aws

Guia do usuário

Amazon Aurora DSQL

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Aurora DSQL: Guia do usuário

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o Amazon Aurora DSQL? 1
Quando usar 1
Atributos principais 1
Disponibilidade do Região da AWS 3
Clusters multirregionais 5
Preços 6
Próximas etapas 6
Introdução 7
Pré-requisitos 7
Criar um cluster de região única 7
Conectar-se a um cluster 8
Executar comandos SQL 9
Criar um cluster multirregional 9
Solução de problemas 12
Autenticação e autorização 13
Gerenciar clusters 13
Conectar-se a um cluster 13
Perfis do PostgreSQL e do IAM 14
Usar ações de política do IAM com o Aurora DSQL 15
Usar ações de política do IAM para se conectar a clusters 15
Usar ações de política do IAM para gerenciar clusters 16
Revogar a autorização usando o IAM e o PostgreSQL 17
Gerar um token de autenticação 18
Console 19
AWS CloudShell 19
AWS CLI 21
SDKs do Aurora DSQL 22
Perfis de banco de dados e autenticação do IAM 31
Perfis do IAM 31
Usuários do IAM 32
Connect 32
Consulta 32
Visualização de mapeamentos 33
Revogar 33

Aurora DSQL e PostgreSQL 35
Destaques da compatibilidade 35
Benefícios da arquitetura distribuída 36
Compatibilidade com SQL 37
Tipos de dados compatíveis 37
Recursos compatíveis 43
Subconjuntos de comandos SQL compatíveis 47
Guia de migração 68
Controle de simultaneidade 76
Conflitos de transação 76
Diretrizes para otimizar o desempenho da transação 77
DDL e transações distribuídas 77
Chaves primárias 79
Estrutura e armazenamento de dados 79
Diretrizes para escolher uma chave primária 79
Sequências e colunas de identidade 80
Funções de manipulação de sequências 80
Colunas de identidade 83
Trabalhar com sequências e colunas de identidade 85
Índices assíncronos 87
Sintaxe 87
Parâmetros 88
Observações de uso 89
Como criar um índice 89
Consultar um índice 90
Falhas na criação de índices únicos 92
Violações de unicidade 92
Tabelas e comandos de sistema 94
Tabelas de sistema 94
Consultas úteis do sistema 105
O comando ANALYZE 106
Planos EXPLAIN 107
Plano EXPLAIN do PostgreSQL 107
Principais elementos 108
Filtrar 109
Ler os planos EXPLAIN 110

DPUs em EXPLAIN ANALYZE 114
Gerenciar clusters do Aurora DSQL 118
Clusters de região única 118
Usar SDKs da AWS 118
Usar a AWS CLI 157
Clusters multirregionais 160
Usar SDKs da AWS 161
Usar a AWS CLI 215
CloudFormation 221
Configuração inicial 221
Localizar clusters 222
Atualizar configuração 222
Ciclo de vida do cluster do Aurora DSQL 223
Status de cluster 223
Visualizar status de clusters 226
Programar com o Aurora DSQL 227
Conectores 227
Conector JDBC 228
Conector do Python 233
Connector Go 244
Conectores do Node.js 252
Conector Ruby 260
Conector .NET 267
Acessar o Aurora DSQL 272
Clientes SQL 273
DBeaver 274
JetBrains DataGrip 277
Psql 279
VSCode 280
Solução de problemas 281
Ferramentas de conectividade com o banco de dados 282
Adaptadores do Aurora DSQL 282
Exemplos de driver de banco de dados 283
Exemplos de ORM e framework 285
Carregamento de dados 286
Escolher uma abordagem de migração 286

Aurora DSQL Loader 287
Caminhos de migração 292
Usar \copy do PostgreSQL 294
Recursos adicionais 295
IA generativa 296
Servidor MCP para AWS Labs Aurora DSQL 296
Direcionamento do Aurora DSQL: skills e powers 304
Query Editor 310
Pré-requisitos 310
Trabalhar com o Editor de consultas 311
Editores de consultas: usar o JupyterLab com o Aurora DSQL 313
Introdução 313
Caderno de exemplo 315
Outras fontes de leitura 316
Backup e restauração 317
Conceitos básicos do AWS Backup 317
Restaurar seus backups 317
Restaurar clusters de região única 318
Restaurar clusters multirregionais 318
Monitoramento e conformidade 318
Recursos adicionais 319
Monitorar e registrar em log 320
Monitoramento com CloudWatch 320
Observabilidade 320
Uso 322
Registrar em log com o CloudTrail 323
Eventos de gerenciamento 324
Eventos de dados 325
Segurança 327
Políticas gerenciadas pela AWS 328
AmazonAuroraDSQLFullAccess 328
AmazonAuroraDSQLReadOnlyAccess 330
AmazonAuroraDSQLConsoleFullAccess 330
AuroraDSQLServiceRolePolicy 332
Atualizações da política 332
Proteção de dados 338

Criptografia de dados 339
Proteção de dados em regiões testemunhas 341
Certificados SSL/TLS 341
Criptografia de dados 339
tipos de chave do KMS 348
Criptografia em repouso 348
Usar chaves do KMS e de dados 350
Autorizar o uso da chave do KMS 352
Contexto de criptografia 354
Como monitorar o AWS KMS 355
Criar um cluster criptografado 357
Remover ou atualizar uma chave 359
Considerações 362
Gerenciamento de identidade e acesso 363
Público 363
Autenticação com identidades 364
Gerenciar o acesso usando políticas 365
Como o Aurora DSQL funciona com o IAM 367
Exemplos de políticas baseadas em identidade 372
Solução de problemas 378
Políticas baseadas em recursos 380
Quando usar 381
Criar com políticas 382
Adicionar e editar políticas 385
Visualizar política 388
Remover política 389
Exemplos de políticas 391
Bloqueio de acesso público 395
Operações de API 398
Usar perfis vinculados ao serviço 401
Permissões de perfis vinculados ao serviço para o Aurora DSQL 401
Criar um perfil vinculado ao serviço 402
Editar um perfil vinculado ao serviço 402
Excluir um perfil vinculado ao serviço 402
Regiões compatíveis com perfis vinculados ao serviço do Aurora DSQL 403
Usar chaves de condição do IAM 403

Criar um cluster em uma região específica ... 403
Criar um cluster multirregional em regiões específicas ... 404
Criar um cluster multirregional com uma região testemunha específica ... 405
Resposta a incidentes ... 406
Validação de compatibilidade ... 406
Resiliência ... 407
Backup e restauração ... 407
Replicação ... 408
Alta disponibilidade ... 408
Teste de injeção de falhas ... 409
Segurança da infraestrutura ... 410
Gerenciar clusters usando o AWS PrivateLink ... 410
Análise de configuração e vulnerabilidade ... 421
Prevenção contra o ataque do “substituto confuso” em todos os serviços ... 422
Práticas recomendadas de segurança ... 423
Práticas recomendadas de segurança de detecção ... 424
Práticas recomendadas de segurança preventiva ... 425
Marcar recursos ... 427
Name tag ... 427
Requisitos de marcação ... 427
Observações sobre o uso de marcação ... 428
Considerações ... 429
Cotas e limites ... 431
Cotas de cluster ... 431
Limites de banco de dados ... 432
Referência de API ... 437
Solução de problemas ... 276
Erros de conexão ... 438
Erros de autenticação ... 439
Erros de autorização ... 440
Erros de SQL ... 440
Erros de OCC ... 441
Conexões SSL/TLS ... 441
Fornecer feedback ... 443
Canais de feedback ... 443
Solicitações efetivas de recursos ... 443

Histórico do documento 444

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 o 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 amplo 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.

