSQL não é compatível com extensões do PostgreSQL. A tabela a seguir mostra as extensões que não são aceitas:

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

Expressões de filtro incompatíveis

A tabela a seguir descreve as cláusulas que não são aceitas no Aurora DSQL.

Categoria	Cláusula primária	Cláusulas não compatível
`CREATE`	`INDEX ASYNC`	`ASC | DESC`
`CREATE`	`INDEX`[1]	
`TRUNCATE`		
`ALTER`	`SYSTEM`	Todos os comandos `ALTER SYSTEM` estão bloqueados.
`CREATE`	`TABLE`	`COLLATE`, `AS SELECT`, `INHERITS`, `PARTITION`
`CREATE`	`FUNCTION`	`LANGUAGE` *`non-sql-lang`* , onde *`non-sql-lang`* é qualquer linguagem diferente de `SQL`.
`CREATE`	`TEMPORARY`	`TABLES`

Categoria	Cláusula primária	Cláusulas não compatível
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	Não é possível criar bancos de dados adicionais.

[1] Consulte Índices assíncronos no Aurora DSQL para criar um índice em uma coluna de uma tabela especificada.

Considerações sobre o Aurora DSQL para compatibilidade com o PostgreSQL

Considere as limitações de compatibilidade a seguir ao usar o Aurora DSQL. Com relação a considerações gerais, consulte Considerações para trabalhar com Amazon Aurora DSQL. Com relação a cotas e limites, consulte Cotas de cluster e limites de banco de dados no Amazon Aurora DSQL.

- O Aurora DSQL usa um único banco de dados incorporado chamado `postgres`. Não é possível criar bancos de dados adicionais nem renomear ou excluir o banco de dados `postgres`.
- O banco de dados `postgres` usa a codificação de caracteres UTF-8. Não é possível alterar a criptografia.
- O banco de dados usa somente o agrupamento `C`.
- O Aurora DSQL usa `UTC` como fuso horário do sistema. Não é possível modificar o fuso horário usando parâmetros ou instruções SQL, como `SET TIMEZONE`.
- O nível de isolamento de transação é fixo em `Repeatable Read` no PostgreSQL.
- As transações têm as seguintes restrições:
  - Uma transação não pode misturar operações de DDL e de DML.

  - Uma transação pode incluir apenas uma instrução de DDL.
  - Uma transação pode modificar até 3 mil linhas, independentemente do número de índices secundários.
  - O limite de 3 mil linhas se aplica a todas as instruções de DML (`INSERT`, `UPDATE` e `DELETE`).
- As conexões de banco de dados atingem o tempo limite após uma hora.
- No momento, o Aurora DSQL não permite que você execute `GRANT [permission] ON DATABASE`. Se você tentar executar essa instrução, o Aurora DSQL exibirá a mensagem de erro `ERROR: unsupported object type in GRANT`.
- O Aurora DSQL não permite que perfis de usuário não administrativo executem o comando `CREATE SCHEMA`. Não é possível executar o comando `GRANT [permission] on DATABASE` e conceder permissões `CREATE` no banco de dados. Se um perfil de usuário não administrativo tentar criar um esquema, o Aurora DSQL exibirá a mensagem de erro `ERROR: permission denied for database postgres`.
- Os usuários que não são administradores não podem criar objetos no esquema público. Somente usuários administradores podem criar objetos no esquema público. O perfil de usuário administrador tem permissões para conceder acesso de leitura, gravação e modificação a esses objetos para usuários não administradores, mas não pode conceder permissões `CREATE` para o esquema público em si. Os usuários não administradores devem usar esquemas diferentes criados pelo usuário para criar objetos.
- O Aurora DSQL não aceita o comando `ALTER ROLE [] CONNECTION LIMIT`. Entre em contato com o AWS Support se precisar aumentar o limite de conexão.
- O Aurora DSQL não oferece suporte ao asyncpg, o driver assíncrono de banco de dados PostgreSQL para Python.

Controle de simultaneidade no Aurora DSQL

A simultaneidade permite que várias sessões acessem e modifiquem dados simultaneamente sem comprometer a integridade e a consistência dos dados. O Aurora DSQL oferece compatibilidade com o PostgreSQL e implementa um mecanismo moderno de controle de simultaneidade sem bloqueio. Ele mantém a conformidade total com ACID por meio do isolamento de snapshots, garantindo a consistência e a confiabilidade dos dados.

Uma das principais vantagens do Aurora DSQL é sua arquitetura sem bloqueios, que elimina gargalos comuns de desempenho do banco de dados. O Aurora DSQL impede que transações

lentas bloqueiem outras operações e elimina o risco de deadlocks. Essa abordagem torna o Aurora DSQL particularmente valioso para aplicações de alto throughput em que o desempenho e a escalabilidade são essenciais.

Conflitos de transação

O Aurora DSQL usa o controle de simultaneidade otimista (OCC), que funciona de forma diferente dos sistemas tradicionais baseados em bloqueios. Em vez de usar bloqueios, o OCC avalia os conflitos no horário da confirmação. Quando várias transações entram em conflito ao atualizar a mesma linha, o Aurora DSQL gerencia as transações da seguinte forma:

- A transação com o horário de confirmação mais antigo é processada pelo Aurora DSQL.
- As transações conflitantes recebem um erro de serialização do PostgreSQL, indicando a necessidade de serem repetidas.

Projete suas aplicações para implementar lógica de novas tentativas para lidar com conflitos. O padrão de design ideal é idempotente, permitindo a repetição da transação como primeiro recurso sempre que possível. A lógica recomendada é semelhante à lógica de cancelar e repetir em uma situação padrão de tempo limite de bloqueio ou deadlock do PostgreSQL. No entanto, o OCC exige que as aplicações exerçam essa lógica com maior frequência.

Diretrizes para otimizar o desempenho da transação

Para otimizar o desempenho, minimize a alta contenção em chaves únicas ou em pequenos intervalos de chaves. Para atingir esse objetivo, projete seu esquema para distribuir atualizações pelo intervalo de chaves do cluster usando as seguintes diretrizes:

- Escolha uma chave primária aleatória para suas tabelas.
- Evite padrões que aumentem a contenção em chaves únicas. Essa abordagem garante um desempenho ideal mesmo com o aumento do volume de transações.

DDL e transações distribuídas no Aurora DSQL

A linguagem de definição de dados (DDL) se comporta de forma diferente no Aurora DSQL em comparação ao PostgreSQL. O Aurora DSQL apresenta uma camada de banco de dados multi-AZ distribuída e sem compartilhamento, criada com base em frotas de computação e armazenamento multilocatário. Como não existe um único nó de banco de dados primário ou líder, o catálogo do

banco de dados é distribuído. Por isso, o Aurora DSQL gerencia as alterações do esquema de DDL como transações distribuídas.

Especificamente, a DDL se comporta de forma diferente no Aurora DSQL da seguinte forma:

Erros de controle de simultaneidade

O Aurora DSQL exibirá um erro de violação de controle de simultaneidade se você executar uma transação enquanto outra transação atualiza um recurso. Por exemplo, considere a seguinte sequência de ações:

1. Na sessão 1, um usuário adiciona uma coluna à tabela `mytable`.
2. Na sessão 2, um usuário tenta inserir uma linha em `mytable`.

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

DDL e DML na mesma transação

As transações no Aurora DSQL podem conter somente uma instrução de DDL e não podem ter instruções de DDL e DML. Essa restrição significa que não é possível criar uma tabela e inserir dados na mesma tabela dentro da mesma transação. Por exemplo, o Aurora DSQL é compatível aceita as transações sequenciais a seguir.

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

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

O Aurora DSQL não aceita a transação a seguir, que inclui instruções `CREATE` e `INSERT`.

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

DDL assíncrona

No PostgreSQL padrão, as operações de DDL, como `CREATE INDEX`, bloqueiam a tabela afetada, tornando-a indisponível para leituras e gravações de outras sessões. No Aurora DSQL, essas instruções de DDL são executadas de forma assíncrona usando um gerenciador em segundo plano. O acesso à tabela afetada não é bloqueado. Assim, a DDL em tabelas grandes pode ser executada sem tempo de inatividade ou impacto no desempenho. Para ter mais informações sobre o gerenciador de trabalhos assíncronos no Aurora DSQL, consulte Índices assíncronos no Aurora DSQL.

Chaves primárias no Aurora DSQL

No Aurora DSQL, uma chave primária é um recurso que organiza fisicamente os dados da tabela. É semelhante à operação `CLUSTER` no PostgreSQL ou a um índice em cluster em outros bancos de dados. Quando você define uma chave primária, o Aurora DSQL cria um índice que inclui todas as colunas na tabela. A estrutura de chave primária no Aurora DSQL garante acesso e gerenciamento eficientes dos dados.

Estrutura e armazenamento de dados

Quando você define uma chave primária, o Aurora DSQL armazena os dados da tabela na ordem da chave primária. Essa estrutura organizada por índice permite que uma pesquisa de chave primária recupere todos os valores das colunas diretamente, em vez de seguir um indicador para os dados, como em um índice tradicional de árvore B. Diferentemente da operação `CLUSTER` no PostgreSQL, que reorganiza os dados apenas uma vez, o Aurora DSQL mantém essa ordem de forma automática e contínua. Essa abordagem melhora o desempenho das consultas que dependem do acesso à chave primária.

O Aurora DSQL também usa a chave primária para gerar uma chave exclusiva em todo o cluster para cada linha em tabelas e índices. Essa chave única também apoia o gerenciamento distribuído de dados. Ele permite o particionamento automático de dados em vários nós, permitindo armazenamento escalável e alta simultaneidade. Por isso, a estrutura de chave primária ajuda o Aurora DSQL a escalar automaticamente e gerenciar workloads simultâneas com eficiência.

Diretrizes para escolher uma chave primária

Ao escolher e usar uma chave primária no Aurora DSQL, pense nestas diretrizes:

- Defina uma chave primária ao criar uma tabela. Não é possível alterar essa chave nem adicionar uma nova chave primária posteriormente. A chave primária se torna parte da chave de todo o cluster usada para particionamento de dados e ajuste de escala automático do throughput de gravação. Se você não especificar uma chave primária, o Aurora DSQL atribuirá um ID sintético oculto.
- Para tabelas com altos volumes de gravação, evite usar números inteiros uniformemente crescentes como chaves primárias. Isso pode gerar problemas de desempenho ao direcionar todas as novas inserções para uma única partição. Em vez disso, use chaves primárias com distribuição aleatória para garantir a distribuição uniforme das gravações nas partições de armazenamento.
- Para tabelas que mudam com pouca frequência ou são somente leitura, você pode usar uma chave ascendente. Exemplos de chave ascendente são carimbos de data/hora ou números de sequência. Uma chave densa tem vários valores próximos ou duplicados. Você pode usar uma chave ascendente mesmo que seja densa, pois o desempenho da gravação é menos importante.
- Se uma verificação completa da tabela não atender aos seus requisitos de desempenho, escolha um método de acesso mais eficiente. Na maioria dos casos, isso significa usar uma chave primária que corresponda à chave de junção e pesquisa mais comum nas consultas.
- O tamanho máximo combinado das colunas em uma chave primária é 1 kibibyte. Para ter mais informações, consulte Limites de banco de dados no Aurora DSQL e Tipos de dados compatíveis no Aurora DSQL.
- Você pode incluir até oito colunas em uma chave primária ou em um índice secundário. Para ter mais informações, consulte Limites de banco de dados no Aurora DSQL e Tipos de dados compatíveis no Aurora DSQL.

Índices assíncronos no Aurora DSQL

O comando `CREATE INDEX ASYNC` cria um índice em uma ou mais colunas de uma tabela especificada. Esse comando é uma operação de DDL assíncrona que não bloqueia outras transações. Ao executar `CREATE INDEX ASYNC`, o Aurora DSQL exibe imediatamente um `job_id`.

Você pode monitorar o status dessa tarefa assíncrona usando a visualização de sistema `sys.jobs`. Enquanto o trabalho de criação de índice está em andamento, você pode usar estes procedimentos e comandos:

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

Bloqueia a sessão atual até que o trabalho especificado seja concluído ou falhe. Retorna um booliano indicando êxito ou falha.

`DROP INDEX`

Cancela uma tarefa de criação de índice em andamento.

Quando a criação assíncrona do índice é concluída, o Aurora DSQL atualiza o catálogo do sistema para marcar o índice como ativo.

> **Note**
>
> Observe que as transações simultâneas que acessam objetos no mesmo namespace durante essa atualização podem encontrar erros de simultaneidade.

Quando o Aurora DSQL conclui uma tarefa de indexação assíncrona, ele atualiza o catálogo do sistema para mostrar que o índice está ativo. Se outras transações fizerem referência aos objetos no mesmo namespace nesse momento, você poderá ver um erro de simultaneidade.

Sintaxe

`CREATE INDEX ASYNC` usa a sintaxe a seguir.

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

Parâmetros

`UNIQUE`

Instrui o Aurora DSQL a verificar se há valores duplicados na tabela ao criar o índice e sempre que você adicionar dados. Se você especificar esse parâmetro, as operações de inserção e atualização que resultariam em entradas duplicadas gerarão um erro.

`IF NOT EXISTS`

Instrui o Aurora DSQL a não lançar uma exceção se já houver um índice com o mesmo nome. Nessa situação, o Aurora DSQL não cria o índice. Observe que o índice que você está tentando criar pode ter uma estrutura muito diferente da estrutura do índice existente. Se você especificar esse parâmetro, o nome do índice será obrigatório.

`name`

O nome do índice. Não é possível incluir o nome do esquema nesse parâmetro.

O Aurora DSQL cria o índice no mesmo esquema da tabela principal. O nome do índice deve ser diferente do nome de qualquer outro objeto, como tabela ou índice, no esquema.

Se você não especificar um nome, o Aurora DSQL gerará um nome automaticamente com base no nome da tabela principal e da coluna indexada. Por exemplo, se você executar `CREATE INDEX ASYNC on table1 (col1, col2)`, o Aurora DSQL atribuirá automaticamente o nome `table1_col1_col2_idx` ao índice.

`NULLS FIRST | LAST`

A ordem de classificação das colunas nulas e não nulas. `FIRST` indica que o Aurora DSQL deve classificar colunas nulas antes de colunas não nulas. `LAST` indica que o Aurora DSQL deve classificar colunas nulas após colunas não nulas.

`INCLUDE`

Uma lista de colunas a serem incluídas no índice como colunas não chave. Não é possível usar uma coluna não chave em uma qualificação de pesquisa de verificação de índice. O Aurora DSQL ignora a coluna em termos de exclusividade para um índice.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Especifica se o Aurora DSQL deve considerar valores nulos como distintos em um índice exclusivo. O padrão é `DISTINCT`, o que significa que um índice exclusivo pode conter vários valores nulos em uma coluna. `NOT DISTINCT` indica que um índice não pode conter vários valores nulos em uma coluna.

Observações de uso

Considere as seguintes diretrizes:

- O comando `CREATE INDEX ASYNC` não introduz bloqueios. Isso também não afeta a tabela base que o Aurora DSQL usa para criar o índice.
- Durante as operações de migração do esquema, o procedimento `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* é útil. Ele garante que as operações subsequentes de DDL e DML tenham como alvo o índice recém-criado.
- Sempre que o Aurora DSQL executa uma nova tarefa assíncrona, ele verifica a visualização `sys.jobs` e exclui tarefas com status `completed` ou `failed` por mais de 30 minutos. Portanto, `sys.jobs` mostra principalmente as tarefas em andamento e não contém informações sobre tarefas antigas.
- Se o Aurora DSQL não conseguir criar um índice assíncrono, o índice permanecerá `INVALID`. Para índices exclusivos, as operações de DML estão sujeitas a restrições de exclusividade até que você elimine o índice. Recomendamos que você elimine os índices inválidos e os recrie.

Criar um índice: exemplo

O exemplo a seguir demonstra como criar um esquema, uma tabela e, em seguida, um índice.

1. Crie uma tabela chamada `test.departments`.

```
CREATE SCHEMA test;

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

2. Insira uma linha de dados na tabela.

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

3. Crie um índice assíncrono.

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

   O comando `CREATE INDEX` exibe um ID de trabalho, conforme mostrado abaixo.

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

O `job_id` indica que o Aurora DSQL enviou um novo trabalho para criar o índice. Você pode usar o procedimento `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* para bloquear outros trabalhos na sessão até que o trabalho seja concluído ou atinja o tempo limite.

Consultar o status da criação do índice: exemplo

Consulte a visualização `sys.jobs` do sistema para verificar o status de criação do índice, conforme mostrado no exemplo a seguir.

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

O Aurora DSQL exibe uma resposta semelhante à seguinte:

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

A coluna de status pode ser um dos seguintes valores:

submitted	**processing**	**failed**	**completed**
A tarefa foi enviada, mas o Aurora DSQL ainda não começou a processá-la.	O Aurora DSQL está processando a tarefa.	A tarefa falhou. Consulte os detalhes da coluna para ter mais informações. Se o Aurora DSQL falhar ao criar o índice, ele não removerá automaticamente a definição do índice. Você deve remover o índice manualmente com o comando `DROP INDEX`.	Aurora DSQL

Você também pode consultar o estado do índice por meio das tabelas `pg_index` e `pg_class` do catálogo. Os atributos `indisvalid` e `indisimmediate`, especificamente, podem indicar em que estado o índice está. Embora o Aurora DSQL crie o índice, o status inicial dele é `INVALID`. O sinalizador `indisvalid` do índice exibe `FALSE` ou `f`, o que indica que o índice não é válido. Se o sinalizador exibir `TRUE` ou `t`, isso significa que o índice está pronto.

```
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)
```

Falhas na criação de índices únicos

Se sua tarefa assíncrona de criação de índice único mostrar um estado de falha com o detalhe `Found duplicate key while validating index for UCVs`, isso indica que não foi possível criar um índice único devido a violações à restrição de unicidade.

Como resolver falhas na criação de índices únicos

1. Remova todas as linhas da tabela primária que tenham entradas duplicadas para as chaves especificadas em seu índice secundário único.
2. Elimine o índice com falha.
3. Emita um novo comando create index.

Detectar violações de unicidade em tabelas primárias

A consulta SQL a seguir ajuda você a identificar valores duplicados em uma coluna especificada da tabela. Isso é particularmente útil quando você precisa impor unicidade em uma coluna que no

momento não está definida como chave primária ou não tem uma restrição única, como endereços de e-mail em uma tabela de usuários.

Os exemplos abaixo demonstram como criar uma tabela de usuários de exemplo, preenchê-la com dados de teste contendo duplicatas conhecidas e, em seguida, executar a consulta de detecção.

Definir esquema de tabela

```
-- 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
);
```

Inserir dados de exemplo que incluam conjuntos de endereços de e-mail duplicados

```
-- 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');
```

Executar consulta de detecção de duplicatas

```
-- 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;
```

Visualizar todos os registros com endereços de e-mail duplicados

```
 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)
```

Se tentássemos a instrução de criação do índice agora, ela falharia:

```
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)
```

Tabelas e comandos de sistema no Aurora DSQL

Consulte as seções a seguir para saber mais sobre tabelas e catálogos de sistema compatíveis no Aurora DSQL.

Tabelas de sistema

O Aurora DSQL é compatível com o PostgreSQL; portanto, muitas tabelas de catálogo do sistema e visualizações do PostgreSQL também existem no Aurora DSQL.

Tabelas de catálogo e visualizações importantes do PostgreSQL

A tabela a seguir descreve as tabelas e visualizações mais comuns que você pode usar no Aurora DSQL.

Nome	Descrição
`pg_namespace`	Informações sobre todos os esquemas
`pg_tables`	Informações sobre todas as tabelas
`pg_attribute`	Informações sobre todos os atributos
`pg_views`	Informações sobre visualizações (pre)definidas
`pg_class`	Descreve todas as tabelas, colunas, índices e objetos semelhantes
`pg_stats`	Uma visualização das estatísticas do planejador
`pg_user`	Informações sobre usuários
`pg_roles`	Informações sobre usuários e grupos
`pg_indexes`	Lista todos os índices
`pg_constraint`	Lista as restrições nas tabelas

Tabelas de catálogo aceitas e não aceitas

A tabela a seguir indica quais tabelas são aceitas e não aceitas no Aurora DSQL.

Name	Aplicável ao Aurora DSQL
`pg_aggregate`	Não
`pg_am`	Sim
`pg_amop`	Não
`pg_amproc`	Não
`pg_attrdef`	Sim
`pg_attribute`	Sim
`pg_authid`	Não (use `pg_roles`)
`pg_auth_members`	Sim
`pg_cast`	Sim
`pg_class`	Sim
`pg_collation`	Sim
`pg_constraint`	Sim
`pg_conversion`	Não
`pg_database`	Não
`pg_db_role_setting`	Sim
`pg_default_acl`	Sim
`pg_depend`	Sim
`pg_description`	Sim

Name	Aplicável ao Aurora DSQL
pg_enum	Não
pg_event_trigger	Não
pg_extension	Não
pg_foreign_data_wrapper	Não
pg_foreign_server	Não
pg_foreign_table	Não
pg_index	Sim
pg_inherits	Sim
pg_init_privs	Não
pg_language	Não
pg_largeobject	Não
pg_largeobject_metadata	Sim
pg_namespace	Sim
pg_opclass	Não
pg_operator	Sim
pg_opfamily	Não
pg_parameter_acl	Sim
pg_partitioned_table	Não
pg_policy	Não
pg_proc	Não

Name	Aplicável ao Aurora DSQL
pg_publication	Não
pg_publication_namespace	Não
pg_publication_rel	Não
pg_range	Sim
pg_replication_origin	Não
pg_rewrite	Não
pg_seclabel	Não
pg_sequence	Não
pg_shdepend	Sim
pg_shdescription	Sim
pg_shseclabel	Não
pg_statistic	Sim
pg_statistic_ext	Não
pg_statistic_ext_data	Não
pg_subscription	Não
pg_subscription_rel	Não
pg_tablespace	Não
pg_transform	Não
pg_trigger	Não
pg_ts_config	Sim

Name	Aplicável ao Aurora DSQL
pg_ts_config_map	Sim
pg_ts_dict	Sim
pg_ts_parser	Sim
pg_ts_template	Sim
pg_type	Sim
pg_user_mapping	Não

Visualizações do sistema aceitas e não aceitas

A tabela a seguir indica quais visualizações são aceitas e não aceitas no Aurora DSQL.

Name	Aplicável ao Aurora DSQL
pg_available_extensions	Não
pg_available_extension_versions	Não
pg_backend_memory_contexts	Sim
pg_config	Não
pg_cursors	Não
pg_file_settings	Não
pg_group	Sim
pg_hba_file_rules	Não
pg_ident_file_mappings	Não
pg_indexes	Sim

Name	Aplicável ao Aurora DSQL
pg_locks	Não
pg_matviews	Não
pg_policies	Não
pg_prepared_statements	Não
pg_prepared_xacts	Não
pg_publication_tables	Não
pg_replication_origin_status	Não
pg_replication_slots	Não
pg_roles	Sim
pg_rules	Não
pg_seclabels	Não
pg_sequences	Não
pg_settings	Sim
pg_shadow	Sim
pg_shmem_allocations	Sim
pg_stats	Sim
pg_stats_ext	Não
pg_stats_ext_exprs	Não
pg_tables	Sim
pg_timezone_abbrevs	Sim

Name	Aplicável ao Aurora DSQL
pg_timezone_names	Sim
pg_user	Sim
pg_user_mappings	Não
pg_views	Sim
pg_stat_activity	Não
pg_stat_replication	Não
pg_stat_replication_slots	Não
pg_stat_wal_receiver	Não
pg_stat_recovery_prefetch	Não
pg_stat_subscription	Não
pg_stat_subscription_stats	Não
pg_stat_ssl	Sim
pg_stat_gssapi	Não
pg_stat_archiver	Não
pg_stat_io	Não
pg_stat_bgwriter	Não
pg_stat_wal	Não
pg_stat_database	Não
pg_stat_database_conflicts	Não
pg_stat_all_tables	Não

Name	Aplicável ao Aurora DSQL
pg_stat_all_indexes	Não
pg_statio_all_tables	Não
pg_statio_all_indexes	Não
pg_statio_all_sequences	Não
pg_stat_slru	Não
pg_statio_user_tables	Não
pg_statio_user_sequences	Não
pg_stat_user_functions	Não
pg_stat_user_indexes	Não
pg_stat_progress_analyze	Não
pg_stat_progress_basebackup	Não
pg_stat_progress_cluster	Não
pg_stat_progress_create_index	Não
pg_stat_progress_vacuum	Não
pg_stat_sys_indexes	Não
pg_stat_sys_tables	Não
pg_stat_xact_all_tables	Não
pg_stat_xact_sys_tables	Não
pg_stat_xact_user_functions	Não
pg_stat_xact_user_tables	Não

Name	Aplicável ao Aurora DSQL
pg_statio_sys_indexes	Não
pg_statio_sys_sequences	Não
pg_statio_sys_tables	Não
pg_statio_user_indexes	Não

As visualizações sys.jobs e sys.iam_pg_role_mappings

O Aurora DSQL não aceita as seguintes visualizações do sistema:

sys.jobs

`sys.jobs` fornece informações sobre o status dos trabalhos assíncronos. Por exemplo, após a criação de um índice assíncrono, o Aurora DSQL exibe um `job_uuid`. Você pode usar esse `job_uuid` com `sys.jobs` para pesquisar o status do trabalho.

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

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

sys.iam_pg_role_mappings

A visualização `sys.iam_pg_role_mappings` fornece informações sobre as permissões concedidas aos usuários do IAM. Por exemplo, se `DQSLDBConnect` for um perfil do IAM que dá ao Aurora DSQL acesso a não administradores e um usuário chamado `testuser` receber o perfil `DQSLDBConnect` e as permissões correspondentes, você poderá consultar a visualização `sys.iam_pg_role_mappings` para ver quais usuários recebem quais permissões.

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

A tabela pg_class

A tabela `pg_class` armazena metadados sobre objetos do banco de dados. Para obter a contagem aproximada de linhas que estão em uma tabela, execute o comando a seguir.

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

Esse comando retorna uma saída semelhante à seguinte.

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

O comando **ANALYZE**

O comando `ANALYZE` coleta estatísticas sobre o conteúdo das tabelas no banco de dados e armazena os resultados na visualização de sistema `pg_stats`. Posteriormente, o planejador de consultas usa essas estatísticas para ajudar a determinar os planos de execução mais eficientes para as consultas.

No Aurora DSQL, não é possível executar o comando `ANALYZE` em uma transação explícita. `ANALYZE` não está sujeito ao limite de tempo da transação do banco de dados.

Para reduzir a necessidade de intervenção manual e manter as estatísticas sempre atualizadas, o Aurora DSQL executa automaticamente `ANALYZE` como um processo em segundo plano. Esse trabalho em segundo plano é acionado automaticamente com base na taxa de alteração observada na tabela. Ele está vinculado ao número de linhas (tuplas) que foram inseridas, atualizadas ou excluídas desde a última análise.

`ANALYZE` é executado de forma assíncrona em segundo plano e sua atividade pode ser monitorada na visualização de sistema sys.jobs com a seguinte consulta:

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

Considerações importantes

> **Note**
>
> Os trabalhos `ANALYZE` são cobrados como outros trabalhos assíncronos no Aurora DSQL. Quando você modifica uma tabela, isso pode acionar indiretamente um trabalho automático

de coleta de estatísticas em segundo plano, o que pode resultar em cobranças de medição devido à atividade associada no nível do sistema.

Os trabalhos ANALYZE em segundo plano, acionados automaticamente, coletam os mesmos tipos de estatística de um ANALYZE manual e os aplicam por padrão às tabelas do usuário. As tabelas de sistema e de catálogo são excluídas desse processo automatizado.

Gerenciar clusters do Aurora DSQL

O Aurora DSQL fornece várias opções de configuração para ajudar você a estabelecer a infraestrutura de banco de dados certa para suas necessidades. Para configurar sua infraestrutura de cluster do Aurora DSQL, examine as seções a seguir.

Tópicos

Os recursos e as funcionalidades discutidos neste guia garantem que o ambiente do Aurora DSQL seja mais resiliente, responsivo e capaz de atender às suas aplicações à medida que elas crescem e evoluem.

Configurar clusters de região única

Criar um cluster

Crie um cluster usando a o comando create-cluster.

> **Note**
>
> A criação de clusters é uma operação assíncrona. Chame a API GetCluster até que o status mude para `ACTIVE`. Você pode se conectar ao cluster depois que ele ficar ativo.

Example Command

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

> **Note**
>
> Para desabilitar a proteção contra exclusão durante a criação, inclua o sinalizador `--no-deletion-protection-enabled`.

Example Resposta

```
{
    "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"
    }
}
```

Descrever um cluster

Obtenha informações sobre um cluster usando o comando get-cluster.

Example Command

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

Example Resposta

```
{
    "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"
    }
}
```

Atualizar um cluster

Atualize um cluster existente usando o comando update-cluster.

> **Note**
>
> As atualizações são operações assíncronas. Chame a API GetCluster até que o status mude para `ACTIVE` a fim de ver suas alterações.

Example Command

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

Example Resposta

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

Excluir um cluster

Exclua um cluster existente usando o comando delete-cluster.

> **Note**
>
> É possível excluir somente clusters que tenham a proteção contra exclusão desabilitada. Por padrão, a proteção contra exclusão está habilitada ao criar clusters.

Example Command

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

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

Example Resposta

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

Como listar clusters

Liste seus clusters usando o comando list-clusters.

Example Command

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

Example Resposta

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

Configurar clusters multirregionais

Este capítulo explica como configurar e gerenciar clusters em várias Regiões da AWS.

Conectar-se ao seu cluster multirregional

Os clusters emparelhados multirregionais fornecem dois endpoints regionais, um em cada Região da AWS do cluster emparelhado. Ambos os endpoints apresentam um único banco de dados lógico que comporta operações simultâneas de leitura e gravação com alta consistência de dados. Além dos clusters emparelhados, um cluster multirregional também tem uma Região testemunha que armazena uma janela limitada de registros de transações criptografados, que é usada para melhorar a durabilidade e a disponibilidade multirregional. As regiões testemunha multirregionais não têm endpoints.

Criar clusters multirregionais

Para criar clusters multirregionais, primeiro crie um cluster com uma região testemunha. Em seguida, você emparelha esse cluster com um segundo cluster que compartilha a mesma Região testemunha do seu primeiro cluster. O exemplo a seguir mostra como criar clusters no Leste dos EUA (Norte da Virgínia) e Leste dos EUA (Ohio) com o Oeste dos EUA (Oregon) como a região testemunha.

Etapa 1: criar o primeiro cluster no Leste dos EUA (Norte da Virgínia)

Para criar um cluster na Região da AWS Leste dos EUA (Norte da Virgínia) com propriedades multirregionais, use o comando abaixo.

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

Example Resposta:

```
{
    "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**
>
> Quando a operação de API é bem-sucedida, o cluster entra no estado `PENDING_SETUP`. A criação do cluster permanece `PENDING_SETUP` até que você atualize o cluster com o ARN de seu cluster emparelhado.

Etapa 2: criar o segundo cluster no Leste dos EUA (Ohio)

Para criar um cluster na Região da AWS Leste dos EUA (Ohio) com propriedades multirregionais, use o comando abaixo.

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

Example Resposta:

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

Quando a operação de API é bem-sucedida, o cluster passa para o estado `PENDING_SETUP`. A criação do cluster permanece `PENDING_SETUP` até que você o atualize com o ARN de outro cluster para emparelhamento.

Etapa 3: emparelhar o cluster no Leste dos EUA (Norte da Virgínia) com o do Leste dos EUA (Ohio)

Para emparelhar o cluster do Leste dos EUA (Norte da Virgínia) com o cluster do Leste dos EUA (Ohio), use o comando `update-cluster`. Especifique o nome do cluster do Leste dos EUA (Norte da Virgínia) e uma string JSON com o ARN do cluster do Leste dos EUA (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 Resposta

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

Etapa 4: emparelhar cluster no Leste dos EUA (Ohio) com o do Leste dos EUA (Norte da Virgínia)

Para emparelhar o cluster do Leste dos EUA (Ohio) com o cluster do Leste dos EUA (Norte da Virgínia), use o comando `update-cluster`. Especifique o nome do cluster do Leste dos EUA (Ohio) e uma string JSON com o ARN do cluster do Leste dos EUA (Norte da Virgínia).

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 Resposta

```
{
```

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

> **Note**
>
> Após o emparelhamento bem-sucedido, os dois clusters passam do status “PENDING_SETUP” para “CREATING” e, finalmente, para o status “ACTIVE” quando estiverem prontos para uso.

Visualizar propriedades de clusters multirregionais

Ao descrever um cluster, você pode visualizar as propriedades multirregionais de clusters em diferentes Regiões da AWS.

Example

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

Example Resposta

```
{
    "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"
       ]
```

```
    }
}
```

Clusters pares durante a criação

Você pode reduzir o número de etapas incluindo informações de emparelhamento durante a criação do cluster. Depois de criar o primeiro cluster no Leste dos EUA (Norte da Virgínia) (Etapa 1), você pode criar o segundo no Leste dos EUA (Ohio) e, ao mesmo tempo, iniciar o processo de emparelhamento incluindo o ARN do primeiro.

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"]}'
```

Isso associa as Etapas 2 e 4, mas você ainda precisa concluir a Etapa 3 (atualizar o primeiro cluster com o ARN do segundo) para estabelecer a relação de emparelhamento. Depois que todas as etapas forem concluídas, os dois clusters passarão pelos mesmos estados do processo padrão (de PENDING_SETUP para CREATING e, finalmente, para ACTIVE) quando estiverem prontos para uso.

Excluir clusters multirregionais

Para excluir um cluster multirregional, você precisa concluir duas etapas.

1. Desative a proteção contra exclusão para cada cluster.
2. Exclua cada cluster emparelhado separadamente nas respectivas Região da AWS.

Atualizar e excluir clusters no Leste dos EUA (Norte da Virgínia)

1. Desative a proteção contra exclusão usando o comando `update-cluster`.

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

2. Exclua o cluster usando o comando `delete-cluster`.

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

O comando retorna a seguinte resposta.

```
{
    "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**
>
> O cluster faz a transição para o status `PENDING_DELETE`. A exclusão não será concluída enquanto você não excluir o cluster emparelhado no Leste dos EUA (Ohio).

Atualizar e excluir clusters no Leste dos EUA (Ohio)

1. Desative a proteção contra exclusão usando o comando `update-cluster`.

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

2. Exclua o cluster usando o comando `delete-cluster`.

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

O comando retorna a seguinte resposta:

```
{
    "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**
>
> O cluster faz a transição para o status `PENDING_DELETE`. Após alguns segundos, o sistema faz automaticamente a transição de ambos os clusters emparelhados para o status `DELETING` depois da validação.

Configurar clusters multirregionais usando o AWS CloudFormation

Você pode usar o mesmo recurso `AWS::DSQL::Cluster` do AWS CloudFormation para implantar e gerenciar clusters do Aurora DSQL de região única e multirregião.

Consulte a referência do tipo de recurso do Amazon Aurora DSQL para saber mais sobre como criar, modificar e gerenciar clusters usando o recurso `AWS::DSQL::Cluster`.

Criação da configuração inicial do cluster

Primeiro crie um modelo AWS CloudFormation para definir seu cluster multirregional:

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

Crie pilhas em ambas as regiões usando os seguintes comandos da CLI da 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
```

Localizar identificadores de cluster

Recupere os IDs de recursos físicos para seus clusters:

```
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"
]
```

Atualizar a configuração do cluster

Atualize seu modelo do AWS CloudFormation para incluir ambos os ARNs do cluster:

```
---
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 a configuração atualizada em ambas as regiões:

```
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
```

Programar com o Aurora DSQL

O Aurora DSQL oferece as ferramentas a seguir para gerenciar recursos do Amazon DSQL de forma programática.

AWS Command Line Interface (AWS CLI)

Você pode criar e gerenciar seus recursos usando a AWS CLI em um shell da linha de comandos. A AWS CLI oferece acesso direto às APIs para Serviços da AWS, como o Aurora DSQL. Consulte a sintaxe e exemplos de comando do Aurora DSQL em dsql na Referência de comandos da AWS CLI.

Kits de desenvolvimento de software (SDKs) da AWS

A AWS fornece SDKs para muitas tecnologias e linguagens de programação conhecidas. Eles facilitam a chamada de Serviços da AWS de dentro de suas aplicações nessa linguagem ou tecnologia. Consulte mais informações sobre esses SDKs em Ferramentas para desenvolver e gerenciar aplicações na AWS.

API do Aurora DSQL

Essa API é outra interface de programação para o Aurora DSQL. Ao usar essa API, você deve formatar cada solicitação HTTPS corretamente e adicionar uma assinatura digital válida a cada solicitação. Para obter mais informações, consulte Referência de API.

AWS CloudFormation

O AWS::DSQL::Cluster é um recurso do AWS CloudFormation que permite criar e gerenciar clusters do Aurora DSQL como parte de sua infraestrutura como código. O AWS CloudFormation ajuda você a definir todo o seu ambiente da AWS em código, facilitando o provisionamento, a atualização e a replicação de sua infraestrutura de forma consistente e confiável.

Ao usar o recurso AWS::DSQL::Cluster em modelos do AWS CloudFormation, você pode provisionar declarativamente clusters do Aurora DSQL com seus demais recursos de nuvem. Isso ajuda a garantir que sua infraestrutura de dados seja implantada e gerenciada com o restante da pilha de aplicações.

SDKs código de exemplo da AWS para o Amazon Aurora DSQL

Os kits de desenvolvimento de software (SDKs) da AWS estão disponíveis para muitas linguagens de programação populares. Cada SDK fornece uma API, exemplos de código e documentação que permitem que os desenvolvedores criem facilmente aplicações em seu idioma de preferência.

[illegible] objetos relacionais (ORMs)
[illegible] aurora-dsql-samples/cpp
[illegible] aurora-dsql-samples/go
[illegible] es/

Linguagem	Exemplos de código do SDK / Exemplos de código de mapeamento objeto-relacionais (ORMs)
	aurora-dsql-samples/java
JavaScript	aws-samples/aurora-dsql-samples/javascript
.NET	aws-samples/aurora-dsql-samples/dotnet

Linguagem	Exemplos de código de SDK	Driver	Mapeamento de objetos relacionais (ORMs)
Python	AWS SDK para Python (Boto3) aws-samples/aurora-dsql-samples/python	psycopg psycopg2	SQLAlchemy Django
Ruby	AWS SDK para Ruby samples/aurora-dsql-samples/ruby	pg	Rails

	Exemplos de código do SDK GitHub	Mapeamento de objetos relacionais (ORMs)
Rust	aurora-dsql-samples/rust	
TypeScript	aurora-dsql-samples/typescript	Sequelize type-orm

Aurora DSQL com a AWS CLI

Consulte as seções a seguir para aprender a gerenciar clusters com a AWS CLI.

CreateCluster

Para criar um cluster, use o comando `create-cluster`.

> **Note**
>
> A criação de clusters ocorre de forma assíncrona. Chame a API `GetCluster` até que o status seja `ACTIVE`. Você pode se conectar a um cluster assim que ele ficar `ACTIVE`.

Exemplo de comando

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

> **Note**
>
> Se você quiser desabilitar a proteção contra exclusão na criação, inclua o sinalizador `--no-deletion-protection-enabled`.

Exemplo de resposta

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

GetCluster

Para descrever um cluster, use o comando `get-cluster`.

Exemplo de comando

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

Exemplo de resposta

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

UpdateCluster

Para atualizar um cluster existente, use o comando `update-cluster`.

> **Note**
>
> As atualizações ocorrem de forma assíncrona. Chame a API `GetCluster` até que o status seja `ACTIVE`, e você observará as alterações.

Exemplo de comando

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

Exemplo de resposta

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

DeleteCluster

Para excluir um cluster existente, use o comando `delete-cluster`.

> **Note**
>
> Você pode excluir somente um cluster que tenha a proteção contra exclusão desabilitada. A proteção contra exclusão está habilitada por padrão ao criar clusters.

Exemplo de comando

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

Exemplo de resposta

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

ListClusters

Para obter uma lista de clusters, use o comando `list-clusters`.

Exemplo de comando

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

Exemplo de resposta

```
{
    "clusters": [
        {
```

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

GetCluster em clusters mutirregionais

Para ter informações sobre um cluster multirregional, use o comando `get-cluster`. Para clusters multirregionais, a resposta incluirá o ARN dos clusters vinculados.

Exemplo de comando

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

Exemplo de resposta

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

```
}
```

Criar, ler, atualizar e excluir clusters do Aurora DSQL

Exemplos de criar, ler, atualizar e excluir (CRUD) são fornecidos para implantações de região única e multirregionais. Há uma seção `cluster_management` dedicada para cada linguagem de programação que demonstra essas tarefas principais de gerenciamento.

As implantações de região única são ideais para aplicações que atendam usuários em uma área geográfica específica, oferecendo gerenciamento simplificado e menor latência. As implantações multirregionais ajudam você a obter maior disponibilidade e recursos de recuperação de desastres distribuindo o banco de dados em várias Regiões da AWS.

Escolha o tipo de implantação que se alinha aos requisitos de disponibilidade, desempenho e distribuição geográfica da aplicação.

Tópicos

Criar um cluster

Consulte as informações a seguir para saber como criar clusters de região única e multirregionais no Aurora DSQL.

Python

Para criar um cluster em uma única Região da AWS, use o exemplo a seguir.

```
import boto3


def create_cluster(region):
    try:
        client = boto3.client("dsql", region_name=region)
```

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

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

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


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


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

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

```
import boto3


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

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

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

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

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

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

C++

O exemplo a seguir permite criar um cluster em uma única Região da AWS.

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

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

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

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

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

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

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

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

    return cluster;
}

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

            auto cluster = CreateCluster(region);

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

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

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

using namespace Aws;
```

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

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

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

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

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

    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

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

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

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

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

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

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

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

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

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

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

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

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

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

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

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

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

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

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

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

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

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

JavaScript

Para criar um cluster em uma única Região da AWS, use o exemplo a seguir.

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

async function createCluster(region) {

    const client = new DSQLClient({ region });

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

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

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

    await createCluster(region);
}

main();
```

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

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

async function createMultiRegionCluster(region1, region2, witnessRegion) {

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

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

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

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

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

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

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

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

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

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

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

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

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Use o exemplo a seguir para criar um cluster em uma única Região da AWS.

```
package org.example;

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

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

public class CreateCluster {

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

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

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

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

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

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

```
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

Use o exemplo a seguir para criar um cluster em uma única Região da AWS.

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

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

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

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

    Client::from_conf(config)
}

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

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

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

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

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

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

```
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

Use o exemplo a seguir para criar um cluster em uma única Região da AWS.

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

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

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

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

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

main if $PROGRAM_NAME == __FILE__
```

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

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

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

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

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

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

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

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

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

```
    w.delay = 10
  end

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

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

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

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

main if $PROGRAM_NAME == __FILE__
```

.NET

Use o exemplo a seguir para criar um cluster em uma única Região da AWS.

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

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

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

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

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

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

                return response;
            }
        }

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

            await Create(region);
```

```
        }
    }
}
```

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

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

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

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

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

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

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

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

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

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

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

                return (cluster1, cluster2);
            }
        }

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

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

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

Golang

Use o exemplo a seguir para criar um cluster em uma única Região da AWS.

```
package main

import (
```

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

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

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

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

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

 deleteProtect := true

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

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

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

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

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

```
 id := clusterProperties.Identifier

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

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

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

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

 region := "us-east-1"

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

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

Para criar um cluster multirregional, use o exemplo a seguir. O processo de criação de clusters multirregionais pode levar algum tempo.

```
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)
 }

}
```

Obter um cluster

Consulte as informações a seguir para saber como exibir informações sobre um cluster no Aurora DSQL.

Python

Para ter informações sobre um cluster de região única ou multirregional, use o exemplo a seguir.

```
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 o exemplo a seguir para ter informações sobre um cluster de região única ou 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 ter informações sobre um cluster de região única ou multirregional, use o exemplo a seguir.

```
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

O exemplo a seguir permite que você obtenha informações sobre um cluster de região única ou 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

O exemplo a seguir permite que você obtenha informações sobre um cluster de região única ou 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

O exemplo a seguir permite que você obtenha informações sobre um cluster de região única ou 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

O exemplo a seguir permite que você obtenha informações sobre um cluster de região única ou 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

O exemplo a seguir permite que você obtenha informações sobre um cluster de região única ou 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)
```

```
 }
}
```

Atualizar um cluster

Consulte as informações a seguir para saber como atualizar um cluster no Aurora DSQL. A atualização de um cluster pode levar um ou dois minutos. Recomendamos que você espere algum tempo e execute get cluster para obter o status do cluster.

Python

Para atualizar um cluster único ou multirregional, use o exemplo a seguir.

```
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 o exemplo a seguir para atualizar um cluster único ou 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 atualizar um cluster único ou multirregional, use o exemplo a seguir.

```
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 o exemplo a seguir para atualizar um cluster de região única ou 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 o exemplo a seguir para atualizar um cluster de região única ou 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 o exemplo a seguir para atualizar um cluster de região única ou 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 o exemplo a seguir para atualizar um cluster de região única ou 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 o exemplo a seguir para atualizar um cluster de região única ou 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)
 }
}
```

Excluir um cluster

Consulte as informações a seguir para saber como atualizar um cluster no Aurora DSQL.