Nome da Região	Região	Endpoint	Protocolo	
Leste dos EUA (Ohio)	us-east-2	dsql.us-east-2.api.aws dsql-fips.us-east-2.api.aws	HTTPS HTTPS	
Leste dos EUA (Norte da Virgínia)	us-east-1	dsql.us-east-1.api.aws dsql-fips.us-east-1.api.aws	HTTPS HTTPS	
Oeste dos EUA (Oregon)	us-west-2	dsql.us-west-2.api.aws dsql-fips.us-west-2.api.aws	HTTPS HTTPS	
Ásia-Pacífico (Melbourne)	ap-southeast-4	dsql.ap-southeast-4.api.aws	HTTPS	

Nome da Região	Região	Endpoint	Protocolo
Ásia-Pacífico (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS
Ásia-Pacífico (Seul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Ásia-Pacífico (Sydney)	ap-southeast-2	dsql.ap-southeast-2.api.aws	HTTPS
Ásia-Pacífico (Tóquio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Canadá (Central)	ca-central-1	dsql.ca-central-1.api.aws dsql-fips.ca-central-1.api.aws	HTTPS HTTPS
Oeste do Canadá (Calgary)	ca-west-1	dsql.ca-west-1.api.aws dsql-fips.ca-west-1.api.aws	HTTPS HTTPS
Europa (Frankfurt)	eu-central-1	dsql.eu-central-1.api.aws	HTTPS
Europa (Irlanda)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europa (Londres)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europa (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS

Disponibilidade de clusters multirregionais para o Aurora DSQL

Você pode criar clusters multirregionais do Aurora DSQL dentro de conjuntos de regiões específicas da AWS. Cada conjunto de regiões agrupa regiões geograficamente relacionadas que podem trabalhar juntas em um cluster multirregional.

Regiões dos EUA

- Leste dos EUA (Norte da Virgínia)
- Leste dos EUA (Ohio)
- Oeste dos EUA (Oregon)

Regiões da Ásia-Pacífico

- Ásia-Pacífico (Osaka)
- Ásia-Pacífico (Seul)
- Ásia-Pacífico (Tóquio)

Regiões europeias

- Europa (Frankfurt)
- Europa (Irlanda)
- Europa (Londres)
- Europa (Paris)

Limitações importantes

Os clusters multirregionais devem ser criados em um único conjunto de regiões. Por exemplo, você não pode criar um cluster que inclua tanto as regiões Leste dos EUA (Norte da Virgínia) quanto Europa (Irlanda).

> **Important**
>
> Atualmente, o Aurora DSQL não dá suporte a clusters multirregionais intercontinentais.

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 com clientes compatíveis com o PostgreSQL
- Aurora DSQL e PostgreSQL

Conceitos básicos do Aurora DSQL

O Amazon Aurora DSQL é um banco de dados relacional distribuído, totalmente gerenciado 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 vai acessar clusters com o Console da AWS e, opcionalmente, interagir com seu banco de dados usando os clientes do PostgreSQL. Ao final, você terá um cluster do Aurora DSQL funcional configurado e pronto para uso em workloads de teste ou produção.

Tópicos

- Pré-requisitos
- Etapa 1: criar um cluster do Aurora DSQL de região única
- Etapa 2: conectar-se a um cluster do Aurora DSQL
- Etapa 3: executar exemplos de comando SQL no Aurora DSQL
- Etapa 4 (opcional): criar um cluster multirregional
- Solução de problemas

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 console.
- Sua identidade do IAM deve atender aos seguintes critérios:
  - Ter acesso para executar qualquer ação em qualquer recurso em sua Conta da AWS.
  - A política gerenciada pela AWS `AmazonAuroraDSQLConsoleFullAccess` é anexada.

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 Console de gerenciamento da AWS e abra o console do Aurora DSQL em https://console.aws.amazon.com/dsql.

2. Escolha Criar cluster e Região única.
3. (Opcional) Altere o valor da tag nome padrão.
4. (Opcional) Adicione tags adicionais para esse cluster.
5. (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. Ao usar uma chave gerenciada pelo cliente, garanta que a política de chave conceda as permissões necessárias ao Aurora DSQL. Para obter mais informações, consulte Política de chaves para uma chave gerenciada pelo cliente.
   - Selecione Habilitar proteção contra exclusão para impedir que o cluster de banco de dados seja excluído. Por padrão, a proteção contra exclusão fica selecionada.
   - Selecione Política baseada em recursos (avançada) para especificar políticas de controle de acesso para esse cluster.
6. Selecione Criar cluster.
7. O console faz com que você retorne para a página Clusters. Banner de notificação indicando que o cluster está sendo criado. Selecione o ID do cluster para abrir a visualização dos detalhes do cluster.

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

O Aurora DSQL comporta várias maneiras de se conectar ao seu cluster, incluindo o Editor de consultas do DSQL, AWS CloudShell, o cliente psql local e outras ferramentas compatíveis com o PostgreSQL. Nesta etapa, você se conecta usando o Editor de consultas do Aurora DSQL, que oferece uma maneira rápida de começar a interagir com seu novo cluster.

Como se conectar usando o Editor de consultas

1. No console do Aurora DSQL (https://console.aws.amazon.com/dsql), abra a página Clusters e confirme se a criação do cluster foi concluída e se seu status é Ativo.
2. Escolha seu cluster na lista ou o ID do cluster para abrir a página de detalhes do cluster.
3. Escolha Conectar com o Editor de consultas.
4. Escolha Conectar como administrador para o cluster que acabou de ser criado.
   - Você também pode se conectar com um perfil personalizado. Consulte Usar perfis de banco de dados e autenticação do IAM.

Etapa 3: executar exemplos de comando SQL no Aurora DSQL

Teste o cluster do Aurora DSQL executando instruções SQL. Depois de abrir o cluster no Editor de consultas, selecione e execute cada exemplo de consulta passo a passo.

Executar exemplos de comando SQL no Aurora DSQL

1. Crie um esquema chamado `test`.

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

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

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

3. Insira uma linha de exemplo.

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

4. Leia os valores inseridos.

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

5. Limpar opcionalmente

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

Etapa 4 (opcional): 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.

Use o procedimento a seguir para criar um cluster inicial, criar um segundo cluster em uma região diferente e, depois, 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 console do Aurora DSQL.
2. No painel de navegação, escolha Clusters.
3. Escolha Criar cluster e Multirregião.
4. (Opcional) Altere o valor da tag nome padrão.
5. (Opcional) Adicione tags adicionais para esse cluster.
6. 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.
7. (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. Ao usar uma chave gerenciada pelo cliente, garanta que a política de chave conceda as permissões necessárias ao Aurora DSQL. Para obter mais informações, consulte Política de chaves para uma chave gerenciada pelo cliente.

- Selecione Habilitar proteção contra exclusão para impedir que o cluster de banco de dados seja excluído. Por padrão, a proteção contra exclusão fica selecionada.
- Selecione Política baseada em recursos (avançada) para especificar políticas de controle de acesso para esse cluster.