Python

Para excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

```
import boto3


def delete_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        cluster = client.delete_cluster(identifier=identifier)
        print(f"Initiated delete of {cluster["arn"]}")

        print("Waiting for cluster to finish deletion")
        client.get_waiter("cluster_not_exists").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )
    except:
        print("Unable to delete cluster " + identifier)
        raise


def main():
    region = "us-east-1"
    cluster_id = "<cluster id>"  # Use a placeholder in docs
    delete_cluster(region, cluster_id)
    print(f"Deleted {cluster_id}")


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

Para excluir um cluster multirregional, use o exemplo a seguir.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

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

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

/**
 * Deletes a single-region cluster in Amazon Aurora DSQL
 */
void DeleteCluster(const Aws::String& region, const Aws::String& identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Delete the cluster
    DeleteClusterRequest deleteRequest;
    deleteRequest.SetIdentifier(identifier);
    deleteRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome = client.DeleteCluster(deleteRequest);
```

```
    if (!deleteOutcome.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << identifier << " in " << region
 << ": "
                  << deleteOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to delete cluster " + identifier + " in " +
 region);
    }

    auto cluster = deleteOutcome.GetResult();
    std::cout << "Initiated delete of " << cluster.GetArn() << std::endl;
}

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

            DeleteCluster(region, clusterId);

            std::cout << "Deleted " << clusterId << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
#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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

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

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

```
  const client = new DSQLClient({ region });

  try {
    const deleteClusterCommand = new DeleteClusterCommand({
      identifier: clusterId,
    });
    const response = await client.send(deleteClusterCommand);

    console.log(`Waiting for cluster ${response.identifier} to finish deletion`);

    await waitUntilClusterNotExists(
      {
        client: client,
        maxWaitTime: 300 // Wait for 5 minutes
      },
      {
        identifier: response.identifier
      }
    );
    console.log(`Cluster Id ${response.identifier} is now deleted`);
    return;
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or already deleted");
    } else {
      console.error("Unable to delete cluster: ", error.message);
    }
    throw error;
  }
}

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

  await deleteCluster(region, clusterId);
}

main();
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

import java.time.Duration;

public class DeleteCluster {
```

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


        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            DeleteClusterResponse cluster = client.deleteCluster(r ->
 r.identifier(clusterId));
            System.out.println("Initiated delete of " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for deletion.
            System.out.println("Waiting for cluster to finish deletion");
            client.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            System.out.println("Deleted " + cluster.arn());
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

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

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

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

    Client::from_conf(config)
```

```
}

/// Delete a DSQL cluster
pub async fn delete_cluster(region: &'static str, identifier: &'static str) {
    let client = dsql_client(region).await;
    let delete_response = client
        .delete_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap();
    println!("Initiated delete of {}", delete_response.arn);

    println!("Waiting for cluster to finish deletion");
    client
        .wait_until_cluster_not_exists()
        .identifier(identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}

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

    delete_cluster(region, cluster_id).await;
    println!("Deleted {cluster_id}");

    Ok(())
}
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

```
equire "aws-sdk-dsql"

def delete_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.delete_cluster(identifier: identifier)
  puts "Initiated delete of #{cluster.arn}"
```

```
  # The DSQL SDK offers built-in waiters to poll for deletion.
  puts "Waiting for cluster to finish deletion"
  client.wait_until(:cluster_not_exists, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Unable to delete cluster #{identifier} in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  delete_cluster(region, cluster_id)
  puts "Deleted #{cluster_id}"
end

main if $PROGRAM_NAME == __FILE__
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

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

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

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

        /// <summary>
        /// Delete a DSQL cluster.
        /// </summary>
        public static async Task Delete(RegionEndpoint region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var deleteRequest = new DeleteClusterRequest
                {
                    Identifier = identifier
                };

                var deleteResponse = await client.DeleteClusterAsync(deleteRequest);
                Console.WriteLine($"Initiated deletion of {deleteResponse.Arn}");
            }
        }

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

            await Delete(region, clusterId);
        }
    }
}
```

Para excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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 excluir um cluster em uma única Região da AWS, use o exemplo a seguir.

```
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 excluir um cluster multirregional, use o exemplo a seguir. A exclusão de um cluster multirregional pode levar algum tempo.

```
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)
 }
}
```

Tutoriais

Os tutoriais e exemplos de código no GitHub apresentados abaixo ajudam você a executar tarefas comuns no Aurora DSQL.

- Usando o Benchbase com o Aurora DSQL: uma ramificação do utilitário de benchmarking de código aberto Benchbase que foi verificado para funcionar com o Aurora DSQL.
- Carregador do Aurora DSQL: esse script Python de código aberto facilita o carregamento de dados no Aurora DSQL para seus casos de uso, como preencher tabelas para testes ou transferir dados para o Aurora DSQL.
- Exemplos do Aurora DSQL: o repositório `aws-samples/aurora-dsql-samples` no GitHub contém exemplos de código de como conectar e usar o Aurora DSQL em várias linguagens de programação utilizando os SDKs da AWS, mapeadores de objetos relacionais (ORMs) e frameworks da web. Os exemplos demonstram como realizar tarefas comuns, como instalar clientes, lidar com a autenticação e realizar operações CRUD.

Usar o AWS Lambda com o Amazon Aurora DSQL

O tutorial a seguir descreve como usar o Lambda com o Aurora DSQL.

Pré-requisitos

- Autorização para criar funções do Lambda. Para ter mais informações, consulte Criar sua primeira função do Lambda.
- Autorização para criar ou modificar a política do IAM criada pelo Lambda. Você precisa das permissões `iam:CreatePolicy` e `iam:AttachRolePolicy`. Para ter mais informações, consulte Actions, resources, and condition keys for IAM.
- Você deve ter instalado o npm v8.5.3 ou posterior.
- Você deve ter instalado o npm v3.0 ou posterior.

Crie uma função no AWS Lambda.

1. Faça login no AWS Management Console e abra o console do AWS Lambda em https://console.aws.amazon.com/lambda/.
2. Escolha a opção Criar função.
3. Forneça um nome, como `dsql-sample`.
4. Não edite as configurações padrão para garantir que o Lambda crie uma função com permissões básicas do Lambda.
5. Escolha a opção Criar função.

Autorize sua função de execução do Lambda a se conectar ao seu cluster.

1. Em sua função do Lambda, escolha Configuração > Permissões.
2. Escolha o nome do perfil para abrir o perfil de execução no console do IAM.
3. Escolha Adicionar permissões > Criar política em linha.
4. Em Ação, cole a ação a seguir para autorizar a identidade do IAM a se conectar usando o perfil de administrador do banco de dados.

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

> **Note**
>
> Estamos usando um perfil de administrador para minimizar as etapas necessárias para começar. Você não deve usar um perfil de administrador de banco de dados para suas aplicações de produção. Consulte Usar perfis de banco de dados e autenticação do IAM para saber como criar perfis de banco de dados personalizados com autorização que tenha o menor número de permissões para o banco de dados.

5. Em Recurso, adicione o nome do recurso da Amazon (ARN) do cluster. Você também pode usar caracteres curinga.

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

6. Escolha Próximo.
7. Insira um nome para a política, como `dsql-sample-dbconnect`.
8. Escolha Criar política.

Crie um pacote para fazer upload no Lambda.

1. Crie uma pasta denominada `myfunction`.
2. Na pasta, crie um arquivo denominado `package.json` com o conteúdo a seguir.

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

3. Na pasta, crie um arquivo denominado `index.mjs` no diretório com o conteúdo a seguir.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";
import pg from "pg";
import assert from "node:assert";
const { Client } = pg;


async function dsql_sample(clusterEndpoint, region) {
  let client;
  try {
    // The token expiration time is optional, and the default value 900 seconds
    const signer = new DsqlSigner({
      hostname: clusterEndpoint,
      region,
    });
    const token = await signer.getDbConnectAdminAuthToken();
    // <https://node-postgres.com/apis/client>
    // By default `rejectUnauthorized` is true in TLS options
    // <https://nodejs.org/api/tls.html#tls_tls_connect_options_callback>
    // The config does not offer any specific parameter to set sslmode to verify-
full
    // Settings are controlled either via connection string or by setting
    // rejectUnauthorized to false in ssl options
    client = new Client({
      host: clusterEndpoint,
      user: "admin",
      password: token,
      database: "postgres",
      port: 5432,
```

```
      // <https://node-postgres.com/announcements> for version 8.0
      ssl: true,
      rejectUnauthorized: false
    });

    // Connect
    await client.connect();

    // Create a new table
    await client.query(`CREATE TABLE IF NOT EXISTS owner (
      id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
      name VARCHAR(30) NOT NULL,
      city VARCHAR(80) NOT NULL,
      telephone VARCHAR(20)
    )`);

    // Insert some data
    await client.query("INSERT INTO owner(name, city, telephone) VALUES($1, $2,
 $3)",
      ["John Doe", "Anytown", "555-555-1900"]
    );

    // Check that data is inserted by reading it back
    const result = await client.query("SELECT id, city FROM owner where name='John
 Doe'");
    assert.deepEqual(result.rows[0].city, "Anytown")
    assert.notEqual(result.rows[0].id, null)

    await client.query("DELETE FROM owner where name='John Doe'");

  } catch (error) {
    console.error(error);
    throw new Error("Failed to connect to the database");
  } finally {
    client?.end();
  }
  Promise.resolve();
}


// https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html
export const handler = async (event) => {
  const endpoint = event.endpoint;
  const region = event.region;
```

```
  const responseCode = await dsql_sample(endpoint, region);

  const response = {
    statusCode: responseCode,
    endpoint: endpoint,
  };
  return response;
};
```

4. Use os comandos a seguir para criar um pacote.

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

Faça upload do pacote de código e teste a função do Lambda.

1. Na guia Código da função do Lambda, escolha Fazer upload de > arquivo .zip.
2. Faça o upload do `pkg.zip` que você criou. Para ter mais informações, consulte Implantar funções do Lambda em Node.js com arquivos .zip.
3. Na guia Teste da função do Lambda, cole a carga útil JSON a seguir e modifique-a para usar o ID do seu cluster.
4. Na guia Teste da função do Lambda, use o evento JSON modificado a seguir para especificar o endpoint do seu cluster.

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

5. Insira um nome de evento, como `dsql-sample-test`. Escolha Salvar.
6. Escolha Test (Testar).
7. Escolha Detalhes para expandir a resposta de execução e a saída do log.
8. Se bem-sucedida, a resposta da execução da função do Lambda deverá exibir um código de status 200.

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

Se o banco de dados exibir um erro ou se a conexão com o banco de dados falhar, a resposta de execução da função do Lambda exibirá um código de status 500.

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

Backup e restauração para o Amazon Aurora DSQL

O Amazon Aurora DSQL ajuda você a atender aos requisitos de conformidade regulatória e continuidade de negócios por meio da integração com o AWS Backup, um serviço de proteção de dados totalmente gerenciado que facilita a centralização e a automação de backups por serviços da AWS, na nuvem e on-premises. O serviço simplifica a criação, o gerenciamento e a restauração de backups para clusters Aurora DSQL de região única e de várias regiões.

Os principais recursos incluem o seguinte:

- Gerenciamento centralizado de backup por meio do AWS Management Console, SDK ou AWS CLI
- Backups completos de cluster
- Programações automatizadas de backup e políticas de retenção
- Capacidades entre regiões e entre contas
- Configuração WORM (write once, read-many) para todos os backups que você armazena

Para obter mais informações sobre os recursos do AWS Backup Vault Lock e uma lista extensa de recursos do AWS Backup disponíveis para o Aurora DSQL, consulte Benefícios do Vault Lock e Disponibilidade de recursos do AWS Backup no Guia do desenvolvedor do AWS Backup.

Conceitos básicos do AWS Backup

O AWS Backup cria cópias completas dos seus clusters do Aurora DSQL. Você pode começar a usar o AWS Backup para Aurora DSQL seguindo as etapas em Introdução ao AWS Backup:

1. Criar backups sob demanda para proteção imediata.
2. Estabelecer planos de backup para backups automatizados e programados.
3. Configurar períodos de retenção e cópia entre regiões.
4. Configure o monitoramento e as notificações para atividades de backup.

Restaurar seus backups

Quando você restaura clusters do Aurora DSQL, o AWS Backup sempre cria novos clusters para preservar seus dados de origem. Para restaurar um cluster de região única do Aurora DSQL, use o

https://console.aws.amazon.com/backup ou a CLI para selecionar o ponto de recuperação (backup) que você deseja restaurar. Defina as configurações do novo cluster que será criado a partir do seu backup.

A restauração de um cluster multirregional do Aurora DSQL só é suportada por meio da AWS CLI. Para restaurar um cluster multirregional do Aurora DSQL, você precisa usar tanto o AWS Backup quanto a CLI do Aurora DSQL.

Para restaurar um cluster multirregional do Aurora DSQL:

1. Selecione o ponto de recuperação do seu cluster multirregional.
2. Copie o ponto de recuperação para outra Região da AWS que ofereça suporte a clusters multirregionais.

   > **Note**
   >
   > As regiões que não oferecem suporte a clusters multirregionais resultarão em uma operação de restauração com falha.

3. Inicie um trabalho de restauração para cada cluster usando a CLI do AWS Backup.
4. Use a documentação Configurar clusters multirregionais para emparelhar os clusters do Aurora DSQL recém-criados.

Para obter instruções detalhadas sobre essas etapas, consulte a documentação de restauração do Amazon Aurora DSQL.

Para restaurar em um cluster do Aurora DSQL multirregional, você pode usar um backup feito em uma única Região da AWS. No entanto, antes de iniciar o processo de restauração, primeiro você deve copiar o backup para outra Região da AWS que ofereça suporte a clusters multirregionais. Essa etapa garante que a operação de restauração possa ser concluída com êxito. Recomendamos criar cópias de backup nas Regiões da AWS principais, como Leste dos EUA (Norte da Virgínia), Leste dos EUA (Ohio) ou Oeste dos EUA (Oregon), para permitir opções robustas de recuperação de desastres e atender aos requisitos de conformidade.

Monitoramento e conformidade

O AWS Backup fornece visibilidade abrangente das operações de backup e restauração com os seguintes recursos.

- Um painel centralizado para rastrear trabalhos de backup e restauração
- Integração com o CloudWatch e o CloudTrail.
- AWS Backup Audit Manager para relatórios de conformidade e auditoria.

Consulte Registrar em log operações do Aurora DSQL usando a AWS CloudTrail para saber mais sobre o registro de ações executadas por um usuário, um perfil ou um AWS service (Serviço da AWS) durante o uso do Aurora DSQL.

Recursos adicionais

Para saber mais sobre os recursos do AWS Backup e seu uso em conjunto com o Aurora DSQL, consulte os seguintes recursos:

- Políticas gerenciadas para o AWS Backup
- Restauração do Amazon Aurora DSQL
- Serviços compatíveis por Região da AWS
- Criptografia para backups no AWS Backup

Ao usar o AWS Backup para Aurora DSQL, você implementa uma estratégia de backup robusta, compatível e automatizada que protege seus recursos essenciais de banco de dados e minimiza a sobrecarga administrativa. Se você gerencia um único cluster ou uma implantação complexa em várias regiões, o AWS Backup fornece as ferramentas necessárias para garantir que seus dados permaneçam seguros e recuperáveis.

Monitoramento e registro em log do Aurora DSQL

O monitoramento e o registro em log são uma parte importante para manter a confiabilidade, a disponibilidade e o desempenho dos recursos do Amazon Aurora. Você deve monitorar e coletar dados de registro em log de todas as partes dos recursos do Aurora DSQL para facilitar a depuração de uma falha em vários pontos.

- O Amazon CloudWatch monitora os recursos da AWS e as aplicações que você executa na AWS em tempo real. Você pode coletar e rastrear métricas, criar painéis personalizados e definir alarmes que o notificam ou que realizam ações quando uma métrica especificada atinge um limite definido. Por exemplo, você pode fazer o CloudWatch acompanhar o uso da CPU ou outras métricas das instâncias do Amazon EC2 e iniciar automaticamente novas instâncias quando necessário. Para obter mais informações, consulte o Guia do usuário do Amazon CloudWatch.
- O AWS CloudTrail captura chamadas de API e eventos relacionados feitos por sua conta da Conta da AWS ou em nome dela e entrega os arquivos de log a um bucket do Amazon S3 que você especificar. Você pode identificar quais usuários e contas chamaram AWS, o endereço IP de origem de onde as chamadas foram feitas e quando elas ocorreram. Para obter mais informações, consulte o Guia do usuário do AWS CloudTrail.

Visualizar o status do cluster do Aurora DSQL

O status do cluster do Aurora DSQL fornece informações essenciais sobre a integridade e a conectividade do cluster. É possível visualizar o status de clusters e de instâncias do cluster usando o AWS Management Console, a AWS CLI ou a API do Aurora DSQL.

Status e definições de cluster do Aurora DSQL

A tabela a seguir descreve cada status possível para um cluster do Aurora DSQL e o que cada status significa.

Status	Descrição
Criando	O Aurora DSQL está tentando criar ou configurar recursos para o cluster. Qualquer tentativa de conexão falhará enquanto o cluster estiver nesse estado.
Ativo	O cluster está operacional e pronto para uso.

Status	Descrição
Ocioso	Um cluster fica ocioso durante o tempo necessário para o Aurora DSQL recuperar os recursos configurados para ele. Quando você se conecta a um cluster ocioso, o Aurora DSQL faz a transição do cluster de volta para o estado Ativo.
Inativo	Um cluster fica inativo quando não há atividade no cluster por um período prolongado. Quando você tenta se conectar a um cluster inativo, o Aurora DSQL faz a transição automática do cluster de volta para o estado Ativo.
Atualizando	Um cluster faz a transição para o status Atualizando quando são feitas alterações na respectiva configuração.
Deleting	Um cluster faz a transição para o status Excluindo quando é enviada uma solicitação para excluí-lo.
Excluído	O cluster foi excluído com êxito.
Com falha	O Aurora DSQL não pôde criar o cluster porque encontrou um erro.
Configuração pendente	Somente para clusters multirregionais. Um cluster multirregional entra no status Configuração pendente quando você cria um cluster multirregional em sua primeira região com uma região testemunha. A criação do cluster é pausada até que você crie outro cluster em uma região secundária e emparelhe ambos.
Exclusão pendente	Somente para clusters multirregionais. Um cluster multirregional entra no status Exclusão pendente quando você exclui um cluster dele. O cluster passa para o estado Excluindo quando você exclui o último cluster emparelhado.

Visualizar o status do cluster do Aurora DSQL

Para visualizar o status de um cluster, use o AWS Management Console, a AWS CLI ou a API do Aurora DSQL.

Console

Siga estas etapas para visualizar o status de um cluster no AWS Management Console:

Como visualizar o status de um cluster no console

1. Abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.
2. Selecione Clusters no painel de navegação.
3. Visualize o status de cada cluster no painel.

AWS CLI

Use o comando da AWS CLI a seguir para verificar o status do cluster.

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

Execute o comando a seguir para listar o status de todos os clusters.

```
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
```

Esta saída de exemplo mostra dois clusters ativos e um cluster em processo de exclusão.

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

Monitorar o Aurora DSQL com o Amazon CloudWatch

Monitorar o Aurora DSQL usando o CloudWatch, que coleta dados brutos e os processa em métricas legíveis praticamente em tempo real. O CloudWatch mantém essas estatísticas por 15 meses para ajudar você a ter uma visão melhor do desempenho de sua aplicação ou serviço da web. Defina alarmes para monitorar limites específicos e enviar notificações ou realizar ações quando eles são atingidos. Exame abaixo as métricas de uso e observabilidade disponíveis para o Aurora DSQL.

Para obter mais informações, consulte o Guia do usuário do Amazon CloudWatch.

Observabilidade e desempenho

Esta tabela descreve as métricas de observabilidade do Aurora DSQL. Ela inclui métricas para rastrear transações somente leitura e o total de transações e oferecer uma caracterização geral da workload. Métricas práticas, como tempos limite de consulta e taxa de conflito de OCC, estão incluídas para ajudar a identificar problemas de desempenho e conflitos de simultaneidade. As métricas relacionadas a sessões, tanto ativas quanto totais, oferecem informações sobre a carga atual no sistema.

Nome da métrica do CloudWatch	Métrica	Unidade	Descrição
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

Nome da métrica do CloudWatch	Métrica	Unidade	Descrição
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

O Aurora DSQL mede todas as atividades baseadas em solicitações, como processamento de consultas, leituras e gravações, usando uma única unidade de cobrança normalizada chamada unidade de processamento distribuído (DPU).

Nome da métrica do CloudWatch	Métrica	Dimensão: ResourceId	Unidade	Descrição
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.

Nome da métrica do CloudWatch	Métrica	Dimensão: ResourceId	Unidade	Descrição
ReadDPU	Read Units	<cluster-id>	DPU	Approximates the read active-use component of your Aurora DSQL cluster DPU usage.
ComputeDPU	Compute Units	<cluster-id>	DPU	Approximates the compute active-use component of your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

Registrar em log operações do Aurora DSQL usando a AWS CloudTrail

O Amazon Aurora DSQL é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, um perfil ou um AWS service (Serviço da AWS). Há dois tipos de evento no CloudTrail: de gerenciamento e de dados. Os eventos de gerenciamento são emitidos para auditar alterações na configuração de recursos da AWS. Os eventos de dados normalmente capturam o uso de recursos da AWS no plano de dados do serviço.

O CloudTrail captura todas as chamadas de API para o Aurora DSQL como eventos. O Aurora DSQL registra a atividade do console como eventos de gerenciamento. Ele também captura tentativas de conexão autenticadas com clusters como eventos de dados.

Usando as informações coletadas pelo CloudTrail, é possível determinar a solicitação feita para o Aurora DSQL, o endereço IP no qual a solicitação foi feita, quando ela foi feita, a identidade do usuário que a fez e detalhes adicionais.

O CloudTrail é habilitado por padrão em sua Conta da AWS quando você cria a conta e tem acesso ao Histórico de eventos do CloudTrail. O Histórico de eventos do CloudTrail fornece um registro visualizável, pesquisável, baixável e imutável dos últimos 90 dias de eventos de gerenciamento gravados em uma Região da AWS. Para obter mais informações, consulte Trabalhar com histórico de eventos do CloudTrail no Guia do usuário do AWS CloudTrail. Não há cobranças do CloudTrail pelo registro no Histórico de eventos.

Para criar um registro contínuo de eventos em sua conta da AWS, incluindo eventos para o Aurora DSQL, crie uma trilha ou um datastore de eventos do AWS CloudTrail Lake (uma solução centralizada de armazenamento e análise para eventos do AWS CloudTrail). Para ter mais informações sobre trilhas, consulte Working with CloudTrail trails. Para saber mais sobre como configurar e gerenciar datastores de eventos, consulte CloudTrail Lake event data stores.

Eventos de gerenciamento do Aurora DSQL no CloudTrail

Os eventos de gerenciamento do CloudTrail fornecem informações sobre operações de gerenciamento executadas em recursos na sua conta da AWS. Também são conhecidas como operações de ambiente de gerenciamento. Por padrão, o CloudTrail captura eventos de gerenciamento no Histórico de eventos.

O Amazon Aurora DSQL registra em log todas as operações do ambiente de gerenciamento como eventos de gerenciamento. Para ver uma lista de operações do ambiente de gerenciamento do Amazon Aurora DSQL que o Aurora DSQL registra em log no CloudTrail, consulte Aurora DSQL API reference.

Logs do ambiente de gerenciamento

O Amazon Aurora DSQL registra em log as operações a seguir do ambiente de gerenciamento do Aurora DSQL no CloudTrail como eventos de gerenciamento.

- CreateCluster

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

Logs de backup e restauração

O Amazon Aurora DSQL registra em log as operações de backup e restauração do Aurora DSQL a seguir no CloudTrail como eventos de gerenciamento.

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

Para saber mais sobre como proteger clusters do Aurora DSQL usando o AWS Backup, consulte Backup e restauração para o Amazon Aurora DSQL.

Logs do AWS KMS

O Amazon Aurora DSQL registra em log as operações do AWS KMS a seguir no CloudTrail como eventos de gerenciamento.

- `GenerateDataKey`
- `Decrypt`

Para saber mais sobre como os logs do CloudTrail rastreiam as solicitações que o Aurora DSQL envia ao AWS KMS em seu nome, consulte Monitorar a interação do Aurora DSQL com o AWS KMS.

Eventos de dados do Aurora DSQL no CloudTrail

Os eventos de dados do CloudTrail normalmente fornecem informações sobre as operações aplicadas a um recurso ou realizadas em um recurso. Eles também são usados para capturar as operações do plano de dados do serviço. Eventos de dados geralmente são atividades de alto volume. Por padrão, o CloudTrail não registra eventos de dados em log. O Histórico de eventos do CloudTrail não registra eventos de dados.

Para obter mais informações sobre como registrar eventos de dados em log, consulte Registrar eventos de dados com o AWS Management Console e Registrar eventos de dados com a AWS Command Line Interface no Guia do usuário do AWS CloudTrail.

Há cobranças adicionais para eventos de dados. Para obter mais informações sobre os preços do CloudTrail, consulte Preços do AWS CloudTrail.

Para o Aurora DSQL, o CloudTrail captura qualquer tentativa de conexão feita a um cluster do Aurora DSQL como um evento de dados. A tabela a seguir lista os tipos de recurso do Aurora DSQL para os quais você pode registrar eventos de dados em log. A coluna Tipo de recurso (console) mostra o valor a ser escolhido na lista Tipo de recurso no console do CloudTrail. A coluna do valor resources.type mostra o valor de `resources.type` que você especificaria ao configurar seletores de eventos avançados usando a AWS CLI ou as APIs do CloudTrail. A coluna APIs de dados registradas no CloudTrail mostra as chamadas de API registradas no CloudTrail para o tipo de recurso.

Tipo de recurso (console)	valor resources.type	APIs de dados registradas no CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Você pode configurar seletores de eventos avançados para filtrar os campos `eventName` e `resources.ARN` e registrar em log somente os eventos de interesse. Para obter mais informações sobre esses campos, consulte AdvancedFieldSelector na Referência de API do AWS CloudTrail.

O exemplo a seguir mostra como usar a AWS CLI para configurar `dsql-data-events-trail` e receber eventos de dados referentes ao 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"] } ]}]'
```