8. 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.
9. Na notificação Configuração do cluster pendente, escolha Concluir configuração do cluster multirregional. Essa ação inicia a criação de um segundo cluster em outra região.
10. 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.

11. Escolha Criar cluster na ***your-second-region***, em que *your-second-region* é o local do segundo cluster. O console abre na segunda região.
12. (Opcional) Escolha as configurações de cluster para o segundo cluster. Por exemplo, você pode escolher uma AWS KMS key. Ao usar uma chave gerenciada pelo cliente, garanta que a política de chave conceda as permissões necessárias ao Aurora DSQL. Para obter mais informações, consulte Política de chaves para uma chave gerenciada pelo cliente.
13. Escolha Criar cluster para criar o segundo cluster.
14. Escolha Emparelhar na ***initial-cluster-region***, em que *initial-cluster-region* é a região que hospeda o primeiro cluster que você criou.
15. 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. Escolha Conectar com o Editor de consultas.
5. Escolha Conectar-se como administrador.

6. Crie um esquema de exemplo e uma tabela e insira dados seguindo as etapas em Etapa 3: executar exemplos de comando SQL no Aurora DSQL.

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. Escolha Conectar com o Editor de consultas.
5. Escolha Conectar-se como administrador.
6. Consulte os dados que você inseriu no segundo cluster.

   Example

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

Solução de problemas

Consulte a seção Solução de problemas da documentação do Aurora DSQL.

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 Console de gerenciamento da AWS, 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 descrever um cluster, sua identidade do IAM deve ter permissões para a ação `dsql:GetCluster` do IAM, como no exemplo de ação de política a seguir.

```
{
  "Effect": "Allow",
  "Action": "dsql:GetCluster",
  "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 do

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

JSON

```
{
"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.

JSON

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

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, a AWS CLI ou SDKs, ele expirará automaticamente em 15 minutos por padrão. 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 Console de gerenciamento da AWS 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 (opcional): 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, você precisa Criar um cluster do Aurora DSQL.

Como gerar token de autenticação usando o AWS CloudShell

1. Faça login no Console de gerenciamento da AWS 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. 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?.

4. 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
```

5. 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:
```

6. 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:

> **Note**
>
> A geração de tokens é uma operação local que assina a solicitação usando suas credenciais atuais do IAM. Ela não entra em contato com a AWS para validar as credenciais. Se credenciais expirarem ou forem inválidas, a geração do token ainda será bem-sucedida, mas a tentativa de conexão falhará. Suas credenciais do IAM devem ser válidas antes de gerar um token.

- 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

> **Tip**
>
> A AWS recomenda usar Conector do Aurora DSQL para Python, que gerencia a geração de tokens automaticamente.

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 personalizado, use `generate_connect_auth_token`.

```
import boto3

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

C++ SDK

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 personalizado, 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) {
    DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQLClient client{clientConfig};
    std::string token = "";

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

    std::cout << token << std::endl;
    return token;
}

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

JavaScript SDK

> **Tip**
>
> A AWS recomenda usar Conectores do Aurora DSQL para Node.js, que lida com a geração de tokens automaticamente.

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 personalizado, 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

> **Tip**
>
> A AWS recomenda usar Conectar-se a clusters do Aurora DSQL com um conector JDBC, que gerencia a geração de tokens automaticamente.

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 personalizado, 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.builder().build())
                .build();

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

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

Rust SDK

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 personalizado, 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

> **Tip**
>
> A AWS recomenda usar Conectar-se a clusters do Aurora DSQL com um conector Ruby, que gerencia a geração de tokens automaticamente.

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 personalizado, use `generate_db_connect_auth_token`.

```
require 'aws-sdk-dsql'

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

  token_generator = Aws::DSQL::AuthTokenGenerator.new({
```

```
    :credentials => credentials
  })

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

PHP SDK

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 personalizado, use `generateDbConnectAuthToken`.

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

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

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

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

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

.NET

> **Tip**
>
> A AWS recomenda usar Conectar-se a clusters do Aurora DSQL com um conector .NET, que gerencia a geração de tokens automaticamente.

> **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 personalizado, 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;

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

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

Console.WriteLine(token);
```

Go

> **Tip**
>
> A AWS recomenda usar Conectar-se a clusters do Aurora DSQL com um conector Go, que gerencia a geração de tokens automaticamente.

O AWS SDK for Go v2 oferece um método integrado para gerar tokens de autenticação no pacote `github.com/aws/aws-sdk-go-v2/feature/dsql/auth`.

- Se você estiver se conectando com um perfil `admin`, use `auth.GenerateDBConnectAdminAuthToken`.
- Se você estiver se conectando com um perfil de banco de dados personalizado, use `auth.GenerateDbConnectAuthToken`.

```
package main

import (
 "context"
 "fmt"

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

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

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

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

 fmt.Println(token)
}
```

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 Console de gerenciamento da AWS.

- 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.

Visualizando mapeamentos de perfis do IAM para o banco de dados

Para visualizar os mapeamentos entre as perfis do IAM e as funções do banco de dados, consulte a tabela do sistema `sys.iam_pg_role_mappings`.

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

Resultado do exemplo:

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

Esta tabela mostra todos os mapeamentos entre os perfis do IAM (identificados pelo ARN) e as funções do banco de dados 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 de aplicações distribuídas modernas e nativas da nuvem.

Destaques da compatibilidade com o PostgreSQL

Atualmente, o Aurora DSQL baseia-se na versão 16 do PostgreSQL. Alguns dos principais destaques são:

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`.

Sintaxe de 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.

Benefícios da arquitetura distribuída

O design distribuído e sem compartilhamento do Aurora DSQL oferece benefícios de desempenho e de escalabilidade que transcendem os oferecidos pelos bancos de dados tradicionais de nó único. Alguns dos principais recursos são:

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 saber mais sobre como usar o comando EXPLAIN no PostgreSQL, consulte DDL e transações distribuídas no Aurora DSQL.

Compatibilidade com recursos SQL no Aurora DSQL

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
- SQL compatível com o Aurora DSQL
- Subconjuntos de comandos SQL compatíveis no Aurora DSQL
- Migrar do PostgreSQL para o Aurora DSQL

Tipos de dados compatíveis no Aurora DSQL

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

Tópicos

- Tipos de dados numéricos
- Tipos de dados de caracteres
- Tipos de dados de data e hora
- Tipos de dados diversos
- Tipos de dados de tempo de execução de consulta

Tipos de dados numéricos

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

Nome	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

Nome	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. [1] O padrão é `numeric (18,6)`.	8 bytes + 2 bytes por dígito de precisão. O tamanho máximo para é 27 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 caracteres

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

Nome	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

Nome	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.

Nome	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.

Nome	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.

Funções JSON em tempo de execução

O Aurora DSQL permite o uso de JSON e JSONB como tipos de dados de runtime para processamento de consultas. Armazene dados JSON como colunas `text` e converta-os em JSON durante a execução da consulta para usar funções e operadores JSON do PostgreSQL.

O Aurora DSQL é compatível com a maioria das funções JSON do PostgreSQL com comportamento idêntico listadas em 9.1.6 JSON Functions and Operators.

Funções que exibem os tipos JSON ou JSONB podem exigir conversão adicional em `text` para exibição adequada.

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

A função retorna uma resposta semelhante à seguinte:

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

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.

`SELECT` Comando

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`	
`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`

Cláusula primária	Cláusulas compatíveis
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.
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.

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

Command	Cláusula primária	Cláusulas compatíveis
		`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	Alias
`COMMIT`	`[WORK | TRANSACTION ]` `[AND NO CHAIN]`	`END`

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

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

Command	Cláusulas compatíveis	Alias
	[AND NO CHAIN]	

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 fornece informações detalhadas sobre os comandos SQL aceitos, concentrando-se em comandos com extensos conjuntos de parâmetros e subcomandos. Por exemplo, no PostgreSQL, CREATE TABLE oferece vários parâmetros e cláusulas, alguns dos quais podem ser usados no Aurora DSQL. Essa seção descreve todos os elementos da sintaxe do PostgreSQL que o Aurora DSQL aceita para esses comandos.

Tópicos

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

`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 |
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:

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

and like_option is:

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

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

Colunas de identidade

> **Note**
>
> Ao usar colunas de identidade, é necessário considerar cuidadosamente o valor do cache. Para ter mais informações, consulte o texto explicativo “Importante” na página CREATE SEQUENCE.
>
> Para obter orientações sobre a melhor forma de usar colunas de identidade com base nos padrões de workload, consulte Trabalhar com sequências e colunas de identidade.

A cláusula GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY (*sequence_options*) cria a coluna como uma coluna de identidade. Ela terá uma sequência implícita anexada a ela e, nas linhas recém-inseridas, a coluna terá automaticamente valores da sequência atribuída a ela. Essa coluna é implicitamente NOT NULL.

As cláusulas ALWAYS e BY DEFAULT determinam como os valores especificados pelo usuário são tratados explicitamente nos comandos INSERT e UPDATE.

Em um comando INSERT, se ALWAYS estiver selecionado, um valor especificado pelo usuário só será aceito se a instrução INSERT especificarOVERRIDING SYSTEM VALUE. Se BY DEFAULT estiver selecionado, o valor especificado pelo usuário terá precedência.

Em um comando UPDATE, se ALWAYS estiver selecionado, qualquer atualização da coluna para qualquer valor diferente de DEFAULT será rejeitada. Se BY DEFAULT estiver selecionado, a coluna poderá ser atualizada normalmente. (Não há nenhuma cláusula OVERRIDING para o comando UPDATE.)

A cláusula *sequence_options* pode ser usada para substituir os parâmetros da sequência. As opções disponíveis incluem aquelas mostradas para CREATE SEQUENCE, mais SEQUENCE NAME *name*. Sem SEQUENCE NAME, o sistema escolherá um nome não utilizado para a sequência.

ALTER TABLE

ALTER TABLE altera a definição de uma tabela.

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

where action is one of:

    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type
    ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET
 sequence_option | RESTART [ [ WITH ] restart ] } [...]
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
```

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

Ações da coluna de identidade

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

Esses formulários alteram se uma coluna é uma coluna de identidade ou alteram o atributo de geração de uma coluna de identidade existente. Para mais detalhes, consulte `CREATE TABLE`. De modo semelhante a `SET DEFAULT`, esses formulários afetam apenas o comportamento dos comandos `INSERT` e `UPDATE` subsequentes; eles não fazem com que as linhas que já estão na tabela sejam alteradas.

É possível usar a opção *`sequence_option`* com `ALTER SEQUENCE`, como `INCREMENT BY`. Esses formulários alteram a sequência subjacente a uma coluna de identidade existente.

`DROP IDENTITY [ IF EXISTS ]`

Esse formulário remove a propriedade de identidade de uma coluna. Se `DROP IDENTITY IF EXISTS` for especificado e a coluna não for uma coluna de identidade, nenhum erro será gerado. Nesse caso, será emitido um aviso em vez disso.

`CREATE SEQUENCE`

`CREATE SEQUENCE`: define um novo gerador de sequência.

> **Important**
>
> No PostgreSQL, a especificação do `CACHE` é opcional e o padrão é 1. Em um sistema distribuído como o Amazon Aurora DSQL, as operações de sequência envolvem coordenação, e o tamanho de cache de 1 pode aumentar os custos indiretos de coordenação quando há alta simultaneidade. Embora valores de cache mais altos permitam atender a números de sequência a partir de intervalos pré-alocados localmente, o que melhora o throughput, valores reservados não utilizados podem ser perdidos, o que torna as lacunas e os efeitos de ordenação mais visíveis. Como as aplicações diferem com relação à susceptibilidade à ordenação de alocação em contraposição ao throughput, o Amazon Aurora DSQL exige que o `CACHE` seja especificado explicitamente e no momento permite o uso de `CACHE = 1` ou `CACHE >= 65536`, oferecendo uma distinção clara entre o comportamento de alocação mais próximo da geração estritamente sequencial e a alocação otimizada para workloads altamente simultâneas.

Quando o `CACHE >= 65536`, os valores de sequência permanecerão garantidamente exclusivos, mas existe a possibilidade de que não sejam gerados em uma ordem crescente estrita entre as sessões e de que haja lacunas, principalmente quando os valores armazenados em cache não são totalmente consumidos. Essas características são consistentes com a semântica do PostgreSQL para sequências armazenadas em cache sob uso simultâneo, caso em que ambos os sistemas garantem valores distintos, mas não uma ordenação estritamente sequencial entre as sessões.

Em uma única sessão de cliente, os valores de sequência nem sempre podem parecer estritamente crescentes, especialmente fora das transações explícitas. Esse comportamento é semelhante a implantações do PostgreSQL que usam agrupamento de conexões. O comportamento de alocação mais próximo de um ambiente PostgreSQL de sessão única pode ser obtido usando `CACHE = 1` ou obtendo valores de sequência em transações explícitas.

Com o `CACHE = 1`, a alocação de sequência segue o comportamento de sequência não armazenada em cache do PostgreSQL.

Para obter orientações sobre a melhor forma de usar sequências com base nos padrões de workload, consulte Trabalhar com sequências e colunas de identidade.

Sintaxe compatível

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

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

Descrição

`CREATE SEQUENCE` cria um gerador de números de sequência. Isso requer a criação e inicialização de uma nova tabela especial de linha única com *`name`*. O gerador pertencerá ao usuário que está emitindo o comando.

Se um nome de esquema for fornecido, a visualização será criada no esquema especificado. Do contrário, a visualização será criada no esquema atual. O nome da sequência deve ser diferente do nome de qualquer outra relação (tabela, sequência, índice, visualização, visão materializada ou tabela estrangeira) no mesmo esquema.

Depois de criar uma sequência, use as funções `nextval`, `currval` e `setval` para operar nela. Essas funções estão documentadas em Funções de manipulação de sequências.

Embora seja possível atualizar uma sequência diretamente, você pode usar uma consulta como:

```
SELECT * FROM name;
```

para examinar alguns dos parâmetros e o estado atual de uma sequência. O campo `last_value` da sequência, especialmente, mostra o último valor alocado por qualquer sessão. (Obviamente, esse valor pode estar obsoleto no momento em que for impresso, se outras sessões estiverem fazendo chamadas `nextval` ativamente.) Outros parâmetros, como *`increment`* e *`maxvalue`*, podem ser observados na visualização `pg_sequences`.

Parâmetros

`IF NOT EXISTS`

Não gera um erro se já houver uma relação com o mesmo nome. Um aviso é emitido nesse caso. Observe que não há garantia de que a relação existente seja semelhante à sequência que teria sido criada e que ela pode até mesmo não ser uma sequência.