Segurança no Amazon Aurora DSQL

A segurança na nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você se beneficiará de data centers e arquiteturas de rede criados para atender aos requisitos das empresas com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isso como a segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa os serviços da AWS na Nuvem AWS. A AWS também fornece serviços que podem ser usados com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos Programas de conformidade da AWS. Para saber mais sobre os programas de conformidade que se aplicam ao Amazon Aurora DSQL, consulte Serviços da AWS em escopo por programa de conformidade.
- Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda a entender como aplicar o Modelo de Responsabilidade Compartilhada ao usar o Aurora DSQL. Os tópicos a seguir mostram como configurar o Aurora DSQL para atender aos seus objetivos de segurança e conformidade. Saiba também como usar outros serviços da AWS que ajudam a monitorar e proteger os recursos do Aurora DSQL.

Tópicos

- Segurança da infraestrutura no Amazon Aurora DSQL
- Análise de configuração e vulnerabilidade no Amazon Aurora DSQL
- Prevenção contra o ataque do “substituto confuso” em todos os serviços
- Práticas recomendadas de segurança para o Aurora DSQL

Políticas gerenciadas pela AWS para o Amazon Aurora DSQL

Uma política gerenciada pela AWS é uma política independente criada e administrada pela AWS. As políticas gerenciadas pela AWS são criadas para fornecer permissões a vários casos de uso comuns e permitir a atribuição de permissões a usuários, grupos e perfis.

Lembre-se de que as políticas gerenciadas pela AWS podem não conceder permissões de privilégio mínimo para casos de uso específicos, por estarem disponíveis para uso por todos os clientes da AWS. Recomendamos que você reduza ainda mais as permissões definindo as políticas gerenciadas pelo cliente que são específicas para seus casos de uso.

Você não pode alterar as permissões definidas em políticas gerenciadas pela AWS. Se a AWS atualiza as permissões definidas em um política gerenciada por AWS, a atualização afeta todas as identidades de entidades principais (usuários, grupos e perfis) às quais a política estiver vinculada. É provável que a AWS atualize uma política gerenciada por AWS quando um novo AWS service (Serviço da AWS) for lançado, ou novas operações de API forem disponibilizadas para os serviços existentes.

Para mais informações, consulte Políticas gerenciadas pela AWS no Manual do usuário do IAM.

Política gerenciada pela AWS: AmazonAuroraDSQLFullAccess

Você pode anexar `AmazonAuroraDSQLFullAccess` aos seus usuários, grupos e funções.

Essa política concede permissões administrativas que autorizam acesso administrativo completo ao Aurora DSQL. As entidades principais que têm essas permissões podem criar, excluir e atualizar clusters do Aurora DSQL, inclusive clusters multirregionais. Elas podem adicionar e remover tags dos clusters. Podem listar clusters e visualizar informações sobre clusters individuais. Podem ver

as tags anexadas aos clusters do Aurora DSQL. Podem se conectar ao banco de dados como qualquer usuário, inclusive administrador. Podem realizar operações de backup e restauração para clusters do Aurora DSQL, incluindo iniciar, interromper e monitorar tarefas de backup e restauração. A política inclui permissões do AWS KMS que autorizam operações em chaves gerenciadas pelo cliente usadas para criptografia de cluster. Podem ver quaisquer métricas do CloudWatch em sua conta. Elas também têm permissões para criar perfis vinculados ao serviço para o serviço `dsql.amazonaws.com`, o que é necessário para criar clusters.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- `dsql`: concede às entidades principais acesso total ao Aurora DSQL.
- `cloudwatch`: concede permissão para publicar pontos de dados de métrica no Amazon CloudWatch.
- `iam`: concede permissão para criar um perfil vinculado ao serviço.
- `backup and restore`: concede permissões para iniciar, interromper e monitorar tarefas de backup e restauração para clusters do Aurora DSQL.
- `kms`: concede as permissões necessárias para validar o acesso às chaves gerenciadas pelo cliente usadas para a criptografia de cluster do Aurora DSQL ao criar, atualizar ou se conectar a clusters.

Você pode encontrar a política `AmazonAuroraDSQLFullAccess` no console do IAM e AmazonAuroraDSQLFullAccess no “Guia de referência de políticas gerenciadas pela AWS”.

Política gerenciada pela AWS: AmazonAuroraDSQLReadOnlyAccess

Você pode anexar `AmazonAuroraDSQLReadOnlyAccess` aos seus usuários, grupos e funções.

Permite acesso de leitura ao Aurora DSQL. As entidades principais que têm essas permissões podem listar clusters e visualizar informações sobre clusters individuais. Podem ver as tags anexadas aos clusters do Aurora DSQL. Podem recuperar e ver quaisquer métricas do CloudWatch em sua conta.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- `dsql`: concede permissões somente de leitura a todos os recursos no Aurora DSQL.
- `cloudwatch`: concede permissão para recuperar valores em lote de dados de métrica do CloudWatch e executar matemática de métrica nos dados recuperados.

Você pode encontrar a política `AmazonAuroraDSQLReadOnlyAccess` no console do IAM e AmazonAuroraDSQLReadOnlyAccess no “Guia de referência de políticas gerenciadas pela AWS”.

Política gerenciada pela AWS: AmazonAuroraDSQLConsoleFullAccess

Você pode anexar `AmazonAuroraDSQLConsoleFullAccess` aos seus usuários, grupos e funções.

Permite acesso administrativo total ao Amazon Aurora DSQL por meio do AWS Management Console. As entidades principais que têm essas permissões podem criar, excluir e atualizar clusters do Aurora DSQL, inclusive clusters multirregionais, com o console. Podem listar clusters e visualizar informações sobre clusters individuais. Podem ver as tags em qualquer recurso da sua conta. Podem se conectar ao banco de dados como qualquer usuário, inclusive administrador. Podem realizar operações de backup e restauração para clusters do Aurora DSQL, incluindo iniciar, interromper e monitorar tarefas de backup e restauração. A política inclui permissões do AWS KMS que autorizam operações em chaves gerenciadas pelo cliente usadas para criptografia de cluster. Podem iniciar o AWS CloudShell por meio do AWS Management Console. Podem ver quaisquer métricas do CloudWatch em sua conta. Elas também têm permissões para criar perfis vinculados ao serviço para o serviço `dsql.amazonaws.com`, o que é necessário para criar clusters.

Você pode encontrar a política `AmazonAuroraDSQLConsoleFullAccess` no console do IAM e AmazonAuroraDSQLConsoleFullAccess no “Guia de referência de políticas gerenciadas pela AWS”.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- `dsql`: concede permissões administrativas completas a todos os recursos no Aurora DSQL por meio do AWS Management Console.
- `cloudwatch`: concede permissão para recuperar valores em lote de dados de métrica do CloudWatch e executar matemática de métrica nos dados recuperados.
- `tag`: concede permissão para retornar as chaves de tag em uso no momento na Região da AWS especificada para a conta que está fazendo a chamada.
- `backup and restore`: concede permissões para iniciar, interromper e monitorar tarefas de backup e restauração para clusters do Aurora DSQL.
- `kms`: concede as permissões necessárias para validar o acesso às chaves gerenciadas pelo cliente usadas para a criptografia de cluster do Aurora DSQL ao criar, atualizar ou se conectar a clusters.
- `cloudshell`: concede permissões para iniciar o AWS CloudShell e interagir com o Aurora DSQL.
- `ec2`: concede permissão para visualizar as informações de endpoint da Amazon VPC necessárias para conexões do Aurora DSQL.

Você pode encontrar a política `AmazonAuroraDSQLReadOnlyAccess` no console do IAM e AmazonAuroraDSQLReadOnlyAccess no “Guia de referência de políticas gerenciadas pela AWS”.

Política gerenciada pela AWS: AuroraDSQLServiceRolePolicy

Não é possível anexar AuroraDSQLServiceRolePolicy às suas entidades do IAM. Essa política é anexada a um perfil vinculado ao serviço que permite que o Aurora DSQL acesse recursos da conta.

Você pode encontrar a política `AuroraDSQLServiceRolePolicy` no console do IAM e AuroraDSQLServiceRolePolicy no “Guia de referência de políticas gerenciadas pela AWS”.

Atualizações do Aurora DSQL nas políticas gerenciadas pela AWS

Visualize detalhes sobre atualizações em políticas gerenciadas pela AWS para o Aurora DSQL desde que esse serviço começou a rastrear essas alterações. Para receber alertas automáticos sobre alterações realizadas nessa página, assine o feed RSS na página de histórico do documento do Aurora DSQL.

Alteração	Descrição	Data
Atualização de AmazonAuroraDSQLFullAccess	Adiciona a capacidade de realizar operações de backup e restauração para clusters do Aurora DSQL, incluindo iniciar, interromper e monitorar tarefas. Também adiciona a capacidade de usar chaves do KMS gerenciadas pelo cliente para criptografia de cluster. Para ter mais informações, consulte AmazonAuroraDSQLFullAccess e Usar perfis vinculados ao serviço no Aurora DSQL.	21 de maio de 2025
Atualização de AmazonAuroraDSQLConsoleFullAccess	Adiciona a capacidade de realizar operações de backup e restauração para clusters do Aurora DSQL por meio do AWS Console Home. Isso inclui iniciar, interromper e monitorar tarefas. Também é possível usar chaves do KMS gerenciadas pelo cliente para criptografia de cluster e para iniciar o AWS CloudShell. Para ter mais informações, consulte AmazonAuroraDSQLConsoleFullAccess e Using service-linked roles in Aurora DSQL.	21 de maio de 2025

Alteração	Descrição	Data
Atualização de AmazonAuroraDSQLFullAccess	A política adiciona quatro novas permissões para criar e gerenciar clusters de banco de dados em várias Regiões da AWS: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` e `RemovePeerCluster` . Essas permissões incluem controles em nível de recursos e chaves de condição para que você possa controlar quais clusters os usuários podem modificar. A política também adiciona a permissão `GetVpcEndpointServiceName` para ajudar você a se conectar aos clusters do Aurora DSQL por meio do AWS PrivateLink. Para ter mais informações, consulte AmazonAuroraDSQLFullAccess e Using service-linked roles in Aurora DSQL.	13 de maio de 2025

Alteração	Descrição	Data
Atualização de AmazonAuroraDSQLReadOnlyAccess	Inclui a capacidade de determinar o nome correto do serviço de endpoint da VPC ao se conectar aos clusters do Aurora DSQL por meio do AWS PrivateLink. O Aurora DSQL cria endpoints exclusivos por célula; portanto, essa API ajuda a garantir que você possa identificar o endpoint correto para o cluster e evitar erros de conexão. Para ter mais informações, consulte AmazonAuroraDSQLReadOnlyAccess e Using service-linked roles in Aurora DSQL.	13 de maio de 2025

Alteração	Descrição	Data
Atualização de AmazonAuroraDSQLConsoleFullAccess	Adiciona novas permissões ao Aurora DSQL para oferecer suporte ao gerenciamento de clusters multirregionais e à conexão de endpoints da VPC. As novas permissões incluem: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` , `RemovePeerCluster` e `GetVpcEndpointServiceName` . Para ter mais informações, consulte AmazonAuroraDSQLConsoleFullAccess e Using service-linked roles in Aurora DSQL.	13 de maio de 2025

Alteração	Descrição	Data
Atualização de AuroraDsqlServiceLinkedRolePolicy	Adiciona à política a capacidade de publicar métricas nos namespaces `AWS/AuroraDSQL` e `AWS/Usage` CloudWatch. Isso permite que o serviço ou perfil associado emita dados de uso e desempenho mais abrangentes para seu ambiente do CloudWatch. Para ter mais informações, consulte AuroraDsqlServiceLinkedRolePolicy e Using service-linked roles in Aurora DSQL.	8 de maio de 2025
Página criada	Começou a monitorar políticas gerenciadas pela AWS relacionadas ao Amazon Aurora DSQL.	3 de dezembro de 2024

Proteção de dados no Amazon Aurora DSQL

O Modelo de Responsabilidade Compartilhada aplica a proteção de dados no Amazon Aurora DSQL. Conforme descrito nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos que usa. Para obter mais informações sobre a privacidade de dados, consulte as Data Privacy FAQ. Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog Shared Responsibility Model and RGPD no Blog de segurança da .

Para fins de proteção de dados, é recomendável que você proteja as credenciais e configure usuários individuais com o AWS IAM Identity Center ou o AWS Identity and Access Management. Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas

obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos da . Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure os logs de API e atividade do usuário com AWS CloudTrail. Para ter informações sobre como usar trilhas para capturar atividades, consulte Working with trails no Guia do usuário do AWS CloudTrail.
- Use as soluções de criptografia, bem como todos os controles de segurança padrão nos Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui trabalhar com ou outros serviços usando o console, a API, a AWS CLI ou os SDKs da AWS. Quaisquer dados inseridos em tags ou em campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Criptografia de dados

O Amazon Aurora DSQL oferece uma infraestrutura de armazenamento resiliente projetada para armazenamento de dados essenciais e primários. Os dados são armazenados de maneira redundante em vários dispositivos de diversas instalações em uma região do Aurora DSQL.

Criptografia em trânsito

Por padrão, a criptografia em trânsito é configurada para você. O Aurora DSQL usa TLS para criptografar todo tráfego entre o cliente SQL e o Aurora DSQL.

Criptografia e assinatura de dados em trânsito entre os clientes da AWS CLI, do SDK ou da API e os endpoints do Aurora DSQL:

- O Aurora DSQL fornece endpoints HTTPS para criptografar dados em trânsito.

- Para proteger a integridade das solicitações de API para o Aurora DSQL, as chamadas de API devem ser assinadas pelo autor da chamada. As chamadas são assinadas por um certificado X.509 ou pela chave de acesso secreta AWS de acordo com o Processo de assinatura do Signature versão 4 (Sigv4). Para obter mais informações, consulte o Processo de assinatura do Signature versão 4 em Referência geral da AWS.
- Use a AWS CLI ou um dos AWS SDKs para fazer solicitações à AWS. Essas ferramentas autenticam automaticamente as solicitações para você com a chave de acesso especificada na configuração das ferramentas.

Sobre a criptografia em repouso, consulte Criptografia em repouso no Aurora DSQL.

Privacidade do tráfego entre redes

As conexões são protegidas entre o Aurora DSQL e as aplicações on-premises e entre o Aurora DSQL e outros recursos da AWS na mesma Região da AWS.

Você tem duas opções de conectividade entre sua rede privada e a AWS:

- Uma conexão AWS Site-to-Site VPN. Para ter mais informações, consulte O que é o AWS Site-to-Site VPN?
- Uma conexão AWS Direct Connect. Para ter mais informações, consulte O que é o AWS Direct Connect?

Você obtém acesso ao Aurora DSQL pela rede usando operações de API publicadas pela AWS. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Configurar certificados SSL/TLS para conexões do Aurora DSQL

O Aurora DSQL requer que todas as conexões usem a criptografia Transport Layer Security (TLS). Para estabelecer conexões seguras, seu sistema cliente deve confiar na Amazon Root Certificate Authority (Amazon Root CA 1). Esse certificado vem pré-instalado em vários sistemas operacionais.

Esta seção oferece instruções para verificar o certificado da Amazon Root CA 1 pré-instalado em vários sistemas operacionais e orienta você pelo processo de instalação manual do certificado, caso ele ainda não esteja instalado.

Recomendamos usar o PostgreSQL versão 17.

> **Important**
>
> Para ambientes de produção, recomendamos usar o modo SSL `verify-full` para garantir o mais alto nível de segurança de conexão. Esse modo verifica se o certificado do servidor está assinado por uma autoridade de certificação confiável e se o nome do host do servidor corresponde ao certificado.

Verificar certificados pré-instalados

Na maioria dos sistemas operacionais, o Amazon Root CA 1 já vem pré-instalado. Para confirmar, você pode seguir as etapas abaixo.

Linux (RedHat/CentOS/Fedora)

Execute os seguintes comandos no terminal:

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

Se o certificado estiver instalado, você verá a seguinte saída:

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

macOS

1. Abra o campo de pesquisa do Spotlight (comando + espaço).
2. Pesquise Keychain Access.
3. Selecione System Roots em System Keychains.
4. Pesquise Amazon Root CA 1 na lista de certificados.

Windows

> **Note**
>
> Devido a um problema conhecido com o cliente psql do Windows, o uso de certificados raiz do sistema (`sslrootcert=system`) pode retornar o seguinte erro: `SSL error: unregistered scheme`. Você pode seguir a seção Conectar-se pelo Windows como alternativa para se conectar ao seu cluster usando SSL.

Se o Amazon Root CA 1 não estiver instalado em seu sistema operacional, siga as etapas abaixo.

Instalar certificados

Se o certificado `Amazon Root CA 1` não estiver pré-instalado em seu sistema operacional, você precisará instalá-lo manualmente para estabelecer conexões seguras com seu cluster do Aurora DSQL.

Instalação do certificado no Linux

Siga estas etapas para instalar o certificado Amazon Root CA em sistemas Linux.

1. Baixe o certificado raiz:

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

2. Copie o certificado para um armazenamento confiável.

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

3. Atualize o armazenamento confiável da CA:

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

4. Verifique a instalação:

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

Instalação do certificado no macOS

Estas etapas de instalação do certificado são opcionais. A Instalação do certificado no Linux também funciona para macOS.

1. Baixe o certificado raiz:

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

2. Adicione o certificado a System keychain:

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

3. Verifique a instalação:

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

Conectar-se com verificação SSL/TLS

Antes de configurar certificados SSL/TLS para conexões seguras com o cluster do Aurora DSQL, verifique se você atende aos pré-requisitos abaixo.

- PostgreSQL versão 17 instalado
- AWS CLI configurada com as credenciais apropriadas
- Informações do endpoint do cluster do Aurora DSQL

Conectar-se pelo Linux

1. Gere e defina o token de autenticação:

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

2. Conecte-se usando certificados do sistema (se pré-instalados):

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

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

3. Ou conecte-se usando um certificado baixado:

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

> **Note**
>
> Para ter mais informações sobre as configurações do PGSSLMODE, consulte sslmode na documentação Database Connection Control Functions do PostgreSQL 17.

Conectar-se pelo macOS

1. Gere e defina o token de autenticação:

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

2. Conecte-se usando certificados do sistema (se pré-instalados):

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

3. Ou baixe o certificado raiz e salve-o como `root.pem` (se ele não estiver pré-instalado).

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

4. Conectar-se usando o psql:

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

Conectar-se pelo Windows

Por meio do prompt de comando

1. Gere um token de autenticação:

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

2. Defina a variável de ambiente de senha:

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

3. Defina a configuração do SSL:

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

4. Conecte-se ao banco de dados:

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

Usando o PowerShell

1. Gere e defina o token de autenticação:

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

2. Defina a configuração do SSL:

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

3. Conecte-se ao banco de dados:

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

Recursos adicionais

- Documentação do SSL do PostgreSQL
- Amazon Trust Services

Criptografia de dados para o Amazon Aurora DSQL

O Amazon Aurora DSQL criptografa todos os dados do usuário em repouso. Para maior segurança, essa criptografia usa o AWS Key Management Service (AWS KMS). Essa funcionalidade ajuda a reduzir a carga e complexidade operacionais necessárias para proteger dados confidenciais. A criptografia em repouso ajuda você a:

- Reduzir a carga operacional necessária para proteger dados sensíveis.
- Criar aplicações com exigências de segurança que atendam a rigorosos requisitos regulatórios e de conformidade de criptografia.
- Sempre proteger seus dados em um cluster criptografado para adicionar uma camada extra de proteção de dados.
- Cumprir políticas organizacionais, regulamentos do setor ou governamentais e requisitos de conformidade.

Com o Aurora DSQL, é possível criar aplicações com exigências de segurança que atendam a rigorosos requisitos regulatórios e de conformidade de criptografia. As seções a seguir explicam como configurar a criptografia para bancos de dados Aurora DSQL novos e existentes e gerenciar suas chaves de criptografia.

Tópicos

Tipos de chave do KMS para o Aurora DSQL

O Aurora DSQL se integra ao AWS KMS para gerenciar as chaves de criptografia dos seus clusters. Para saber mais sobre tipos e estados de chave, consulte AWS Key Management Service concepts no Guia do desenvolvedor do AWS Key Management Service. Ao criar um cluster, você pode escolher entre os seguintes tipos de chave do KMS para criptografá-lo:

Chave pertencente à AWS

> Tipo de criptografia padrão. A chave pertence ao Aurora DSQL e não há custo adicional para você. O Amazon Aurora DSQL descriptografa de forma transparente os dados do cluster quando você acessa um cluster criptografado. Não é necessário alterar o código ou as aplicações para usar ou gerenciar clusters criptografados, e todas as consultas do Aurora DSQL funcionam com seus dados criptografados.

Chave gerenciada pelo cliente

> Você pode criar, deter e gerenciar a chave em sua Conta da AWS. Você tem controle total sobre a chave do KMS. Há cobranças do AWS KMS para isso.

A criptografia em repouso usando a Chave pertencente à AWS é disponibilizada sem custo adicional. No entanto, as cobranças do AWS KMS se aplicam a chaves gerenciadas pelo cliente. Para obter mais informações, consulte a página de Definição de preço doAWS KMS.

É possível alternar entre esses tipos de chave a qualquer momento. Para ter mais informações sobre tipos de chave, consulte Customer managed keys e Chaves pertencentes à AWS no Guia do desenvolvedor do AWS Key Management Service.

> **Note**
>
> A criptografia em repouso do Aurora DSQL está disponível em todas as regiões da AWS em que o Aurora DSQL está disponível.

Criptografia em repouso no Aurora DSQL

O Amazon Aurora DSQL usa o Advanced Encryption Standard de 256 bits (AES-256) para criptografar seus dados em repouso. Essa criptografia ajuda a proteger seus dados contra acesso não autorizado ao armazenamento subjacente. O AWS KMS gerencia as chaves de criptografia dos clusters. Você pode usar Chaves pertencentes à AWS padrão ou optar por usar Chaves gerenciadas pelo cliente do AWS KMS. Para saber mais sobre como especificar e gerenciar chaves para clusters do Aurora DSQL, consulte Criar um cluster criptografado do Aurora DSQL e Remover ou atualizar uma chave para um cluster do Aurora DSQL.

Tópicos

- Chaves pertencentes à AWS
- Chaves gerenciadas pelo cliente

Chaves pertencentes à AWS

O Aurora DSQL criptografa todos os clusters por padrão com Chaves pertencentes à AWS. Essas chaves são de uso gratuito e são alternadas anualmente para proteger os recursos da sua conta. Você não precisa visualizar, gerenciar, usar ou auditar essas chaves; portanto, nenhuma ação é necessária proteger os dados. Para obter mais informações sobre as Chaves pertencentes à AWS, consulte Chaves pertencentes à AWS no Guia do desenvolvedor do AWS Key Management Service.

Chaves gerenciadas pelo cliente

É possível criar, deter e gerenciar as chaves gerenciadas pelo cliente em sua Conta da AWS. Você tem controle total sobre essas chaves do KMS, inclusive sobre as respectivas políticas, o material de criptografia, as tags e os aliases. Para ter mais informações sobre como gerenciar permissões, consulte Customer managed keys no Guia do desenvolvedor do AWS Key Management Service.

Quando você especifica uma chave gerenciada pelo cliente para criptografia em nível de cluster, o Aurora DSQL criptografa o cluster e todos os respectivos dados regionais com essa chave. Para evitar a perda de dados e manter o acesso ao cluster, o Aurora DSQL precisa acessar sua chave de criptografia. Se você desabilitar a chave gerenciada pelo cliente, programar a chave para exclusão ou tiver uma política que restrinja seu acesso ao serviço, o status de criptografia do cluster será alterado para `KMS_KEY_INACCESSIBLE`. Quando o Aurora DSQL não consegue acessar a chave, os usuários não conseguem se conectar ao cluster, o status da criptografia do cluster muda para `KMS_KEY_INACCESSIBLE` e o serviço perde acesso aos dados do cluster.

Com relação a clusters multirregionais, os clientes podem configurar a chave de criptografia do AWS KMS de cada região separadamente, e cada cluster regional usa sua própria chave de criptografia em nível de cluster. Se o Aurora DSQL não conseguir acessar a chave de criptografia de um cluster emparelhado multirregional, o status desse cluster se tornará `KMS_KEY_INACCESSIBLE` e ele ficará indisponível para operações de leitura e gravação. Outros clusters emparelhados prosseguem com suas operações normais.

> **Note**
>
> Se o Aurora DSQL não conseguir acessar a chave gerenciada pelo cliente, o status de criptografia do cluster mudará para `KMS_KEY_INACCESSIBLE`. Após a restauração do acesso à chave, o serviço detectará automaticamente a restauração em 15 minutos. Para ter mais informações, consulte “Cluster idling”.
> Para clusters multirregionais, se não houver acesso à chave por um longo período, o tempo de restauração do cluster dependerá da quantidade de dados gravados enquanto a chave estava inacessível.

Usar chaves do AWS KMS e de dados com o Aurora DSQL

O recurso de criptografia em repouso do Aurora DSQL usa uma AWS KMS key e uma hierarquia de chaves de dados para proteger os dados do cluster.

Recomendamos que você planeje uma estratégia de criptografia antes de implementar seu cluster no Aurora DSQL. Se você armazenar dados sensíveis ou confidenciais no Aurora DSQL, considere incluir a criptografia do lado do cliente em seu plano. Dessa forma, você poderá criptografar os dados o mais próximo possível de sua origem e garantir sua proteção durante todo o ciclo de vida.

Tópicos

Usar uma AWS KMS key com o Aurora DSQL

A criptografia em repouso protege o cluster do Aurora DSQL usando uma AWS KMS key. Por padrão, o Aurora DSQL usa uma Chave pertencente à AWS, uma chave de criptografia multilocatário que é criada e gerenciada em uma conta de serviço do Aurora DSQL. Mas você pode criptografar os clusters do Aurora DSQL com uma chave gerenciada pelo cliente em sua Conta da AWS. Você pode selecionar uma chave do KMS diferente para cada cluster, mesmo que ele participe de uma configuração multirregional.

Você seleciona a chave do KMS para um cluster ao criar ou atualizar esse cluster. É possível alterar a chave do KMS de um cluster a qualquer momento, seja no console do Aurora DSQL ou usando a operação `UpdateCluster`. O processo de alternar chaves não exige tempo de inatividade nem degradação.

> **Important**
>
> O Aurora DSQL aceita apenas chaves simétricas do KMS. Não é possível usar uma chave assimétrica do KMS para criptografar clusters do Aurora DSQL.

Uma chave gerenciada pelo cliente oferece os benefícios a seguir.

- Você cria e gerencia a chave do KMS, incluindo a configuração de políticas de chave e políticas do IAM para controlar o acesso a ela. Você pode habilitar e desabilitar a chave do KMS, habilitar e desabilitar a alternância automática de chaves e excluir a chave do KMS quando ela não estiver mais em uso.

- Você pode usar uma chave gerenciada pelo cliente com material de chave importado ou uma chave gerenciada pelo cliente em um armazenamento de chaves personalizado que você possui e gerencia.
- Você pode auditar a criptografia e a descriptografia do cluster do Aurora DSQL examinando as chamadas de API do Aurora DSQL para o AWS KMS nos logs do AWS CloudTrail.

No entanto, a Chave pertencente à AWS é gratuita e seu uso não é computado nas cotas de solicitações ou de recursos do AWS KMS. As chaves gerenciadas pelo cliente geram cobrança para cada chamada de API, e as cotas do AWS KMS são aplicáveis a essas chaves.

Usar chaves de cluster com o Aurora DSQL

O Aurora DSQL usa o AWS KMS key para o cluster gerar e criptografar uma chave de dados exclusiva para o próprio cluster, conhecida como chave de cluster.

A chave de cluster é usada como uma chave de criptografia de chaves. O Aurora DSQL usa essa chave de cluster para proteger as chaves de criptografia de dados usadas para criptografar os dados do cluster. O Autora DSQL gera uma chave de criptografia de dados exclusiva para cada estrutura subjacente em um cluster, mas vários itens de cluster podem ser protegidos com a mesma chave de criptografia de dados.

Para descriptografar a chave do cluster, o Aurora DSQL envia uma solicitação ao AWS KMS quando você acessa um cluster criptografado pela primeira vez. Para manter o cluster disponível, o Aurora DSQL verifica periodicamente o acesso de descriptografia à chave do KMS, mesmo quando você não esteja acessando ativamente o cluster.

O Aurora DSQL armazena e usa a chave do cluster e as chaves de criptografia de dados fora do AWS KMS. Ele protege todas as chaves com a criptografia Advanced Encryption Standard (AES) e chaves de criptografia de 256 bits. Em seguida, armazena as chaves criptografadas com os dados criptografados para que estejam disponíveis para descriptografar os dados do cluster sob demanda.

Se você alterar a chave do KMS do cluster, o Aurora DSQL criptografará novamente a chave do cluster existente com a nova chave do KMS.

Armazenamento de chaves de cluster em cache

Para evitar chamar o AWS KMS para cada operação do Aurora DSQL, o Aurora DSQL armazena as chaves de cluster em texto simples para cada chamador na memória. Se o Aurora DSQL receber

uma solicitação para a chave de cluster armazenada em cache após 15 minutos de inatividade, ele enviará uma nova solicitação ao AWS KMS para descriptografar a chave de cluster. Essa chamada capturará todas as alterações feitas nas políticas de acesso da AWS KMS key no AWS KMS ou no AWS Identity and Access Management (IAM) desde a última solicitação para descriptografar a chave de cluster.

Autorizar o uso da AWS KMS key para o Aurora DSQL

Se você usar uma chave gerenciada pelo cliente em sua conta para proteger o cluster do Aurora DSQL, as políticas nessa chave deverão conceder ao Aurora DSQL permissão para usá-la em seu nome.

Você tem controle total sobre as políticas em uma chave gerenciada pelo cliente. O Aurora DSQL não precisa de autorização adicional para usar a Chave pertencente à AWS padrão para proteger os clusters do Aurora DSQL na sua Conta da AWS.

Política de chaves para uma chave gerenciada pelo cliente

Quando uma chave gerenciada pelo cliente é selecionada para proteger um cluster do Aurora DSQL, o Aurora DSQL precisa de permissão para usar a AWS KMS key em nome da entidade principal que faz a seleção. Essa entidade principal, que pode ser um usuário ou um perfil, deve ter as permissões na AWS KMS key que o Aurora DSQL exige. É possível fornecer essas permissões em uma política de chave ou em uma política do IAM.

O Aurora DSQL exige, no mínimo, as seguintes permissões em uma chave gerenciada pelo cliente:

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

Por exemplo, a política de chaves de exemplo a seguir fornece somente as permissões necessárias. A política tem os seguintes efeitos:

- Permite que o Aurora DSQL use a AWS KMS key em operações criptográficas, mas somente quando está atuando em nome de entidades principais na conta que tem permissão para usar

o Aurora DSQL. Se as entidades principais especificadas na declaração de política não tiverem permissão para usar o Aurora DSQL, a chamada falhará, mesmo se vier do serviço do Aurora DSQL.

- A chave de condição `kms:ViaService` concede as permissões somente quando a solicitação vem do Aurora DSQL em nome das entidades principais listadas na declaração de política. Essas entidades principais não podem chamar essas operações diretamente.
- Concede aos administradores da AWS KMS key (usuários que podem assumir o perfil `db-team`) acesso somente leitura à AWS KMS key.

Antes de usar uma política de chaves de exemplo, substitua o exemplo de entidades principais por entidades principais reais da sua conta da Conta da 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 criptografia do Aurora DSQL

Um contexto de criptografia é um conjunto de pares de chave-valor que contêm dados arbitrários não secretos. Quando você inclui um contexto de criptografia em uma solicitação para criptografar dados, o AWS KMS vincula de forma criptográfica o contexto de criptografia aos dados criptografados. Para descriptografar os dados, você deve passar o mesmo contexto de criptografia.

O Aurora DSQL usa o mesmo contexto de criptografia em todas as operações de criptografia do AWS KMS. Se você usar uma chave gerenciada pelo cliente para proteger seu cluster do Aurora DSQL, poderá usar o contexto de criptografia para identificar o uso da AWS KMS key em registros e logs de auditoria. Ele também é exibido em texto simples em logs, como no AWS CloudTrail.

O contexto de criptografia também pode ser usado como uma condição para autorização em políticas.

Nas solicitações ao AWS KMS, o Aurora DSQL usa um contexto de criptografia com dois pares de chave-valor.

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

O par de chave-valor identifica o cluster que o Aurora DSQL está criptografando. A chave é `aws:dsql:ClusterId`. O valor é o identificador do cluster.

Monitorar a interação do Aurora DSQL com o AWS KMS

Se você usa uma chave gerenciada pelo cliente para proteger clusters do Aurora DSQL, é possível usar logs do AWS CloudTrail para rastrear as solicitações que o Aurora DSQL envia ao AWS KMS em seu nome.

Expanda as seções a seguir para saber como o Aurora DSQL usa as operações `GenerateDataKey` e `Decrypt` do AWS KMS.

`GenerateDataKey`

Quando você habilita a criptografia em repouso em um cluster, o Aurora DSQL cria uma chave de cluster exclusiva. Ele envia uma solicitação `GenerateDataKey` ao AWS KMS que especifica a AWS KMS key para o cluster.

O evento que registra a operação `GenerateDataKey` é semelhante ao evento de exemplo a seguir. O usuário é a conta de serviço do Aurora DSQL. Os parâmetros incluem o nome do recurso da Amazon (ARN) da AWS KMS key, um especificador de chave que requer uma chave de 256 bits e o contexto de criptografia que identifica o cluster.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dsql.amazonaws.com"
    },
    "eventTime": "2025-05-16T18:41:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "dsql.amazonaws.com",
    "userAgent": "dsql.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dsql:ClusterId": "w4abucpbwuxx"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
    },
    "responseElements": null,
    "requestID": "2da2dc32-d3f4-4d6c-8a41-aff27cd9a733",
    "eventID": "426df0a6-ba56-3244-9337-438411f826f4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
```

```
            "ARN": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "f88e0dd8-6057-4ce0-b77d-800448426d4e",
    "vpcEndpointId": "AWS Internal",
    "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
    "eventCategory": "Management"
}
```

Decrypt

Quando você acessa um cluster criptografado do Aurora DSQL, o Aurora DSQL precisa descriptografar a chave de cluster para que possa descriptografar as chaves abaixo dela na hierarquia. Em seguida, ele descriptografa os dados no cluster. Para descriptografar a chave do cluster, o Aurora DSQL envia uma solicitação `Decrypt` ao AWS KMS que especifica a AWS KMS key para o cluster.

O evento que registra a operação `Decrypt` é semelhante ao evento de exemplo a seguir. O usuário é a entidade principal na sua Conta da AWS que está acessando o cluster. Os parâmetros incluem a chave de cluster criptografada (como um blob de texto cifrado) e o contexto de criptografia que identifica cluster. O AWS KMS extrai o ID da AWS KMS key do 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"
}
```


Criar um cluster criptografado do Aurora DSQL

Todos os clusters do Aurora DSQL são criptografados em repouso. Por padrão, eles usam uma Chave pertencente à AWS sem custo, ou você pode especificar uma chave personalizada do AWS KMS. Siga estas etapas para criar o cluster criptografado por meio do AWS Management Console ou da AWS CLI.

Console

Como criar um cluster criptografado usando o AWS Management Console

1. Inicie a sessão no Console de Gerenciamento da AWS e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql/.
2. No painel de navegação no lado esquerdo do console, em DAX, selecione Clusters.
3. Escolha Criar cluster no canto superior direito e selecione Região única.

4. Em Configurações de criptografia de cluster, escolha uma das opções a seguir.
   - Aceite as configurações padrão para criptografar com uma Chave pertencente à AWS sem custo adicional.
   - Selecione Personalizar configurações de criptografia (avançado) para especificar uma chave personalizada do KMS. Em seguida, pesquise ou insira o ID ou o alias da chave do KMS. Ou é possível escolher Criar uma chave do AWS KMS para criar uma chave no console do AWS KMS.
5. Selecione Criar cluster.

Para confirmar o tipo de criptografia do cluster, navegue até a página Clusters e selecione o ID do cluster para ver os respectivos detalhes. Analise a guia Configurações de cluster. A configuração Chave do KMS do cluster mostra Chave padrão do Aurora DSQL para clusters que usam chaves pertencentes à AWS ou o ID da chave para outros tipos de criptografia.

> **Note**
>
> Se você optar por deter e gerenciar uma chave própria, defina apropriadamente a política de chave do KMS. Para obter mais informações e exemplos, consulte the section called “Política de chaves para uma chave gerenciada pelo cliente”.

CLI

Como criar um cluster criptografado com a Chave pertencente à AWS padrão

- Use o comando a seguir para criar um cluster do Aurora DSQL.

```
aws dsql create-cluster
```

Conforme mostrado nos detalhes de criptografia a seguir, o status de criptografia do cluster está habilitado por padrão e o tipo de criptografia padrão é chave de propriedade da AWS. O cluster agora está criptografado com a chave padrão pertencente à AWS na conta de serviço Aurora DSQL.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
```

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

Como criar um cluster criptografado com a chave gerenciada pelo cliente

- Use o comando a seguir para criar um cluster do Aurora DSQL, substituindo o ID da chave em texto vermelho pelo ID da chave gerenciada pelo cliente.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Conforme mostrado nos detalhes de criptografia a seguir, o status de criptografia do cluster está habilitado por padrão e o tipo de criptografia é chave do KMS gerenciada pelo cliente. O cluster agora está criptografado com sua chave.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Remover ou atualizar uma chave para um cluster do Aurora DSQL

Você pode usar o AWS Management Console ou a AWS CLI para atualizar ou remover as chaves de criptografia nos clusters existentes no Amazon Aurora DSQL. Se você remover uma chave sem a substituir, o Aurora DSQL usará a Chave pertencente à AWS padrão. Siga estas etapas para atualizar as chaves de criptografia de um cluster existente por meio do console do Aurora DSQL ou da AWS CLI.

Console

Como atualizar ou remover uma chave de criptografia no AWS Management Console

1. Inicie a sessão no Console de Gerenciamento da AWS e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql/.
2. No painel de navegação no lado esquerdo do console, em DAX, selecione Clusters.
3. Na visualização de lista, localize a linha do cluster que você deseja atualizar.

4. Selecione o menu Ações e escolha Modificar.
5. Em Configurações de criptografia de cluster, escolha uma das opções a seguir para modificar as configurações de criptografia.

   - Se você quiser mudar de uma chave personalizada para umaChave pertencente à AWS, desmarque a opção Personalizar configurações de criptografia (avançado). As configurações padrão aplicarão e criptografarão o cluster com uma Chave pertencente à AWS sem custo.
   - Se você quiser alternar de uma chave personalizada do KMS para outra ou de uma Chave pertencente à AWS para uma chave do KMS, selecione a opção Personalizar configurações de criptografia (avançado) se ela ainda não estiver selecionada. Em seguida, pesquise e selecione o ID ou alias da chave que você deseja usar. Ou é possível escolher Criar uma chave do AWS KMS para criar uma chave no console do AWS KMS.
6. Escolha Salvar.

CLI

Os exemplos a seguir mostram como usar a AWS CLI para atualizar o cluster criptografado.

Como atualizar um cluster criptografado com a Chave pertencente à AWS padrão

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

O status `EncryptionStatus` da descrição do cluster é definido como `ENABLED` e o `EncryptionType` é `AWS_OWNED_KMS_KEY`:

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

Esse cluster agora está criptografado com a Chave pertencente à AWS padrão na conta de serviço do Aurora DSQL.

Como atualizar um cluster criptografado com uma chave gerenciada pelo cliente para o Aurora DSQL

Atualize o cluster criptografado, como no seguinte exemplo:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-ab1234a1b234
```

O status `EncryptionStatus` da descrição do cluster é definido como `UPDATING`, e o `EncryptionType` é `CUSTOMER_MANAGED_KMS_KEY`: Depois que o Aurora DSQL terminar de propagar a nova chave pela plataforma, o status da criptografia será mudado para `ENABLED`.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Se você optar por deter e gerenciar uma chave própria, defina apropriadamente a política de chave do KMS. Para obter mais informações e exemplos, consulte the section called “Política de chaves para uma chave gerenciada pelo cliente”.

Considerações sobre criptografia com o Aurora DSQL

- O Aurora DSQL criptografa todos os dados do cluster em repouso. Não é possível desabilitar essa criptografia ou criptografar somente alguns itens em um cluster.
- O AWS Backup criptografa backups e quaisquer clusters restaurados por meio desses backups. É possível criptografar seus dados de backup no AWS Backup usando a chave de propriedade da AWS ou uma chave gerenciada pelo cliente.
- Os seguintes estados de proteção de dados estão habilitados para o Aurora DSQL:

  - Dados em repouso: o Aurora DSQL criptografa todos os dados estáticos na mídia de armazenamento persistente.
  - Dados em trânsito: o Aurora DSQL criptografa todas as comunicações usando Transport Layer Security (TLS) por padrão.
- Ao fazer a transição para uma chave diferente, recomendamos que você mantenha a chave original habilitada até que a transição seja concluída. A AWS precisa da chave original para descriptografar os dados antes de criptografar seus dados com a nova chave. O processo estará concluído quando o `encryptionStatus` do cluster for `ENABLED` e você vir o `kmsKeyArn` da nova chave gerenciada pelo cliente.
- Ao desabilitar a chave gerenciada pelo cliente ou revogar o acesso do Aurora DSQL para usá-la, o cluster entra no estado `IDLE`.
- O AWS Management Console e a API do Amazon Aurora DSQL usam termos diferentes para tipos de criptografia:
  - Console da AWS: no console, você verá `KMS` ao usar uma chave gerenciada pelo cliente e `DEFAULT` ao usar uma Chave pertencente à AWS.
  - API: a API do Amazon Aurora DSQL usa `CUSTOMER_MANAGED_KMS_KEY` para chaves gerenciadas pelo cliente e `AWS_OWNED_KMS_KEY` para Chaves pertencentes à AWS.
- Se você não especificar uma chave de criptografia durante a criação do cluster, o Aurora DSQL criptografará automaticamente seus dados usando a Chave pertencente à AWS.
- É possível alternar entre uma Chave pertencente à AWS e uma chave gerenciada pelo cliente a qualquer momento. Faça essa alteração usando o AWS Management Console, a AWS CLI ou a API do Amazon Aurora DSQL.