`name`

O nome (opcionalmente qualificado para o esquema) da sequência a ser criada.

`data_type`

A cláusula opcional `AS` *`data_type`* especifica o tipo de dados da sequência. O tipos válidos são `bigint`. `bigint` é o padrão. O tipo de dados determina os valores mínimo e máximo padrão da sequência.

`increment`

A cláusula opcional `INCREMENT BY` *`increment`* especifica qual valor é adicionado ao valor da sequência atual para criar outro valor. Um valor positivo criará uma sequência crescente; um valor negativo criará uma sequência decrescente. O valor padrão é 1.

minvalue / **NO MINVALUE**

A cláusula opcional `MINVALUE` *minvalue* determina o valor mínimo que uma sequência pode gerar. Se essa cláusula não for fornecida ou `NO MINVALUE` for especificada, serão usados os padrões. O padrão para uma sequência crescente é 1. O padrão para uma sequência decrescente é o valor mínimo do tipo de dados.

maxvalue / **NO MAXVALUE**

A cláusula opcional `MAXVALUE` *maxvalue* determina o valor máximo da sequência. Se essa cláusula não for fornecida ou `NO MAXVALUE` for especificada, serão usados os padrões. O padrão para uma sequência crescente é o valor máximo do tipo de dados. O padrão para uma sequência decrescente é -1.

CYCLE / **NO CYCLE**

A opção `CYCLE` permite que a sequência seja contornada quando o *maxvalue* ou *minvalue* for atingido por uma sequência crescente ou decrescente, respectivamente. Se o limite for atingido, o próximo número gerado será o *minvalue* ou *maxvalue*, respectivamente.

Se `NO CYCLE` for especificada, qualquer chamada para `nextval` depois que a sequência atingir o valor máximo exibirá um erro. Se tanto `CYCLE` quanto `NO CYCLE` não forem especificados, o padrão será `NO CYCLE`.

start

A cláusula opcional `START WITH` *start* permite que a sequência comece em qualquer lugar. O valor inicial padrão é *minvalue* para sequências crescentes e *maxvalue* para sequências decrescentes.

cache

A cláusula `CACHE` *cache* especifica quantos números de sequência devem ser pré-alocados e armazenados na memória para que o acesso seja mais rápido. O valor aceitável para `CACHE` no Aurora DSQL é 1 ou qualquer número >= 65.536. O valor mínimo é 1 (somente um valor pode ser gerado por vez, o que significa que não há cache).

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

A opção `OWNED BY` faz com que a sequência seja associada a uma coluna específica da tabela, de forma que, se essa coluna (ou a tabela inteira) for eliminada, a sequência também será automaticamente eliminada. A tabela especificada deve ter o mesmo proprietário e estar no mesmo esquema que a sequência. `OWNED BY NONE`, o padrão, especifica que essa associação não existe.

Observações

Use DROP SEQUENCE para remover uma sequência.

Como as sequências são baseadas na aritmética bigint, o intervalo não pode exceder o intervalo de um inteiro de 8 bytes (-9223372036854775808 to 9223372036854775807).

Visto que as chamadas nextval e setval nunca são revertidas, não será possível usar objetos de sequência se for necessária uma atribuição de números de sequência "sem intervalos".

Cada sessão alocará e armazenará em cache valores de sequência sucessivos durante um acesso ao objeto de sequência e aumentará o last_value do objeto de sequência adequadamente. Em seguida, os próximos usos de *cache*-1 de nextval dentro dessa sessão simplesmente exibirão os valores pré-alocados, sem tocar no objeto de sequência. Portanto, quaisquer números alocados, mas não usados em uma sessão, serão perdidos quando essa sessão terminar, gerando "lacunas" na sequência.

Além disso, embora exista a garantia de que várias sessões alocarão valores de sequência distintos, os valores poderão ser gerados fora da sequência quando todas as sessões forem consideradas. Por exemplo, com uma configuração de *cache* de 10, a sessão A pode reservar valores 1..10 e exibir nextval=1 e a sessão B pode reservar valores 11..20 e exibir nextval=11 antes que a sessão A tenha gerado nextval=2. Portanto, com uma configuração de *cache* igual a um, é seguro presumir que os valores nextval serão gerados sequencialmente; com uma configuração de *cache* maior que um, você só deve presumir que todos os valores nextval serão distintos, não que serão gerados de uma forma puramente sequencial. Além disso, last_value exibirá o valor mais recente reservado por qualquer sessão, independentemente de ele já ter sido exibido por nextval.

Outra consideração é que a execução de setval em tal sequência não será percebida por outras sessões enquanto elas não usarem todos os valores pré-alocados que armazenaram em cache.

Exemplos

Crie uma sequência crescente chamada serial, começando com 101:

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

Selecione o próximo número dessa sequência:

```
SELECT nextval('serial');

 nextval
```