Gerenciamento de identidade e acesso para o Aurora DSQL

O AWS Identity and Access Management (IAM) é um serviço da AWS service (Serviço da AWS) que ajuda um administrador a controlar com segurança o acesso aos recursos da AWS. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (ter permissões) para usar os recursos da Aurora DSQL. O IAM é um AWS service (Serviço da AWS) que pode ser usado sem custo adicional.

Tópicos

- Gerenciar o acesso usando políticas
- Como o Amazon Aurora DSQL funciona com o IAM
- Exemplos de política baseada em identidade para o Amazon Aurora DSQL
- Solução de problemas de identidade e acesso do Amazon Aurora DSQL

Público

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que for realizado no Aurora DSQL.

Usuário do serviço: se você usa o serviço Aurora DSQL para fazer seu trabalho, o administrador fornece as credenciais e as permissões necessárias. À medida que usar mais recursos do Aurora DSQL para fazer seu trabalho, você poderá precisar de permissões adicionais. Compreenda como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se você não puder acessar um recurso no Aurora DSQL, consulte Solução de problemas de identidade e acesso do Amazon Aurora DSQL.

Administrador do serviço: se você for o responsável pelos recursos do Aurora DSQL na sua empresa, provavelmente terá acesso total ao Aurora DSQL. Cabe a você determinar quais funcionalidades e recursos do Aurora DSQL os usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com o Aurora DSQL, consulte Como o Amazon Aurora DSQL funciona com o IAM.

Administrador do IAM: se você for um administrador do IAM, é recomendável saber os detalhes sobre como criar políticas para gerenciar o acesso ao Aurora DSQL. Para visualizar exemplos de políticas baseadas em identidade do Aurora DSQL que podem ser usadas no IAM, consulte Exemplos de política baseada em identidade para o Amazon Aurora DSQL.

Autenticação com identidades

A autenticação é a forma como fazer login na AWS usando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como Usuário raiz da conta da AWS, como usuário do IAM, ou assumindo um perfil do IAM.

É possível fazer login na AWS como uma identidade federada usando credenciais fornecidas por uma fonte de identidades. AWS IAM Identity Center (Centro de Identidade do IAM), a autenticação

única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Quando você acessa a AWS usando a federação, está indiretamente assumindo um perfil.

A depender do tipo de usuário, você pode fazer login no AWS Management Console ou no portal de acesso AWS. Para obter mais informações sobre como fazer login na AWS, consulte Como fazer login na contaConta da AWS no Início de Sessão da AWS Guia do usuário.

Se você acessar a AWS de forma programática, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface de linha de comandos (CLI) para você assinar de forma criptográfica as solicitações usando as suas credenciais. Se você não utilizar as ferramentas da AWS, deverá assinar as solicitações por conta própria. Para obter mais informações sobre como usar o método recomendado para designar solicitações por conta própria, consulte Versão 4 do AWS Signature para solicitações de API no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser necessário fornecer informações adicionais de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação multifator no Guia do usuário do AWS IAM Identity Center e Usar a autenticação multifator da AWS no IAM no Guia do usuário do IAM.

Usuário-raiz Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login com acesso completo a todos os Serviços da AWS e recursos na conta. Essa identidade, chamada usuário-raiz da Conta da AWS, é acessada por login com o endereço de e-mail e a senha usada para criar a conta. É altamente recomendável não usar o usuário-raiz para tarefas diárias. Proteja as credenciais do usuário-raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário-raiz, consulte Tarefas que exigem credenciais de usuário-raiz no Guia do Usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários, inclusive os que precisam de acesso de administrador, usem a federação com um provedor de identidades para acessar os Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário de seu diretório de usuários corporativos, um provedor de identidades da web, o AWS Directory Service, o diretório do Centro de Identidade ou qualquer

usuário que acesse os Serviços da AWS usando credenciais fornecidas por meio de uma fonte de identidade. Quando as identidades federadas acessam Contas da AWS, elas assumem perfis que fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, é recomendável usar o AWS IAM Identity Center. É possível criar usuários e grupos no IAM Identity Center ou conectar-se e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todas as suas Contas da AWS e aplicações. Para obter mais informações sobre o Centro de Identidade do IAM, consulte O que é o Centro de Identidade do IAM? no Guia do Usuário do AWS IAM Identity Center.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Sempre que possível, é recomendável contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, é recomendável alternar as chaves de acesso. Para obter mais informações, consulte Alternar as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do Usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e conceder a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Casos de uso para usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade na Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Para assumir temporariamente um perfil do IAM no AWS Management Console, você pode alternar de um usuário para um perfil do IAM (console). É possível presumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre métodos para usar perfis, consulte Métodos para assumir um perfil no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, é possível criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas por ele. Para ter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidade de terceiros (federação) no Guia do usuário do IAM. Se usar o Centro de Identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de Identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Guia do Usuário do AWS IAM Identity Center.
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem anexar uma política diretamente a um recurso (em vez de usar um perfil como proxy). Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.
- Acesso entre serviços: alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões da entidade principal da chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
  - Sessões de acesso direto (FAS): qualquer pessoa que utilizar um perfil ou usuário do IAM para executar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS usa as permissões da entidade principal chamando um AWS service (Serviço da AWS), bem como o AWS service (Serviço da AWS) solicitante, para fazer solicitações para serviços subsequentes. As solicitações de FAS são feitas somente quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de

serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

- Perfil vinculado a serviço: um perfil vinculado a serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode presumir o perfil para executar uma ação em seu nome. Perfis vinculadas ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados a serviço.

- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da API da AWS. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir um perfil da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que esteja anexado a ela. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Utilizar um perfil do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade principal (usuário, usuário-raiz ou sessão de perfil) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas é armazenada na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e perfis não têm permissões. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a

ação `iam:GetRole`. Um usuário com essa política pode obter informações de perfis do AWS Management Console, da AWS CLI ou da API AWS.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e perfis na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, funções, usuários federados ou Serviços da AWS.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Não é possível usar as políticas gerenciadas pela AWS do IAM em uma política baseada em atributos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços que oferecem compatibilidade com ACLs. Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do Desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

A AWS oferece compatibilidade com tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs): SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço que agrupa e gerencia centralmente várias Contas da AWS pertencentes a sua empresa. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. A SCP limita as permissões para entidades em contas de membros, o que inclui cada Usuário raiz da conta da AWS. Para obter mais informações sobre o Organizations e SCPs, consulte Service control policies no Guia do usuário do AWS Organizations.
- Políticas de controle de recursos (RCPs): RCPs são políticas JSON que podem ser usadas para definir o máximo de permissões disponíveis para recursos em suas contas sem atualizar as políticas do IAM anexadas a cada recurso que você possui. A RCP limita as permissões para recursos nas contas-membro e pode afetar as permissões efetivas para identidades, incluindo o Usuário raiz da conta da AWS, independentemente de pertencerem a sua organização. Consulte mais informações sobre o Organizations e as RCPs, incluindo uma lista de Serviços da AWS compatível com RCPs em Resource control policies (RCPs) no Guia do usuário do AWS Organizations.
- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do

usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recursos. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir ou não uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliação de políticas no Guia do usuário do IAM.

Como o Amazon Aurora DSQL funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao Aurora DSQL, saiba quais recursos do IAM estão disponíveis para uso com o Aurora DSQL.

Recursos do IAM que você pode usar com o Amazon Aurora DSQL

Atributo do IAM	Suporte ao Aurora DSQL
Políticas baseadas em identidade	Sim
Políticas baseadas em recurso	Não
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de políticas	Sim
ACLs	Não
ABAC (tags em políticas)	Sim
Credenciais temporárias	Sim
Permissões de entidade principal	Sim
Perfis de serviço	Sim

Atributo do IAM	Suporte ao Aurora DSQL
Perfis vinculados a serviço	Sim

Para ter uma visão geral sobre como o Aurora DSQL e outros serviços da AWS funcionam com a maioria dos recursos do IAM, consulte Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para o Aurora DSQL

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim como as condições sob as quais as ações são permitidas ou negadas. Você não pode especificar a entidade principal em uma política baseada em identidade porque ela se aplica ao usuário ou perfil ao qual ela está anexada. Para saber mais sobre todos os elementos que podem ser usados em uma política JSON, consulte Referência de elemento de política JSON do IAM no Guia do usuário do IAM.

Exemplos de política baseada em identidade para o Aurora DSQL

Para visualizar exemplos de política baseada em identidade do Aurora DSQL, consulte Exemplos de política baseada em identidade para o Amazon Aurora DSQL.

Políticas baseadas em recursos no Aurora DSQL

Compatibilidade com políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico.

Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Para permitir o acesso entre contas, você pode especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recursos. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento da relação de confiança. Quando a entidade principal e o recurso estão em diferentes Contas da AWS, um administrador do IAM da conta confiável também deve conceder à entidade principal (usuário ou perfil) permissão para acessar o recurso. Eles concedem permissão ao anexar uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma política baseada em identidade adicional será necessária. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Ações de política para o Aurora DSQL

Compatível com ações de políticas: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. As ações de políticas geralmente têm o mesmo nome que a operação de API da AWS associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de ações do Aurora DSQL, consulte Actions Defined by Amazon Aurora DSQL na Referência de autorização do serviço.

As ações de política no Aurora DSQL usam o seguinte prefixo antes da ação:

```
dsql
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [
      "dsql:action1",
      "dsql:action2"
]
```

Para visualizar exemplos de política baseada em identidade do Aurora DSQL, consulte Exemplos de política baseada em identidade para o Amazon Aurora DSQL.

Recursos de política para o Aurora DSQL

Compatível com recursos de políticas: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Isso pode ser feito para ações que oferecem compatibilidade com um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem compatibilidade com permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de recurso e os respectivos ARNs, consulte Resources Defined by Amazon Aurora DSQL na Referência de autorização do serviço. Para saber com quais ações você pode especificar o ARN de cada recurso, consulte Actions Defined by Amazon Aurora DSQL.

Para visualizar exemplos de política baseada em identidade do Aurora DSQL, consulte Exemplos de política baseada em identidade para o Amazon Aurora DSQL.

Chaves de condição de política para o Aurora DSQL

Compatível com chaves de condição de política específicas de serviço: sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco `Condition`) permite que você especifique condições nas quais uma instrução estiver em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos de `Condition` em uma declaração ou várias chaves em um único elemento de `Condition`, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição usando uma operação lógica `OR`. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos da política do IAM: variáveis e tags no Guia do usuário do IAM.

A AWS oferece compatibilidade com chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição globais da AWS, consulte Chaves de contexto de condição globais da AWS no Guia do usuário do IAM.

Para ver uma lista de chaves de condição do Aurora DSQL, consulte Condition Keys for Amazon Aurora DSQL na Referência de autorização do serviço. Para saber com quais ações e recursos você pode usar a chave de condição, consulte Actions Defined by Amazon Aurora DSQL.

Para visualizar exemplos de política baseada em identidade do Aurora DSQL, consulte Exemplos de política baseada em identidade para o Amazon Aurora DSQL.

ACLs no Aurora DSQL

Compatível com ACLs: não

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

ABAC com o Aurora DSQL

Compatível com ABAC (tags em políticas): sim

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define as permissões com base em atributos. Na AWS, esses atributos são chamados de tags. É possível anexar tags a entidades do IAM (usuários ou perfis) e a muitos recursos da AWS. Marcar de entidades e atributos é a primeira etapa do ABAC. Em seguida, você cria políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag do recurso que ela estiver tentando acessar.

O ABAC é útil em ambientes que estão crescendo rapidamente e ajuda em situações em que o gerenciamento de políticas se torna um problema.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`.

Se um serviço for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobre o ABAC, consulte Definir permissões com autorização do ABAC no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do usuário do IAM.

Usar credenciais temporárias com o Aurora DSQL

Compatível com credenciais temporárias: sim

Alguns Serviços da AWS não funcionam quando você faz login usando credenciais temporárias. Para obter informações adicionais, como quais Serviços da AWS funcionam com credenciais temporárias, consulte Serviços da AWS que funcionem com o IAM no Guia do usuário do IAM.

Você está usando credenciais temporárias se fizer login no AWS Management Console por qualquer método, exceto nome de usuário e uma senha. Por exemplo, quando você acessa a AWS usando o link de autenticação única (SSO) da sua empresa, esse processo cria credenciais temporárias automaticamente. Você também cria automaticamente credenciais temporárias quando faz login no console como usuário e, em seguida, alterna perfis. Para obter mais informações sobre como alternar funções, consulte Alternar para um perfil do IAM (console) no Guia do usuário do IAM.

É possível criar credenciais temporárias manualmente usando a API AWS CLI ou AWS. Em seguida, você pode usar essas credenciais temporárias para acessar a AWS. A AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para obter mais informações, consulte Credenciais de segurança temporárias no IAM.

Permissões de entidade principal entre serviços para o Aurora DSQL

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

O usuário ou perfil do IAM usado para executar ações na AWS é considerado uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS usa as permissões da entidade principal chamando um AWS service (Serviço da AWS), bem como o AWS service (Serviço da AWS) solicitante, para fazer solicitações para serviços subsequentes. As solicitações de FAS são feitas somente quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.

Perfis de serviço para o Aurora DSQL

Compatível com perfis de serviço: sim

O perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

> **Warning**
>
> Alterar as permissões de um perfil de serviço pode prejudicar a funcionalidade do Aurora DSQL. Edite perfis de serviço somente quando o Aurora DSQL fornecer orientação para fazê-lo.

Perfis vinculados ao serviço para o Aurora DSQL

Compatibilidade com perfis vinculados a serviços: sim

Um perfil vinculado a serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode presumir o perfil de executar uma ação em seu nome. Perfis vinculadas ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados a serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculados a serviço do Autor, consulte Usar perfis vinculados ao serviço no Aurora DSQL.

Exemplos de política baseada em identidade para o Amazon Aurora DSQL

Por padrão, usuários e perfis não têm permissão para criar ou modificar recursos do Aurora DSQL. Eles também não podem executar tarefas usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API da AWS. Para conceder aos usuários permissões para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte Criar políticas do IAM (console) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recurso definidos pelo Aurora DSQL, bem como o formato dos ARNs de cada tipo de recurso, consulte Actions, Resources, and Condition Keys for Amazon Aurora DSQL na Referência de autorização do serviço.

Tópicos

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do Aurora DSQL na sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas gerenciadas pela AWS e avance para as permissões de privilégio mínimo: para começar a conceder permissões a seus usuários e workloads, use as políticas gerenciadas pela AWS, que concedem permissões para muitos casos de uso comuns. Elas estão disponíveis em seus Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da AWS que são específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também

conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.

- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Também pode usar condições para conceder acesso a ações de serviço, se elas forem usadas por meio de um AWS service (Serviço da AWS) específico, como o AWS CloudFormation. Para obter mais informações, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA): se houver um cenário que exija usuários do IAM ou um usuário raiz em sua Conta da AWS, ative a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Usar o console do Aurora DSQL

Para acessar o console do Amazon Aurora DSQL, você deve ter um conjunto mínimo de permissões. Essas permissões devem autorizar você a listar e visualizar detalhes sobre os recursos do Aurora DSQL na sua Conta da AWS. Caso crie uma política baseada em identidade mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Não é necessário conceder permissões mínimas do console para usuários que fazem chamadas somente à AWS CLI ou à API do AWS. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que os usuários e perfis ainda possam usar o console do Aurora DSQL, anexe também a política `AmazonAuroraDSQLConsoleFullAccess` ou `AmazonAuroraDSQLReadOnlyAccess` gerenciada pelaAWS às entidades. Para obter informações, consulte Adicionar permissões a um usuário no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a API da 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": "*"
        }
```

```
    ]
}
```

Solução de problemas de identidade e acesso do Amazon Aurora DSQL

Use as informações a seguir para ajudar a diagnosticar e corrigir problemas comuns encontrados ao trabalhar com o Aurora DSQL e o IAM.

Tópicos

- Não tenho autorização para realizar uma ação no Aurora DSQL
- Não estou autorizado a executar iam:PassRole
- Quero permitir que pessoas fora da minha Conta da AWS acessem os recursos do meu Aurora DSQL

Não tenho autorização para realizar uma ação no Aurora DSQL

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando `mateojackson` tenta usar o console para visualizar detalhes sobre o recurso *`my-dsql-cluster`* e não tem as permissões *`GetCluster`*.

```
User: iam::user/mateojackson is not authorized to perform: GetCluster on resource: my-
dsql-cluster
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-dsql-cluster`* usando a ação *`GetCluster`*.

Se você precisar de ajuda, entre em contato com seu administrador . Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que não tem autorização para executar a ação `iam:PassRole`, suas políticas deverão ser atualizadas para permitir a passagem de um perfil para o Aurora DSQL.

Alguns Serviços da AWS permitem que você passe um perfil existente para o serviço, em vez de criar um perfil de serviço ou perfil vinculado ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O erro do exemplo a seguir ocorre quando um usuário do IAM chamado `marymajor` tenta usar o console para executar uma ação no Aurora DSQL. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem os recursos do meu Aurora DSQL

Você pode criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem compatibilidade com políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o Aurora DSQL comporta esses recursos, consulte Como o Amazon Aurora DSQL funciona com o IAM.
- Para saber como conceder acesso a seus recursos em todas as Contas da AWS pertencentes a você, consulte Fornecimento de acesso a um usuário do IAM em outra Conta da AWS pertencente a você no Guia de usuário do IAM.
- Para saber como conceder acesso a seus recursos para Contas da AWS de terceiros, consulte Fornecimento de acesso a Contas da AWS pertencentes a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.

- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Usar perfis vinculados ao serviço no Aurora DSQL

O Aurora DSQL usa perfis vinculados ao serviço do AWS Identity and Access Management (IAM). Um perfil vinculado ao serviço é um tipo exclusivo de perfil do IAM diretamente vinculado ao Aurora DSQL. Os perfis vinculados ao serviço são predefinidos pelo Aurora DSQL e incluem todas as permissões exigidas para chamar Serviços da AWS em nome do cluster do Aurora DSQL.

Os perfis vinculados ao serviço facilitam a configuração do Aurora DSQL porque você não precisa adicionar manualmente as permissões necessárias para usar o Aurora DSQL. Quando você cria um cluster, o Aurora DSQL cria um perfil vinculado ao serviço para você. Você pode excluir o perfil vinculado ao serviço somente depois de excluir todos os seus clusters. Isso protege os recursos do Aurora DSQL porque não é possível remover inadvertidamente as permissões necessárias para acessar os recursos.

Para obter informações sobre outros produtos que são compatíveis com as funções vinculadas a serviços, consulte Serviços da AWS compatíveis com o IAM e procure os serviços que contêm Yes (Sim) na coluna Service-Linked Role (Função vinculada a serviço). Escolha um Sim com um link para visualizar a documentação do perfil vinculado para esse serviço.

Os perfis vinculados ao serviço estão disponíveis em todas as regiões compatíveis do Aurora DSQL.

Permissões de perfis vinculados ao serviço para o Aurora DSQL

O Aurora DSQL usa o perfil vinculado ao serviço denominado `AWSServiceRoleForAuroraDsql`, que permite que o Amazon Aurora DSQL crie e gerencie recursos da AWS em seu nome. Esse perfil vinculado ao serviço é anexado à seguinte política gerenciada: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua um perfil vinculado a serviço. Você pode se deparar com a seguinte mensagem de erro: `You don't have the permissions to create an Amazon Aurora DSQL service-linked role`. Se essa mensagem for exibida, verifique se você tem as seguintes permissões habilitadas:
>
> ```
> {
> ```

```
"Sid" : "CreateDsqlServiceLinkedRole",
    "Effect" : "Allow",
    "Action" : "iam:CreateServiceLinkedRole",
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "iam:AWSServiceName" : "dsql.amazonaws.com"
      }
    }
  }
```

Para ter mais informações, consulte Permissões de perfis vinculados ao serviço.

Criar um perfil vinculado ao serviço

Você não precisa criar manualmente o perfil vinculado ao serviço AuroraDSQLServiceLinkedRolePolicy. O Aurora DSQL cria o perfil vinculado ao serviço para você. Se o perfil vinculado ao serviço AuroraDSQLServiceLinkedRolePolicy tiver sido excluída da conta, o Aurora DSQL criará o perfil quando você iniciar um novo cluster do Aurora DSQL.

Editar um perfil vinculado ao serviço

O Aurora DSQL não permite que você edite o perfil vinculado ao serviço AuroraDSQLServiceLinkedRolePolicy. Depois que você criar um perfil vinculado ao serviço, não poderá alterar o nome do perfil, pois várias entidades podem fazer referência ao perfil. No entanto, você pode editar a descrição da função usando o console do IAM, a AWS Command Line Interface (AWS CLI), ou a API do IAM.

Excluir um perfil vinculado ao serviço

Se você não precisar mais usar um recurso ou serviço que requer um perfil vinculado ao serviço, é recomendável excluí-lo. Dessa forma, você não terá uma entidade não utilizada que não seja ativamente monitorada ou mantida.

Para que você possa excluir um perfil vinculado ao serviço referente a uma conta, será necessário desligar e excluir todos os clusters da conta.

Também é possível usar o console do IAM, a AWS CLI ou a API do IAM para excluir uma função vinculada ao serviço. Para ter mais informações, consulte Criar um perfil vinculado ao serviço no "Guia do usuário do IAM".

Regiões compatíveis com perfis vinculados ao serviço do Aurora DSQL

O Aurora DSQL permite usar perfis vinculados ao serviço em todas as regiões em que o serviço está disponível. Para obter mais informações, consulte Regiões e endpoints da AWS.

Usar chaves de condição do IAM com o Amazon Aurora DSQL

Ao conceder permissões no Aurora DSQL, você pode especificar as condições que determinam como uma política de permissões entra em vigor. Os exemplos a seguir mostram como você pode usar chaves de condição em políticas de permissões do IAM do Aurora DSQL.