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

Selecione o próximo número dessa sequência:

```
SELECT nextval('serial');

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

Use essa sequência em um comando `INSERT`:

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

Redefina a sequência para um valor específico usando `setval`:

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

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

Compatibilidade

`CREATE SEQUENCE` segue o padrão SQL, com as seguintes exceções:

- A obtenção do próximo valor é feita usando a função `nextval()` em vez da expressão padrão `NEXT VALUE FOR`.
- A cláusula `OWNED BY` é uma extensão do PostgreSQL.

`ALTER SEQUENCE`

`ALTER SEQUENCE`: altera a definição de um gerador de sequência.

> **Important**
>
> Ao usar sequências, é necessário considerar cuidadosamente o valor do cache. Para ter mais informações, consulte o texto explicativo “Importante” na página `CREATE SEQUENCE`.

Para obter orientações sobre a melhor forma de usar sequências com base nos padrões de workload, consulte Trabalhar com sequências e colunas de identidade.

Sintaxe compatível

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

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

Descrição

`ALTER SEQUENCE` altera os parâmetros de um gerador de sequência existente. Todos os parâmetros não definidos especificamente no comando `ALTER SEQUENCE` mantêm as configurações anteriores.

Para usar `ALTER SEQUENCE`, a sequência deve pertencer a você. Para alterar o esquema de uma sequência, também é necessário ter o privilégio `CREATE` no novo esquema. Para alterar o proprietário, você deve poder aplicar `SET ROLE` ao novo perfil de propriedade, o qual deve ter o privilégio `CREATE` no esquema da sequência. (Essas restrições determinam que a alteração do proprietário não faz nada que você não possa fazer ao remover e recriar a sequência. No entanto, em todo caso, um superusuário pode alterar a propriedade de qualquer sequência.)

Parâmetros

name

O nome (opcionalmente qualificado para o esquema) de uma sequência a ser alterada.

`IF EXISTS`

Não gerará um erro se a sequência não existir. Um aviso é emitido nesse caso.

`increment`

Qualquer cláusula `INCREMENT BY` *`increment`* é opcional. Um valor positivo criará uma sequência crescente; um valor negativo criará uma sequência decrescente. Se não for especificado, o valor de incremento antigo será mantido.

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

A cláusula opcional `MINVALUE` *`minvalue`* determina o valor mínimo que uma sequência pode gerar. Se `NO MINVALUE` for especificado, serão usados os padrões 1 e do valor mínimo do tipo de dados para sequências crescentes e decrescentes, respectivamente. Se nenhuma dessas opções for especificada, o valor mínimo atual será mantido.

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

A cláusula opcional `MAXVALUE` *`maxvalue`* determina o valor máximo da sequência. Se `NO MAXVALUE` for especificado, serão usados os padrões do valor máximo do tipo de dados e -1 para sequências crescentes e decrescentes, respectivamente. Se nenhuma dessas opções for especificada, o valor máximo atual será mantido.

`CYCLE`

A palavra-chave `CYCLE` opcional pode ser usada para permitir que a sequência seja contornada quando o *`maxvalue`* ou *`minvalue`* for atingido por uma sequência crescente ou decrescente, respectivamente. Se o limite for atingido, o próximo número gerado será o *`minvalue`* ou *`maxvalue`*, respectivamente.

`NO CYCLE`

Se a palavra-chave `NO CYCLE` opcional for especificada, qualquer chamada para `nextval` depois que a sequência atingir o valor máximo exibirá um erro. Se tanto `CYCLE` quanto `NO CYCLE` não forem especificados, o comportamento do ciclo antigo será mantido.

`start`

A cláusula opcional `START WITH` *`start`* altera o valor inicial registrado da sequência. Isso não tem efeito no valor da sequência atual; simplesmente define o valor que os futuros comandos `ALTER SEQUENCE RESTART` usarão.

reiniciar

A cláusula opcional `RESTART [ WITH restart ]` altera o valor atual da sequência. Isso é semelhante a chamar a função `setval` com `is_called = false`: o valor especificado será exibido na próxima chamada de `nextval`. Escrever `RESTART` sem valor de *reinicialização* equivale a fornecer o valor inicial que foi registrado por `CREATE SEQUENCE` ou definido pela última vez por`ALTER SEQUENCE START WITH`.

Ao contrário de uma chamada `setval`, uma operação `RESTART` em uma sequência é transacional e impede que transações simultâneas obtenham números da mesma sequência. Se esse não for o modo de operação desejado, deve-se usar `setval`.

cache

A cláusula `CACHE cache` permite a pré-alocação e o armazenamento de números de sequência na memória para que o acesso seja mais rápido. O valor deve ser 1 ou algum valor >= 65.536. Se não for especificado, o valor de cache antigo será mantido. Para ter mais informações sobre comportamentos de cache, consulte `CREATE SEQUENCE`.

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

A opção `OWNED BY` faz com que a sequência seja associada a uma coluna específica da tabela, de forma que, se essa coluna (ou a tabela inteira) for eliminada, a sequência também será automaticamente eliminada. Se especificada, essa associação substitui qualquer associação especificada anteriormente para a sequência. A tabela especificada deve ter o mesmo proprietário e estar no mesmo esquema que a sequência. A especificação `OWNED BY NONE` remove qualquer associação existente, tornando a sequência “independente”.

new_owner

O nome de usuário do novo proprietário da sequência.

new_name

O novo nome da sequência.

new_schema

O novo esquema para a sequência.

Observações

`ALTER SEQUENCE` não afetará imediatamente os resultados de `nextval` em backends que tenham valores de sequência pré-alocados (armazenados em cache), a não ser no atual. Eles usarão todos

os valores armazenados em cache antes de perceber os parâmetros de geração de sequência alterados. O backend atual será afetado imediatamente.

`ALTER SEQUENCE` não afeta o status `currval` da sequência.

`ALTER SEQUENCE` pode provocar outras transações no OCC.

Por motivos históricos, `ALTER TABLE` também pode ser usado com sequências, mas as únicas variantes de `ALTER TABLE` permitidas com sequências são equivalentes aos formulários mostrados acima.

Exemplos

Reinicie uma sequência chamada `serial`, em 105:

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

Compatibilidade

`ALTER SEQUENCE` segue o padrão SQL, exceto para as cláusulas `AS`, `START WITH`, `OWNED BY`, `OWNER TO`, `RENAME TO` e `SET SCHEMA`, que são extensões do PostgreSQL.

DROP SEQUENCE

`DROP SEQUENCE`: remove uma sequência.

Sintaxe compatível

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

Descrição

`DROP SEQUENCE`: remove geradores de números de sequência. Uma sequência só pode ser removida pelo proprietário ou um superusuário.

Parâmetros

IF EXISTS

Não gerará um erro se a sequência não existir. Um aviso é emitido nesse caso.

name

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

`CASCADE`

Descarta automaticamente os objetos que dependem da sequência e, por sua vez, todos os objetos que dependem desses objetos.

`RESTRICT`

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

Exemplos

Para remover a sequência `seq`:

```
DROP SEQUENCE seq;
```

Compatibilidade

`DROP SEQUENCE` segue o padrão SQL, com a exceção de que o padrão permite que apenas uma sequência seja eliminada por comando, e com exceção da opção `IF EXISTS`, que é uma extensão do PostgreSQL.

`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 por linha, mas essa opção ainda fará 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.

`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`.
- 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.
- 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.

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';
```

Crie uma visualização com `LOCAL CHECK OPTION`.

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

Cria uma visualização recursiva.

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

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`.) Para usar `ALTER VIEW`, a visualização deve pertencer a você. 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.

Parâmetros

`name`

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

`column_name`

Nome de uma coluna existente ou 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.

`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] [, ... ] )`

Define uma opção de visualização. As seguintes opções são oferecidas:

- `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.
- `security_invoker (boolean)`: altera a propriedade do invocador de segurança da visualização.

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

Redefine uma opção de visualização, mudando-a para o valor padrão.

Exemplos

Mudar 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.

Migrar do PostgreSQL para o Aurora DSQL

O Aurora DSQL foi projetado para ser compatível com o PostgreSQL. Ele deve comportar recursos relacionais principais, como transações ACID, índices secundários, junções e operações DML padrão. A maioria das aplicações PostgreSQL existentes pode migrar para o Aurora DSQL com o mínimo de alterações.

Esta seção fornece orientação prática para migrar sua aplicação para o Aurora DSQL, incluindo compatibilidade de frameworks, padrões de migração e considerações arquitetônicas.

Compatibilidade com frameworks e ORM

O Aurora DSQL utiliza o protocolo de comunicação padrão do PostgreSQL, garantindo a compatibilidade com drivers PostgreSQL e frameworks. Os ORMs mais conhecidos funcionam com o Aurora DSQL com o mínimo ou nenhuma alteração. Consulte the section called “Adaptadores do Aurora DSQL” para acessar implementações de referência e integrações de ORM disponíveis.

Padrões comuns de migração

Ao migrar do PostgreSQL para o Aurora DSQL, alguns recursos funcionam de forma diferente ou têm sintaxe alternativa. Esta seção fornece orientação sobre cenários comuns de migração.

Alternativas de operações DDL

O Aurora DSQL fornece alternativas modernas às operações DDL tradicionais do PostgreSQL:

Compilação de índice

Use `CREATE INDEX ASYNC` em vez de `CREATE INDEX` para criação de índices sem bloqueio.

Benefício: criação de índice de tempo de inatividade zero em tabelas grandes.

Remoção de dados

Use `DELETE FROM table_name` em vez de `TRUNCATE`.

Alternativa: para a recriação completa da tabela, use `DROP TABLE` seguido de `CREATE TABLE`.

Configuração do sistema

Como o Aurora DSQL é totalmente gerenciado, a configuração é feita automaticamente com base nos padrões de workload. Use a API ou o Console de Gerenciamento da AWS para gerenciar configurações de cluster.

Benefício: não há necessidade de ajuste do banco de dados nem gerenciamento de parâmetros.

Padrões de projeto do esquema

Adapte estes padrões comuns do PostgreSQL para compatibilidade com o Aurora DSQL:

Padrões de integração referenciais

O Aurora DSQL comporta relações e operações `JOIN` de tabelas. Para obter integridade referencial, implemente a validação na camada da aplicação. Esse design se alinha aos padrões

dos bancos de dados distribuídos modernos, nos quais a validação da camada da aplicação oferece maior flexibilidade e evita gargalos de desempenho das operações em cascata.

Padrão: implemente verificações de integridade referencial em sua camada de aplicação usando convenções de nomenclatura consistentes, lógica de validação e limites de transação. Muitas aplicações de alta escala preferem essa abordagem para melhor controle sobre o tratamento de erros e a performance.

Tratamento de dados temporários

Use CTEs, subconsultas ou tabelas regulares com lógica de limpeza em vez de tabelas temporárias.

Alternativa: crie tabelas com nomes específicos da sessão e limpe-as em sua aplicação.

Noções básicas sobre as diferenças de arquitetura

A arquitetura distribuída e sem servidor do Aurora DSQL difere intencionalmente do PostgreSQL tradicional em várias áreas. Essas diferenças possibilitam os principais benefícios de simplicidade e escala do Aurora DSQL.

Modelo de banco de dados simplificado

Único banco de dados por cluster

O Aurora DSQL fornece um banco de dados incorporado denominado `postgres` por cluster.

Dica de migração: se sua aplicação usa vários bancos de dados, crie clusters do Aurora DSQL separados para separação lógica ou use esquemas em um único cluster.

Sem tabelas temporárias

Para o tratamento temporário de dados, é NECESSÁRIO usar expressões de tabela comuns (CTEs) e subconsultas, que oferecem alternativas flexíveis para consultas complexas.

Alternativa: use CTEs com cláusulas `WITH` para conjuntos de resultados temporários ou tabelas regulares com nomenclatura exclusiva para dados específicos da sessão.

Gerenciamento automático de armazenamento

O Aurora DSQL elimina os espaços de tabela e o gerenciamento manual do armazenamento. O armazenamento escala e otimizado automaticamente com base em seus padrões de dados.

Benefício: não é necessário monitorar o espaço em disco, planejar a alocação de armazenamento nem gerenciar configurações de espaço de tabela.

Padrões de aplicação moderna

O Aurora DSQL incentiva padrões de desenvolvimento de aplicações modernas que melhoram a capacidade de manutenção e a performance:

Lógica em nível de aplicação em vez de gatilhos de banco de dados

Para ter uma funcionalidade semelhante a um acionador, implemente a lógica orientada a eventos na camada da aplicação.

Estratégia de migração: mova a lógica de gatilho para o código da aplicação, use arquiteturas orientadas por eventos com serviços da AWS, como o EventBridge, ou implemente trilhas de auditoria usando o registro em log de aplicações.

Funções SQL para processamento de dados

O Aurora DSQL comporta funções baseadas em SQL, mas não a linguagens processuais, como PL/pgSQL.

Alternativa: use funções SQL para transformações de dados ou mova uma lógica complexa para sua camada de aplicação ou funções do AWS Lambda.

Controle de simultaneidade otimista em vez de bloqueio pessimista

O Aurora DSQL usa o controle otimista (OCC), uma abordagem sem bloqueio que difere de sistemas tradicionais baseados em bloqueios. Em vez de adquirir bloqueios que impedem outras transações, o Aurora DSQL permite que as transações prossigam sem bloqueio e detecta conflitos no momento da confirmação. Isso elimina deadlocks e impede que transações lentas bloqueiem outras operações.

Diferença fundamental: quando ocorrem conflitos, o Aurora DSQL exibe um erro de serialização em vez de fazer as transações aguardarem bloqueios. Isso exige que as aplicações implementem uma lógica de repetição, semelhante à manipulação de tempos limite de bloqueio em bancos de dados tradicionais, mas os conflitos são resolvidos imediatamente, em vez de causar esperas de bloqueio.

Padrão de projeto: implemente uma lógica de transação idempotente com mecanismos de repetição. Crie esquemas para minimizar a contenção usando chaves primárias aleatórias e

distribuindo atualizações por todo o intervalo de chaves. Para obter detalhes, consulte Controle de simultaneidade no Aurora DSQL.

Relacionamentos e integridade referencial

O Aurora DSQL aceita relações de chave externa entre tabelas, como operações `JOIN` . Para obter integridade referencial, implemente a validação na camada da aplicação. Embora impor a integridade referencial possa ser valioso, operações em cascata (como exclusões em cascata) podem criar problemas de performance inesperados, por exemplo, excluir um pedido com mil itens de linha se torna uma transação de 1.001 linhas. Muitos clientes evitam restrições de chave externa por esse motivo.

Padrão de projeto: implemente verificações de integridade referencial em sua camada de aplicação, use padrões de consistência eventual ou utilize serviços da AWS para validação de dados.

Simplificações operacionais

O Aurora DSQL elimina muitas tarefas tradicionais de manutenção de banco de dados, reduzindo a sobrecarga operacional:

Não é necessária manutenção manual

O Aurora DSQL gerencia automaticamente a otimização do armazenamento, a coleta de estatísticas e o ajuste de desempenho. Os comandos de manutenção tradicionais, como `VACUUM`, são gerenciados pelo sistema.

Benefício: elimina a necessidade de janelas de manutenção do banco de dados, agendamento de vácuo e ajuste dos parâmetros do sistema.

Escalabilidade e particionamento automático

O Aurora DSQL particiona e distribui automaticamente seus dados com base nos padrões de acesso. Use UUIDs ou IDs gerados pela aplicação para ter uma distribuição ideal.

Dica de migração: remova a lógica de particionamento manual e deixe o Aurora DSQL lidar com a distribuição de dados. Use UUIDs ou IDs gerados pela aplicação para ter uma distribuição ideal. Se a aplicação exigir identificadores sequenciais, consulte Sequências e colunas de identidade.

Migração agêntica com ferramentas de IA

Os agentes de codificação de IA podem acelerar a migração para o Aurora DSQL analisando esquemas, transformando código e executando migrações DDL com verificações de segurança integradas.

Usar o Kiro para migração

Agentes de codificação, como o Kiro, podem ajudar você a analisar e migrar seu código do PostgreSQL para o Aurora DSQL:

- Análise do esquema: faça upload de seus arquivos de esquema existentes e peça ao Kiro que identifique possíveis problemas de compatibilidade e sugira alternativas.
- Transformação de código: forneça o código de sua aplicação e peça ao Kiro que ajude a refatorar a lógica de gatilho, substituir sequências por UUIDs ou modificar padrões de transação.
- Planejamento de migração: peça à Kiro que crie um plano de migração passo a passo com base em sua arquitetura de aplicação específica.
- Migrações DDL: execute modificações no esquema usando o padrão de recriação de tabela com verificações de segurança e verificação de usuário integradas.

Exemplos de prompts:

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

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

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

"Drop the legacy_status column from the orders table"

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

Migração DDL com recriação de tabela

Ao usar agentes de IA com o servidor MCP do Aurora DSQL, determinadas operações ALTER TABLE usam um padrão de recriação de tabela que migra os dados com segurança. O agente lida com a complexidade e, ao mesmo tempo, mantém você a par de cada etapa.