Exemplo 1: conceder permissão para criar um cluster em uma Região da AWS específica

A política a seguir concede permissão para criar clusters nas regiões Leste dos EUA (Norte da Virgínia) e Leste dos EUA (Ohio). Essa política usa o ARN do recurso para limitar as regiões permitidas; portanto, o Aurora DSQL só pode criar clusters se esse ARN for especificado na seção `Resource` da política.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            # Control where clusters can be created
            "Action": ["CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Exemplo 2: conceder permissão para criar um cluster multirregional em uma Região da AWS específica

A política a seguir concede permissão para criar clusters multirregionais nas regiões Leste dos EUA (Norte da Virgínia) e Leste dos EUA (Ohio). Essa política usa o ARN do recurso para limitar as regiões permitidas; portanto, o Aurora DSQL pode criar clusters multirregionais se esse ARN for especificado na seção `Resource` da política. Observe que a criação de clusters multirregionais também exige as permissões `PutMultiRegionProperties`, `PutWitnessRegion` e `AddPeerCluster` em cada região especificada.

```
{
    "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/*"
        ]
      }
    ]
}
```

Exemplo 3: conceder permissão para criar um cluster multirregional com uma região testemunha específica

A política a seguir usa uma chave de condição `dsql:WitnessRegion` do Aurora DSQL e permite que um usuário crie clusters multirregionais com uma região testemunha no Oeste dos EUA (Oregon). Se você não especificar a condição `dsql:WitnessRegion`, poderá usar qualquer região como região testemunha.

```
{
    "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"
                     ]
                }

            }
          }
      ]
}
```

Resposta a incidentes no Amazon Aurora DSQL

A segurança é a maior prioridade na AWS. Como parte do AWSModelo de Responsabilidade Compartilhada da Nuvem AWS, a gerencia uma arquitetura de data center, rede e software que atende às exigências das organizações com os maiores requisitos de segurança. A AWS é responsável por qualquer resposta a incidentes relacionada ao serviço Amazon Aurora DSQL em si. Além disso, como cliente da AWS, você compartilha a responsabilidade de manter a segurança na nuvem. Isso significa que você controla a segurança que escolhe implementar com base nas ferramentas e recursos da AWS aos quais tem acesso. Além disso, você é responsável pela resposta a incidentes do seu lado do Modelo de Responsabilidade Compartilhada.

Ao estabelecer uma referência de segurança que atenda aos objetivos de suas aplicações executadas na nuvem, você pode detectar desvios aos quais pode reagir. Para ajudar você

a compreender o impacto que a resposta a incidentes e suas escolhas têm em suas metas corporativas, é recomendável analisar os seguintes recursos:

- Guia de resposta a incidentes de segurança da AWS
- Práticas recomendadas de segurança, identidade e conformidade da AWS
- Perspectiva de segurança no whitepaper AWS Cloud Adoption Framework (CAF)

O Amazon GuardDuty é um serviço gerenciado de detecção de ameaças que monitora continuamente comportamentos mal-intencionados ou não autorizados para ajudar os clientes a proteger Contas da AWS e workloads e identificar possíveis atividades suspeitas antes que elas se transformem em um incidente. Ele monitora atividades, como chamadas incomuns de API ou implantações potencialmente não autorizadas, indicando possível comprometimento ou reconhecimento de contas ou recursos por agentes mal-intencionados. Por exemplo, o Amazon GuardDuty é capaz de detectar atividades suspeitas em APIs do Amazon Aurora DSQL, como um usuário que está fazendo login de um novo local e criando um cluster.

Validação de conformidade para o Amazon Aurora DSQL

Para saber se um AWS service (Serviço da AWS) está no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo por programa de conformidade e selecione o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de Conformidade da AWS.

É possível baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Serviços da AWS é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelos regulamentos e leis aplicáveis. A AWS fornece os recursos a seguir para ajudar com a conformidade:

- Governança e conformidade de segurança: esses guias de implementação de solução abordam considerações sobre a arquitetura e fornecem etapas para implantar recursos de segurança e conformidade.
- Referência de serviços qualificados para HIPAA: lista os serviços qualificados para HIPAA. Nem todos os Serviços da AWS estão qualificados pela HIPAA.
- Recursos de Conformidade da AWS: essa coleção de manuais e guias pode ser aplicada ao seu setor e local.

- Guias de conformidade do cliente da AWS: entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as práticas recomendadas para proteção de Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- Avaliar recursos com regras no Guia do desenvolvedor do AWS Config: o serviço AWS Config avalia como as configurações de recursos estão em conformidade com práticas internas, diretrizes do setor e regulamentos.
- AWS Security Hub: este AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança na AWS. O Security Hub usa controles de segurança para avaliar os recursos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços e controles aceitos, consulte a Referência de controles do Security Hub.
- Amazon GuardDuty: este AWS service (Serviço da AWS) detecta possíveis ameaças às suas Contas da AWS, workloads, contêineres e dados ao monitorar o ambiente em busca de atividades suspeitas e maliciosas. O GuardDuty pode ajudar você a atender a diversos requisitos de conformidade, como o PCI DSS, com o cumprimento dos requisitos de detecção de intrusões requeridos por determinadas estruturas de conformidade.
- AWS Audit Manager: este AWS service (Serviço da AWS) ajuda você a auditar continuamente o seu uso da AWS para simplificar o modo como você gerencia os riscos e a conformidade com regulamentos e padrões do setor.

Resiliência no Amazon Aurora DSQL

A infraestrutura global da AWS se baseia em Regiões da AWS e zonas de disponibilidade (AZs). A Regiões da AWS oferece várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, alto throughput e em redes altamente redundantes. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais. O Aurora DSQL foi projetado para que você possa aproveitar a infraestrutura regional da AWS e, ao mesmo tempo, oferecer a mais alta disponibilidade do banco de dados. Por padrão, os clusters de região única no Aurora DSQL têm disponibilidade multi-AZ, oferecendo tolerância a grandes falhas de componentes e interrupções na infraestrutura que podem afetar o acesso a uma

AZ completa. Os clusters multirregionais oferecem todos os benefícios da resiliência multi-AZ e, ao mesmo tempo, fornecem uma disponibilidade de banco de dados altamente consistente, até nos casos em que a Região da AWS está inacessível aos clientes da aplicação.

Para obter mais informações sobre Regiões da AWS e zonas de disponibilidade, consulte Infraestrutura global da AWS.

Além da infraestrutura global da AWS, o Aurora DSQL oferece vários recursos para atender às suas necessidades de resiliência de dados e backup.

Backup e restauração

O Aurora DSQL permite backup e restauração com o Console do AWS Backup. Você pode realizar backup e restauração completos de seus clusters de região única e multirregionais. Para obter mais informações, consulte Backup e restauração para o Amazon Aurora DSQL.

Replicação

O Aurora DSQL foi projetado para confirmar todas as transações de gravação em um log de transações distribuídas e replicar de forma síncrona todos os dados de log confirmados em réplicas de armazenamento do usuário em três AZs. Os clusters multirregionais oferecem recursos completos de replicação entre regiões e entre regiões de leitura e gravação.

Uma região testemunha designada permite gravações somente no log de transações e não consome armazenamento. As regiões testemunha não têm um endpoint. Isso significa que elas armazenam somente logs de transações criptografados, não exigem administração nem configuração e não são acessíveis aos usuários.

Os logs de transações e o armazenamento do usuário do Aurora DSQL são distribuídos com todos os dados apresentados aos processadores de consulta do Aurora DSQL como um único volume lógico. O Aurora DSQL divide, mescla e replica automaticamente os dados com base no intervalo de chaves primárias e nos padrões de acesso do banco de dados. O Aurora DSQL aumenta e reduz a escala verticalmente das réplicas de leitura, de maneira automática, com base na frequência de acesso de leitura.

As réplicas de armazenamento do cluster são distribuídas em uma frota de armazenamento multilocatário. Se um componente ou uma AZ sofrer danos, o Aurora DSQL redirecionará automaticamente o acesso aos componentes que permanecerem funcionais e reparará de forma assíncrona as réplicas ausentes. Depois que o Aurora DSQL corrige as réplicas danificadas, ele as adiciona automaticamente ao quórum de armazenamento e as disponibiliza para seu cluster.

Alta disponibilidade

Por padrão, os clusters de região única e multirregionais no Aurora DSQL são ativos-ativos, e você não precisa provisionar, configurar ou reconfigurar manualmente nenhum cluster. O Aurora DSQL automatiza totalmente a recuperação de clusters, o que elimina a necessidade de operações tradicionais de failover primário-secundário. Como a replicação é sempre síncrona e feita em várias AZs, não há risco de perda de dados devido ao atraso na replicação ou ao failover para um banco de dados secundário assíncrono durante a recuperação de falhas.

Os clusters de região única fornecem um endpoint redundante multi-AZ que habilita automaticamente o acesso simultâneo com alta consistência de dados em três AZs. Isso significa que as réplicas de armazenamento do usuário em qualquer uma dessas três AZs sempre exibem o mesmo resultado para um ou mais leitores e estão sempre disponíveis para receber gravações. Essa alta consistência e resiliência multi-AZ estão disponíveis em todas as regiões para clusters multirregionais do Aurora DSQL. Isso significa que os clusters multirregionais fornecem dois endpoints regionais altamente consistentes, para que os clientes possam ler ou gravar indiscriminadamente em qualquer uma das regiões sem atraso de replicação na confirmação.

O Aurora DSQL fornece disponibilidade de 99,99% para clusters de uma única região e 99,999% para clusters multirregionais.

Segurança da infraestrutura no Amazon Aurora DSQL

Como um serviço gerenciado, o Amazon Aurora DSQL é protegido pelos procedimentos de segurança de rede global descritos em Best Practices for Security, Identity, and Compliance.

Você usa chamadas de API publicadas pela AWS para acessar o Aurora DSQL por meio da rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.2 ou posterior. Os clientes também devem ter suporte a conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Gerenciar e conectar-se com clusters do Amazon Aurora DSQL usando o AWS PrivateLink

Com o AWS PrivateLink para Amazon Aurora DSQL, é possível provisionar endpoints da Amazon VPC de interface (endpoints de interface) em sua Amazon Virtual Private Cloud. Esses endpoints podem ser acessados diretamente por meio da Amazon VPC e do AWS Direct Connect pelas aplicações que estão no ambiente on-premises ou por emparelhamento da Amazon VPC pelas aplicações que estão em uma Região da AWS diferente. Usando endpoints de interface e o AWS PrivateLink, é possível simplificar a conectividade de rede privada das aplicações com o Aurora DSQL.

As aplicações dentro da Amazon VPC podem acessar o Aurora DSQL usando endpoints de interface da Amazon VPC sem exigir endereços IP públicos.

Os endpoints de interface são representados por uma ou mais interfaces de rede elástica (ENIs) que recebem endereços IP privados de sub-redes na Amazon VPC. As solicitações ao Aurora DSQL por meio de endpoints de interface permanecem na AWS. Para ter mais informações sobre como conectar a Amazon VPC à rede on-premises, consulte o Guia do usuário do AWS Direct Connect e o Guia do usuário do AWS Site-to-Site VPN.

Para ter mais informações sobre como criar endpoints de interface, consulte Access an AWS service using an interface Amazon VPC endpoint no Guia do usuário do AWS PrivateLink.

Tipos de endpoint da Amazon VPC para o Aurora DSQL

O Aurora DSQL exige dois tipos diferentes de endpoint do AWS PrivateLink.

1. Endpoint de gerenciamento: esse endpoint é usado para operações administrativas, como `get`, `create`, `update`, `delete` e `list`, em clusters do Aurora DSQL. Consulte Gerenciar clusters do Aurora DSQL usando o AWS PrivateLink.
2. Endpoint de conexão: esse endpoint é usado para estabelecer conexão com clusters do Aurora DSQL por meio de clientes PostgreSQL. Consulte Conectar-se com clusters do Aurora DSQL usando o AWS PrivateLink.

Considerações ao usar o AWS PrivateLink para o Aurora DSQL

As considerações sobre a Amazon VPC se aplicam ao AWS PrivateLink para o Aurora DSQL. Para ter mais informações, consulte Access an AWS service using an interface VPC endpoint e AWS PrivateLink quotas no “Guia do AWS PrivateLink”.

Gerenciar clusters do Aurora DSQL usando o AWS PrivateLink

Você pode usar a AWS Command Line Interface ou os kits de desenvolvimento de software (SDKs) da AWS para gerenciar clusters do Aurora DSQL por meio de endpoints de interface do Aurora DSQL.

Criar um Amazon VPC endpoint

Para criar um endpoint de interface da Amazon VPC, consulte Create an Amazon VPC endpoint no "Guia do 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 usar o nome de DNS regional padrão para solicitações de API do Aurora DSQL, não desabilite o DNS privado ao criar o endpoint de interface do Aurora DSQL. Quando o DNS privado estiver habilitado, as solicitações ao serviço Aurora DSQL dentro da Amazon VPC serão automaticamente resolvidas para o endereço IP privado do endpoint da Amazon VPC, em vez do nome de DNS público. Quando o DNS privado estiver habilitado, as solicitações do Aurora DSQL feitas na Amazon VPC serão automaticamente resolvidas para seu endpoint da Amazon VPC.

Se o DNS privado não estiver habilitado, use os parâmetros `--region` e `--endpoint-url` com comandos da AWS CLI para gerenciar clusters do Aurora DSQL por meio dos endpoints de interface do Aurora DSQL.

Listar clusters usando um URL de endpoint

No exemplo a seguir, substitua a Região da AWS `us-east-1` e o nome de DNS do ID `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` do endpoint da VPC por suas próprias informações.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-
east-1.vpce.amazonaws.com list-clusters
```

Operações de API

Consulte a Referência de API do Aurora DSQL para ver a documentação sobre gerenciamento de recursos no Aurora DSQL.

Gerenciar políticas de endpoint

Ao testar e configurar minuciosamente as políticas de endpoint da Amazon VPC, você pode ajudar a garantir que o cluster do Aurora DSQL seja seguro, compatível e alinhado com os requisitos específicos de controle de acesso e governança da sua organização.

Exemplo: política de acesso completo ao Aurora DSQL

A política a seguir concede acesso completo a todas as ações e recursos do Aurora DSQL por meio do endpoint especificado da Amazon VPC.

```
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": "*"
        }
      ]
    }'
```

Exemplo: política de acesso restrito ao Aurora DSQL

A política a seguir só permite estas ações do Aurora DSQL.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Todas as outras ações do Aurora DSQL são negadas.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Conectar-se com clusters do Aurora DSQL usando o AWS PrivateLink

Depois que o endpoint do AWS PrivateLink estiver configurado e ativo, você poderá se conectar ao cluster do Aurora DSQL usando um cliente PostgreSQL. As instruções de conexão abaixo descrevem as etapas para criar o nome de host adequado para conexão por meio do endpoint do AWS PrivateLink.

Configurar um endpoint de conexão do AWS PrivateLink

Etapa 1: obter o nome do serviço para o cluster

Ao criar um endpoint do AWS PrivateLink para se conectar ao cluster, primeiro é necessário buscar o nome do serviço específico do cluster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Exemplo de resposta

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

O nome do serviço inclui um identificador, como `dsql-fnh4` no exemplo. Esse identificador também é necessário ao criar o nome do host para se conectar ao cluster.

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);
```

Etapa 2: criar o endpoint da Amazon VPC

Usando o nome do serviço obtido na etapa anterior, crie um endpoint da Amazon VPC.

> **Important**
>
> As instruções de conexão abaixo só funcionam para conexão com clusters quando o DNS privado está habilitado. Não use o sinalizador `--no-private-dns-enabled` ao criar o endpoint, pois isso impedirá que as instruções de conexão abaixo funcionem corretamente. Se você desabilitar o DNS privado, precisará criar seu próprio registro de DNS privado curinga que aponte para o endpoint criado.

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
```

Exemplo de resposta

```
{
    "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

Use um URL de endpoint para as APIs do 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);
```

Conectar-se a um cluster do Aurora DSQL usando um endpoint de conexão do AWS PrivateLink

Depois que o endpoint do AWS PrivateLink estiver configurado e ativo (verifique se `State` está `available`), você pode se conectar ao cluster do Aurora DSQL usando um cliente PostgreSQL. Para ter instruções sobre como usar os SDKs da AWS, siga os guias em Programming with Aurora DSQL. Você deve alterar o endpoint do cluster para que corresponda ao formato do nome do host.

Criar o nome do host

O nome do host para conexão por meio do AWS PrivateLink é diferente do nome do host de DNS público. Você precisa criá-lo usando os componentes a seguir.

1. `Your-cluster-id`
2. O identificador do serviço do nome do serviço. Por exemplo: `dsql-fnh4`
3. A Região da AWS

Use o seguinte formato: *cluster-id*.*service-identifier*.*region*.on.aws.

Exemplo: conectar-se usando o 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
```

Solução de problemas com o AWS PrivateLink

Problemas e soluções comuns

A tabela a seguir lista problemas e soluções comuns relacionados ao AWS PrivateLink com o Aurora DSQL.

Problema	Possível causa	Solução
Tempo limite da conexão	Grupo de segurança não configurado corretamente	Use o Amazon VPC Reachability Analyzer para garantir que sua configuração de rede permita tráfego na porta 5432.
Falha na resolução de DNS	DNS privado não habilitado	Verifique se o endpoint da Amazon VPC foi criado com o DNS privado habilitado.
Falha na autenticação	Credenciais incorretas ou token expirado	Gere um novo token de autenticação e verifique o nome de usuário.
Nome do serviço não encontrado	ID do cluster incorreto	Verifique novamente o ID do cluster e a Região da AWS ao buscar o nome do serviço.

Recursos relacionados

Para obter mais informações, consulte os seguintes recursos:

- Guia do usuário do Amazon Aurora DSQL
- Documentação do AWS PrivateLink
- Access AWS services through AWS PrivateLink

Análise de configuração e vulnerabilidade no Amazon Aurora DSQL

A AWS se encarrega das tarefas básicas de segurança, como aplicação de patches a bancos de dados e sistemas operacionais (SOs) convidados, configuração de firewalls e recuperação de desastres. Esses procedimentos foram revisados e certificados por terceiros certificados. Para obter mais detalhes, consulte os seguintes recursos da :

- Modelo de responsabilidade compartilhada
- Amazon Web Services: visão geral dos processos de segurança (whitepaper)

Prevenção contra o ataque do “substituto confuso” em todos os serviços

“Confused deputy” é um problema de segurança no qual uma entidade sem permissão para executar uma ação pode coagir uma entidade mais privilegiada a executá-la. Na AWS, a personificação entre serviços pode resultar no problema do ‘confused deputy’. A personificação entre serviços pode ocorrer quando um serviço (o serviço de chamada) chama outro serviço (o serviço chamado). O serviço de chamada pode ser manipulado de modo a usar suas permissões para atuar nos recursos de outro cliente de uma forma na qual ele não deveria ter permissão para acessar. Para evitar isso, a AWS fornece ferramentas que ajudam você a proteger seus dados para todos os serviços com entidades principais de serviço que receberam acesso aos recursos em sua conta.

Recomendamos o uso das chaves de contexto de condição globais `aws:SourceArn` e `aws:SourceAccount` em políticas de recursos para limitar as permissões que o Amazon Aurora DSQL concede ao recurso para outro serviço. Use `aws:SourceArn` se quiser que apenas um recurso seja associado ao acesso entre serviços. Use `aws:SourceAccount` se quiser permitir que qualquer recurso nessa conta seja associado ao uso entre serviços.

A maneira mais eficaz de se proteger contra o problema do substituto confuso é usar a chave de contexto de condição global `aws:SourceArn` com o ARN completo do recurso. Se você não souber o ARN completo do recurso ou especificar vários recursos, use a chave de condição de contexto global `aws:SourceArn` com caracteres curinga (*) para as partes desconhecidas do ARN. Por exemplo, .`arn:aws:`*servicename*`:*:`*123456789012*`:*`

Se o valor de `aws:SourceArn` não contiver o ID da conta, como um ARN de bucket do Amazon S3, você deverá usar ambas as chaves de contexto de condição global para limitar as permissões.

O valor de `aws:SourceArn` deve ser ResourcedEscription.

O exemplo a seguir mostra como você pode usar as chaves de contexto de condição globais `aws:SourceArn` e `aws:SourceAccount` no Aurora DSQL para evitar o problema de representante confuso.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Práticas recomendadas de segurança para o Aurora DSQL

O Aurora DSQL oferece uma série de recursos de segurança a serem considerados no desenvolvimento e na implementação das suas próprias políticas de segurança. As práticas recomendadas a seguir são diretrizes gerais e não representam uma solução completa de segurança. Como essas práticas recomendadas podem não ser adequadas ou suficientes para o seu ambiente, trate-as como considerações úteis em vez de prescrições.

Tópicos

Práticas recomendadas de segurança de detecção para o Amazon Aurora DSQL

Além das formas de usar o Aurora DSQL com segurança apresentadas a seguir, consulte Segurança em AWS Well-Architected Tool para saber como as tecnologias de nuvem melhoram a segurança.

Alarmes do Amazon CloudWatch

Com o uso de alarmes do Amazon CloudWatch, você observa uma única métrica durante um período especificado. Se a métrica exceder determinado limite, uma notificação será enviada para um tópico do Amazon SNS ou para uma política do AWS Auto Scaling. Os alarmes do CloudWatch não invocam ações só porque estão em um determinado estado. O estado deve ter sido alterado e mantido por uma quantidade especificada de períodos.

Marcar recursos do Aurora DSQL para identificação e automação

Você pode atribuir metadados aos seus recursos da AWS na forma de tags. Cada tag é um rótulo simples que consiste em uma chave definida pelo cliente e um valor opcional que pode facilitar o gerenciamento, a pesquisa e a filtragem de recursos.

A atribuição de tags (tagging) permite a implementação de controles agrupados. Embora não haja tipos de tags inerentes, elas permitem categorizar recursos por finalidade, proprietário, ambiente ou outros critérios. Veja os seguintes exemplos:

- Segurança: usada para determinar requisitos como criptografia.
- Confidencialidade: um identificador do nível de confidencialidade de dados específico permitido por um recurso.

- Ambiente: usada para distinguir entre as infraestruturas de desenvolvimento, teste e produção.

Você pode atribuir metadados aos seus recursos da AWS na forma de tags. Cada tag é um rótulo simples que consiste em uma chave definida pelo cliente e um valor opcional que pode facilitar o gerenciamento, a pesquisa e a filtragem de recursos.

A atribuição de tags (tagging) permite a implementação de controles agrupados. Embora não haja tipos de tags inerentes, elas permitem categorizar recursos do por finalidade, proprietário, ambiente ou outros critérios. Veja os seguintes exemplos.

- Segurança: usada para determinar requisitos como criptografia.
- Confidencialidade: um identificador do nível de confidencialidade de dados específico permitido por um recurso.
- Ambiente: usada para distinguir entre as infraestruturas de desenvolvimento, teste e produção.

Para ter mais informações, consulte Best Practices for Tagging AWSResources.

Práticas recomendadas de segurança preventiva para o Aurora DSQL

Além das formas de usar o Aurora DSQL com segurança apresentadas a seguir, consulte Segurança em AWS Well-Architected Tool para saber como as tecnologias de nuvem melhoram a segurança.

Use perfis do IAM para autenticar o acesso ao Aurora DSQL.

Usuários, aplicações e outros Serviços da AWS que acessam o Aurora DSQL devem incluir credenciais válidas da AWS nas solicitações da API da AWS e da AWS CLI. Você não deve armazenar credenciais da AWS diretamente na aplicação ou em instâncias do EC2. Essas são credenciais de longo prazo que não são alternadas automaticamente. Haverá um impacto significativo na empresa se essas credenciais forem comprometidas. Um perfil do IAM permite obter chaves de acesso temporárias que podem acessar recursos e Serviços da AWS.

Para obter mais informações, consulte Autenticação e autorização para o Aurora DSQL.

Use políticas do IAM para autorização básica do Aurora DSQL.

Ao conceder permissões, você decide quem as recebe, a quais APIs do Auroras DSQL as permissões se referem e as ações específicas que deseja permitir nesses recursos. A implementação do privilégio mínimo é fundamental para reduzir o risco de segurança e o impacto que pode resultar de erros ou usuários mal-intencionados.

Anexe políticas de permissões a perfis do IAM e conceda permissões para executar operações em recursos do Aurora DSQL. Também estão disponíveis limites de permissões para entidades do IAM, que possibilitam definir as permissões máximas que uma política baseada em identidade pode conceder a uma entidade do IAM.

Assim como as práticas recomendadas de usuário-raiz para sua Conta da AWS, não use O perfil `admin` no Aurora DSQL para realizar operações diárias. Em vez disso, recomendamos que você crie perfis de banco de dados personalizados para gerenciar e se conectar ao cluster. Para ter mais informações, consulte Accessing Aurora DSQL e Understanding authentication and authorization for Aurora DSQL.

Use **`verify-full`** em ambientes de produção.

Essa configuração verifica se o certificado do servidor está assinado por uma autoridade de certificação confiável e se o nome do host do servidor corresponde ao certificado.

Atualize seu cliente PostgreSQL.

Atualize regularmente seu cliente PostgreSQL para a versão mais recente a fim de se beneficiar das melhorias de segurança. Recomendamos usar o PostgreSQL versão 17.

Marcação de recursos no Aurora DSQL

Na AWS, as tags são pares de chave-valor definidos pelo usuário e que são definidos e associados aos recursos do Aurora DSQL, como clusters. As tags são opcionais. Se você fornecer uma chave, o valor será opcional.

Você pode usar o AWS Management Console, a AWS CLI, ou os SDKs da AWS para adicionar, listar e excluir tags nos clusters do Aurora DSQL. É possível adicionar tags durante e após a criação do cluster usando o Console da AWS. Para marcar um cluster após a criação com a AWS CLI, use a operação `TagResource`.

Marcar clusters com um nome

O Aurora DSQL cria clusters com um identificador globalmente exclusivo atribuído como nome do recurso da Amazon (ARN). Se você quiser atribuir um nome fácil ao cluster, recomendamos que use uma tag.

Se você criar um cluster com o console do Aurora DSQL, o Aurora DSQL criará automaticamente uma tag. Essa tag tem uma chave de nome e um valor gerado automaticamente que representa o nome do cluster. Esse valor é configurável, então você pode atribuir um nome mais fácil ao cluster. Se um cluster tiver uma tag de nome com um valor associado, você poderá ver o valor em todo o console do Aurora DSQL.

Requisitos de marcação

As tags têm os seguintes requisitos:

- As chaves não podem ser prefixadas com `aws:`.
- As chaves devem ser exclusivas por conjunto de tags.
- Uma chave deve ter entre 1 e 128 caracteres permitidos.
- Um valor deve ter entre 0 e 256 caracteres permitidos.
- Os valores não precisam ser exclusivos por conjunto de tags.
- Os caracteres permitidos para chaves e valores são letras, dígitos, espaço em branco e qualquer um dos seguintes símbolos: _ . : / = + - @.
- As chaves e os valores diferenciam letras maiúsculas de minúsculas.

Observações sobre o uso de marcação

Ao usar tags no Aurora DSQL, considere o seguinte:

- Ao usar a CLI ou a API, forneça o nome do recurso da Amazon (ARN) do recurso do Aurora DSQL com o qual deseja trabalhar. Para ter mais informações, consulte Amazon Resource Name (ARNs) format for Aurora DSQL resources.
- Cada recurso tem um conjunto de tags, que é uma coleção de uma ou mais tags atribuídas ao recurso.
- Cada recurso pode ter até 50 tags por conjunto de tags.
- Se você excluir um recurso, todas as tags associadas serão excluídas.
- É possível adicionar tags ao criar um recurso. Você pode visualizar e modificar tags usando as seguintes operações de API: `TagResource`, `UntagResource` e `ListTagsForResource`.
- Você pode usar tags com políticas do IAM. Também é possível usá-las para gerenciar o acesso a clusters do Aurora DSQL e controlar quais ações podem ser aplicadas a esses recursos. Para saber mais, consulte Controlar o acesso a recursos da AWS usando tags.
- Você pode usar tags para várias outras atividades na AWS. Para saber mais, consulte Common tagging strategies.

Considerações para trabalhar com Amazon Aurora DSQL

Considere os comportamentos a seguir ao trabalhar com Amazon Aurora DSQL. Para ter mais informações sobre compatibilidade e suporte do PostgreSQL, consulte Compatibilidade com recursos SQL no Aurora DSQL. Com relação a cotas e limites, consulte Cotas de cluster e limites de banco de dados no Amazon Aurora DSQL.

- O Aurora DSQL não conclui as operações `COUNT(*)` antes do tempo limite da transação para tabelas grandes. Para recuperar a contagem de linhas da tabela do catálogo do sistema, consulte Using systems tables and commands in Aurora DSQL.
- Os drivers que estão chamando `PG_PREPARED_STATEMENTS` podem fornecer uma visão inconsistente das instruções preparadas em cache para o cluster. Você pode ver mais do que o número esperado de instruções preparadas por conexão para o mesmo cluster e perfil do IAM. O Aurora DSQL não preserva o nome das instruções que você prepara.
- Em raros cenários de comprometimento de clusters vinculados a várias regiões, pode levar mais tempo do que o esperado para que a disponibilidade de confirmação da transação seja retomada. Em geral, as operações automatizadas de recuperação de clusters podem gerar erros transitórios de controle de simultaneidade ou de conexão. Na maioria dos casos, você só verá os efeitos para uma porcentagem da sua workload. Quando você vir esses erros transitórios, tente novamente a transação ou se reconecte com seu cliente.
- Alguns clientes SQL, como o DataGrip, fazem chamadas abrangentes aos metadados do sistema para preencher as informações do esquema. O Aurora DSQL não oferece suporte a todas essas informações e exibe erros. Esse problema não afeta a funcionalidade de consulta SQL, mas pode afetar a exibição do esquema.
- O perfil de administrador tem um conjunto de permissões relacionadas a tarefas de gerenciamento de banco de dados. Por padrão, essas permissões não se estendem aos objetos que outros usuários criam. O perfil de administrador não pode nem conceder permissões nesses objetos criados pelo usuário a outros usuários, nem as revogar. O usuário administrador pode conceder a si mesmo qualquer outro perfil para obter as permissões necessárias nesses objetos.

Cotas de cluster e limites de banco de dados no Amazon Aurora DSQL

As seções a seguir descrevem as cotas de cluster e os limites de banco de dados para o Aurora DSQL.

Cotas de cluster

Sua Conta da AWS tem as cotas de cluster a seguir no Aurora DSQL. Para solicitar um aumento nas cotas de serviço para clusters de região única e multirregionais em uma Região da AWS específica, use a página do console Service Quotas. Para outros aumentos de cota, entre em contato com o AWS Support.

Descrição	Limite padrão	Configurável?	Código de erro do Aurora DSQL
Máximo de clusters de região única por Conta da AWS	20 clusters	Sim	Código de erro `ServiceQuotaExceededExceptio`
Máximo de clusters multirregionais por Conta da AWS	5 clusters	Sim	Código de erro `ServiceQuotaExceededExceptio`
Armazenamento máximo por cluster	Limite padrão de 10 TiB e até 128 TiB com aumento de limite aprovado	Sim	`DISK_FULL(53100)`

Descrição	Limite padrão	Configurável?	Código de erro do Aurora DSQL
Máximo de conexões por cluster	10.000 conexõ	Sim	`TOO_MANY_CONNECTIONS(53300)`
Máximo de conexões por cluster	100 conexões por segundo	Não	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacidade de expansão máxima da conexão por cluster	1.000 conexõe	Não	Nenhum código do erro
Máximo de trabalhos de restauração simultâneos	4	Não	Nenhum código do erro
Taxa de reabastec imento de conexão	100 conexões por segundo	Não	Nenhum código do erro

Limites de banco de dados no Aurora DSQL

A tabela a seguir descreve os limites de banco de dados no Aurora DSQL.

Descrição	Limite padrão	Configurável?	Código de erro do Aurora DSQL	Mensagem de erro
Tamanho máximo combinado das colunas usadas em uma chave primária	1 KiB	Não	54000	ERROR: key size too large
Tamanho máximo combinado das colunas em um índice secundário	1 KiB	Não	54000	ERROR: key size too large
Tamanho máximo de uma linha em uma tabela	2 MiB	Não	54000	ERROR: maximum row size excee
Tamanho máximo de uma coluna que não faz parte de um índice	1 MiB	Não	54000	ERROR: maximum column size ex
Número máximo de colunas em uma chave primária ou em um índice secundário	8	Não	54011	ERROR: more than 8 column key are not supported

Descrição	Limite padrão	Configurável?	Código de erro do Aurora DSQL	Mensagem de erro
Número máximo de colunas em uma tabela	255	Não	54011	ERROR: tables can have at mos
Número máximo de índices em uma tabela	24	Não	54000	ERROR: more than 24 indexes p allowed
Tamanho máximo de todos os dados modificados em uma transação de gravação	10 MiB	Não	54000	ERROR: transaction size limit DETAIL: Current transaction s 10mb
Número máximo de linhas de tabela e índice que podem ser alteradas em um bloco de transação	3.000 linhas por transação . Consulte Considerações sobre o Aurora DSQL para compatibilidade com o PostgreSQL.	Não	54000	ERROR: transaction row limit
Quantidade básica máxima de memória que uma operação de consulta pode usar	128 MiB por transação	Não	53200	ERROR: query requires too muc out of memory.

Descrição	Limite padrão	Configurável?	Código de erro do Aurora DSQL	Mensagem de erro
Número máximo de esquemas definidos em um banco de dados	10	Não	54000	ERROR: more than 10 schemas n
Número máximo de tabelas em um banco de dados	1.000 tabelas	Não	54000	ERROR: creating more than 100 allowed
Número máximo de bancos de dados em um cluster	1	Não	Nenhum código do erro	ERROR: unsupported statement
Tempo máximo da transação	5 minutos	Não	54000	ERROR: transaction age limit exceeded
Duração máxima da conexão	60 minutos	Não	Nenhum código do erro	Nenhuma mensagem de erro
Número máximo de visualizações em um banco de dados	5.000	Não	54000	ERROR: creating more than 500 allowed
Tamanho máximo de definição de visualização	2 MiB	Não	54000	ERROR: view definition too la

Para ver os limites de tipo de dados específicos do Aurora DSQL, consulte Tipos de dados compatíveis no Aurora DSQL.

Referência de API do Aurora DSQL

Além do AWS Management Console e da AWS Command Line Interface (AWS CLI), o Aurora DSQL também oferece uma interface de API. Você pode usar as operações de API para gerenciar recursos no Aurora DSQL.

Para obter uma lista alfabética das operações de API, consulte Actions.

Para obter uma lista alfabética de tipos de dados, consulte Tipos de dados.

Para obter uma lista de parâmetros de consulta comuns, consulte Parâmetros comuns.

Para obter descrições dos códigos de erro, consulte Erros comuns.

Para ter mais informações sobre a AWS CLI, consulte a referência da AWS Command Line Interfacepara o Amazon DSQL.

Solução de problemas no Aurora DSQL

> **Note**
>
> Os tópicos a seguir fornecem orientações para a solução de erros e problemas com os quais você pode se deparar ao usar o Aurora DSQL. Se encontrar um problema que não esteja listado aqui, entre em contato com o AWS Support.

Tópicos

Solução de problemas de erros de conexão

error: unrecognized SSL error code: 6

Causa: você deve estar usando uma versão do psql anterior à versão 14, que não permite indicação de nome de servidor (SNI). O SNI é necessário para entrar no Aurora DSQL.

Você pode verificar a versão do cliente com `psql --version`.

erro: NetworkUnreachable

Um erro `NetworkUnreachable` durante as tentativas de conexão pode indicar que seu cliente não permite conexões IPv6, em vez de sinalizar um problema real na rede. Esse erro geralmente ocorre em instâncias somente IPv4 devido à forma como os clientes PostgreSQL lidam com conexões de pilha dupla. Quando um servidor permite o modo de pilha dupla, esses clientes primeiro resolvem os nomes de host para endereços IPv4 e IPv6. Eles tentam primeiro uma conexão IPv4 e, se a conexão inicial falhar, tentam IPv6. Se o seu sistema não permitir IPv6, você verá um erro geral `NetworkUnreachable` em vez de uma mensagem clara de “IPv6 not supported”.

Solucionar erros de autenticação

IAM authentication failed for user "..."

Quando você gera um token de autenticação do IAM para o Aurora DSQL, a duração máxima que você pode definir é uma semana. Após uma semana, você não pode se autenticar com esse token.

Além disso, o Aurora DSQL rejeitará sua solicitação de conexão se o perfil assumido tiver expirado. Por exemplo, se você tentar se conectar com um perfil temporário do IAM, mesmo que o token de autenticação não tenha expirado, o Aurora DSQL rejeitará a solicitação de conexão.

Para saber mais sobre como o IAM funciona com o Aurora DSQL, consulte Understanding authentication and authorization for Aurora DSQL e AWS Identity and Access Management in Aurora DSQL.

An error occurred (InvalidAccessKeyId) when calling the GetObject operation: The AWS Access Key ID you provided does not exist in our records

O IAM rejeitou sua solicitação. Para ter mais informações, consulte Why requests are signed.

IAM role <role> does not exist

O Aurora DSQL não conseguiu encontrar seu perfil do IAM. Para obter mais informações, consulte os perfis do IAM.

IAM role must look like an IAM ARN

Consulte Identificadores do IAM e ARNs do IAM para obter mais informações.

Solucionar erros de autorização

Role <role> not supported

O Aurora DSQL não aceita a operação `GRANT`. Consulte Supported subsets of PostgreSQL commands in Aurora DSQL.

Cannot establish trust with role <role>

O Aurora DSQL não aceita a operação `GRANT`. Consulte Supported subsets of PostgreSQL commands in Aurora DSQL.

Role <role> does not exist

O Aurora DSQL não conseguiu encontrar o usuário do banco de dados especificado. Consulte Authorize custom database roles to connect to a cluster.

ERROR: permission denied to grant IAM trust with role <role>

Para conceder acesso a um perfil de banco de dados, você deve se conectar ao cluster com o perfil de administrador. Para saber mais, consulte Authorize database roles to use SQL in a database.

ERROR: role <role> must have the LOGIN attribute

Você deve ter permissão para criar perfis de `LOGIN`.

Para resolver esse erro, você deve criar o perfil do PostgreSQL com a permissão `LOGIN`. Para ter mais informações, consulte CREATE ROLE e GRANT na documentação do PostgreSQL.

ERROR: role <role> cannot be dropped because some objects depend on it

Se você eliminar um perfil de banco de dados que tenha uma relação com o IAM, o Aurora DSQL exibirá um erro enquanto você não revogar essa relação usando `AWS IAM REVOKE`. Para saber mais, consulte Revoking authorization.

Solucionar de erros de SQL

Error: Not supported

O Aurora DSQL não oferece suporte a todos os dialetos baseados em PostgreSQL. Para saber mais sobre o que é possível usar, consulte Supported PostgreSQL features in Aurora DSQL.

Error: SELECT FOR UPDATE in a read-only transaction is a no-op

Você está tentando realizar uma operação que não é permitida em uma transação somente leitura. Para saber mais, consulte Understanding concurrency control in Aurora DSQL.

Error: use **`CREATE INDEX ASYNC`** instead

Para criar um índice em uma tabela com linhas existentes, você deve usar o comando `CREATE INDEX ASYNC`. Para saber mais, consulte Creating indexes asynchronously in Aurora DSQL.

Solucionar erros de OCC

OC000 “ERROR: mutation conflicts with another transaction, retry as needed”

OC001 “ERROR: schema has been updated by another transaction, retry as needed”

Sua sessão do PostgreSQL tinha uma cópia em cache do catálogo de esquemas. Essa cópia em cache era válida no momento em que foi carregada. Vamos chamar o momento T1 e a versão V1.

Outra transação atualiza o catálogo no momento T2. Vamos chamar isso de V2.

Quando a sessão original tenta ler do armazenamento no momento T2, ela ainda está usando a versão V1 do catálogo. A camada de armazenamento do Aurora DSQL rejeita a solicitação porque a versão mais recente do catálogo em T2 é V2.

Quando você tenta novamente no momento T3 usando a sessão original, o Aurora DSQL atualiza o cache do catálogo. A transação em T3 está usando o catálogo V2. O Aurora DSQL concluirá a transação desde que nenhuma outra alteração no catálogo tenha ocorrido desde o momento T2.

Solucionar problemas em conexões SSL/TLS

SSL error: certificate verify failed

Esse erro indica que o cliente não consegue verificar o certificado do servidor. Verifique se:

1. O certificado Amazon Root CA 1 está instalado corretamente. Consulte Configurar certificados SSL/TLS para conexões do Aurora DSQL para obter instruções sobre como validar e instalar esse certificado.
2. A variável de ambiente `PGSSLROOTCERT` aponta para o arquivo de certificado correto.
3. O arquivo de certificado tem as permissões corretas.

Unrecognized SSL error code: 6

Esse erro ocorre com clientes PostgreSQL abaixo da versão 14. Para resolver esse problema, atualize seu cliente PostgreSQL para a versão 17.

SSL error: unregistered scheme (Windows)

Esse é um problema conhecido com o cliente psql do Windows ao usar certificados do sistema. Use o método de arquivo de certificado baixado descrito nas instruções em Conectar-se pelo Windows.

Histórico do documento do Guia do usuário do Amazon Aurora DSQL

A tabela a seguir descreve as versões de documentação para o Aurora DSQL.

Alteração	Descrição	Data
Disponibilidade geral (GA) do Amazon Aurora DSQL	O Amazon Aurora DSQL agora está disponível ao público com suporte adicional para monitoramento do CloudWatch, recursos aprimorados de proteção de dados e integração com o AWS Backup. Para ter mais informações, consulte Monitoring Aurora DSQL with CloudWatch, Backup and restore for Amazon Aurora DSQL, e Data encryption for Amazon Aurora DSQL.	27 de maio de 2025
Atualização de AmazonAuroraDSQLFullAccess	Adiciona a capacidade de realizar operações de backup e restauração para clusters do Aurora DSQL, incluindo iniciar, interromper e monitorar tarefas. Também adiciona a capacidade de usar chaves do KMS gerenciadas pelo cliente para criptografia de cluster. Para ter mais informaçõ es, consulte AmazonAuroraDSQLFullAccess e Usar	21 de maio de 2025

	perfis vinculados ao serviço no Aurora DSQL.	
Atualização de AmazonAuroraDSQLConsoleFullAccess	Adiciona a capacidade de realizar operações de backup e restauração para clusters do Aurora DSQL por meio do AWS Console Home. Isso inclui iniciar, interromper e monitorar tarefas. Também é possível usar chaves do KMS gerenciadas pelo cliente para criptografia de cluster e para iniciar o AWS CloudShell. Para ter mais informações, consulte AmazonAuroraDSQLConsoleFullAccess e Using service-linked roles in Aurora DSQL.	21 de maio de 2025
Atualização de AmazonAuroraDSQLReadOnlyAccess	Inclui a capacidade de determinar o nome correto do serviço de endpoint da VPC ao se conectar aos clusters do Aurora DSQL por meio do AWS PrivateLink. O Aurora DSQL cria endpoints exclusivos por célula; portanto, essa API ajuda a garantir que você possa identificar o endpoint correto para o cluster e evitar erros de conexão. Para ter mais informações, consulte AmazonAuroraDSQLReadOnlyAccess e Using service-linked roles in Aurora DSQL.	13 de maio de 2025

Atualização de AmazonAuroraDSQLFullAccess	A política adiciona quatro novas permissões para criar e gerenciar clusters de banco de dados em várias Regiões da AWS: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` e `RemovePeerCluster` . Essas permissões incluem controles em nível de recursos e chaves de condição para que você possa controlar quais clusters os usuários podem modificar. A política também adiciona a permissão `GetVpcEndpointServiceName` para ajudar você a se conectar aos clusters do Aurora DSQL por meio do AWS PrivateLink. Para ter mais informações, consulte AmazonAuroraDSQLConsoleFullAccess e Using service-linked roles in Aurora DSQL.	13 de maio de 2025

Atualização de AmazonAuroraDSQLConsoleFullAccess	Adiciona novas permissões ao Aurora DSQL para oferecer suporte ao gerenciamento de clusters multirregionais e à conexão de endpoints da VPC. As novas permissões incluem: `PutMultiRegionProperties`, `PutWitnessRegion`, `AddPeerCluster`, `RemovePeerCluster` e `GetVpcEndpointServiceName`. Para ter mais informações, consulte AmazonAuroraDSQLConsoleFullAccess e Usar perfis vinculados ao serviço no Aurora DSQL.	13 de maio de 2025
Atualização de AuroraDsqlServiceLinkedRolePolicy	Adiciona à política a capacidade de publicar métricas nos namespaces `AWS/AuroraDSQL` e `AWS/Usage` CloudWatch. Isso permite que o serviço ou perfil associado emita dados de uso e desempenho mais abrangentes para seu ambiente do CloudWatch. Para ter mais informações, consulte AuroraDsqlServiceLinkedRolePolicy e Using service-linked roles in Aurora DSQL.	8 de maio de 2025

AWS PrivateLink para Amazon Aurora DSQL	O Aurora DSQL agora é compatível com o AWS PrivateLink. Com o AWS PrivateLink, é possível simplificar a conectividade de rede privada entre nuvens privadas virtuais (VPCs), o Aurora DSQL e seus data centers on-premises usando endpoints da Amazon VPC de interface e endereços IP privados. Para ter mais informações, consulte Managing and connecting to Amazon Aurora DSQL clusters using AWS PrivateLink.	8 de maio de 2025
Lançamento inicial	Versão inicial do Guia do usuário do Amazon Aurora DSQL.	3 de dezembro de 2024