As seguintes operações usam o padrão de recriação de tabela:

Operation	Abordagem
`DROP COLUMN`	Excluir coluna da nova tabela
`ALTER COLUMN TYPE`	Converter o tipo de dados durante a migração
`ALTER COLUMN SET/DROP NOT NULL`	Alterar restrição na nova definição de tabela
`ALTER COLUMN SET/DROP DEFAULT`	Definir padrão na nova definição de tabela
`ADD/DROP CONSTRAINT`	Incluir ou remover restrições na nova tabela
`MODIFY PRIMARY KEY`	Definir uma nova chave primária com validação de exclusividade
Dividir/mesclar colunas	Usar SPLIT_PART, SUBSTRING ou CONCAT

As seguintes operações ALTER TABLE podem ser usadas diretamente sem precisar recriar a tabela:

- `ALTER TABLE ... RENAME COLUMN`: renomear uma coluna.
- `ALTER TABLE ... RENAME TO`: renomear uma tabela.
- `ALTER TABLE ... ADD COLUMN`: adicionar uma nova coluna.

Recursos de segurança: durante a execução de migrações DDL, os agentes de IA apresentam o plano de migração, verificam a compatibilidade dos dados, confirmam a contagem de linhas e solicitam aprovação explícita antes de realizar qualquer operação destrutiva, como DROP TABLE.

Migrações em lote: para tabelas com mais de 3.000 linhas, o agente agrupa automaticamente a migração em lote em incrementos de 500 a 1.000 linhas para respeitar os limites de transação.

Servidor MCP do Aurora DSQL

O servidor de protocolo de contexto para modelos (MCP) do Aurora DSQL permite que assistentes de IA se conectem diretamente ao cluster do Aurora DSQL e pesquisem a documentação do Aurora DSQL. Isso permite que a IA:

- Analise seu esquema existente e sugira alterações de migração.
- Execute migrações DDL com o padrão de recriação de tabela.
- Teste as consultas e verifique a compatibilidade durante a migração.
- Forneça orientações precisas e atualizadas com base na documentação mais recente do Aurora DSQL.

Para usar o servidor MCP do Aurora DSQL com assistentes de IA, consulte as instruções de configuração do servidor MCP do Aurora DSQL.

Considerações sobre o Aurora DSQL para compatibilidade com o PostgreSQL

O Aurora DSQL tem diferenças de suporte a recursos em relação ao PostgreSQL autogerenciado, que permitem sua arquitetura distribuída, operação sem servidor e escalabilidade automática. A maioria das aplicações funciona dentro dessas diferenças sem modificações.

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 integrado por cluster, chamado `postgres`. Para separação lógica, crie clusters do Aurora DSQL separados ou use esquemas em um único cluster.
- O banco de dados `postgres` usa a codificação de caracteres UTF-8, que oferece amplo suporte a caracteres internacionais.
- O banco de dados usa somente o agrupamento `C`.
- O Aurora DSQL usa `UTC` como fuso horário do sistema. O Postgres armazena todas as datas e horários com reconhecimento de fuso horário internamente em UTC. Você pode definir o parâmetro de configuração `TimeZone` para converter a forma como ele é exibido para o cliente e servir como padrão para a entrada do cliente que o servidor usará para converter em UTC internamente.
- O nível de isolamento de transação é fixo em no PostgreSQL `Repeatable Read`.
- As transações têm as seguintes restrições:
  - As operações de DDL e DML requerem transações separadas.
  - 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.
- O Aurora DSQL gerencia permissões por meio de concessões em nível de esquema. Os usuários administradores criam esquemas usando `CREATE SCHEMA` e concedem acesso usando `GRANT USAGE ON SCHEMA`. Os usuários administradores gerenciam objetos no esquema público, enquanto os usuários não administradores criam objetos em esquemas criados pelo usuário para definir limites claros de propriedade. Para obter mais informações, consulte Autorizar perfis de banco de dados a usar SQL no banco de dados.

Precisa de ajuda com a migração?

Se você encontrar recursos essenciais para sua migração, mas que atualmente não são aceitos no Aurora DSQL, consulte Fornecer feedback sobre o Amazon Aurora DSQL para acessar informações sobre como compartilhar feedback com a AWS.

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.

Sequências e colunas de identidade

Sequências e colunas de identidade geram valores de inteiro e são úteis quando identificadores compactos ou legíveis por humanos são necessários. Esses valores envolvem o comportamento de alocação e armazenamento em cache descrito na documentação `CREATE SEQUENCE`.

Tópicos

- Funções de manipulação de sequências
- Colunas de identidade
- Trabalhar com sequências e colunas de identidade

Funções de manipulação de sequências

Esta seção descreve funções para operar em objetos de sequência, também chamados de geradores de sequência ou apenas sequências. Objetos de sequência são tabelas especiais de linha única criadas com`CREATE SEQUENCE`. Eles são comumente usados para gerar identificadores exclusivos para linhas de uma tabela. As funções de sequência oferecem métodos simples e seguros para vários usuários quando se deseja obter valores de sequência sucessivos de objetos de sequência.

> **Important**
>
> Ao usar sequências, é necessário considerar cuidadosamente o valor do cache. Para ter mais informações, consulte o texto explicativo “Importante” na página `CREATE SEQUENCE`. Para obter orientações sobre a melhor forma de usar sequências com base nos padrões de workload, consulte Trabalhar com sequências e colunas de identidade.

Função	Descrição
`nextval ( regclass ) # bigint`	Avança o objeto de sequência para o próximo valor e exibe esse valor. Isso é feito atomicamente: mesmo que várias sessões executem `nextval` simultane amente, cada uma receberá com segurança um valor de sequência distinto. Se o objeto de sequência tiver sido criado com parâmetros padrão, as chamadas `nextval` sucessivas exibirão valores crescentes que começam com 1. Outros comportamentos podem ser obtidos usando os parâmetros apropriados no comando `CREATE SEQUENCE`. Essa função requer o privilégio USAGE ou UPDATE na sequência.
`setval ( regclass, bigint [, boolean ] ) # bigint`	Define o valor atual do objeto de sequência e, opcionalm ente, o respectivo sinalizador `is_called` . O formulári o de dois parâmetros define o campo `last_value` da sequência com o valor especificado e define o campo `is_called` como `true`, o que significa que o próximo `nextval` avançará a sequência antes de exibir um valor. O valor que será relatado por `currval` também é definido como o valor especificado. No formulário de três parâmetros, é possível definir `is_called` como `true` ou `false`. `true` tem o mesmo efeito que o formulário de dois parâmetros. Se estiver definido como `false`, o próximo `nextval` exibirá exatamente o valor especific ado e o avanço da sequência começará com o `nextval`

Função	Descrição

	subsequente. Além disso, o valor informado por `currval` não é alterado aqui. Por exemplo:
	
	`SELECT setval('myseq', 42); -- Next nextval will return 43`
	`SELECT setval('myseq', 42, true); -- Same as above`
	`SELECT setval('myseq', 42, false); -- Next nextval will return 42`
	
	O resultado exibido por `setval` é simplesmente o valor do segundo argumento. Essa função requer o privilégio `UPDATE` na sequência.

`currval ( regclass ) # bigint`	Exibe o valor obtido mais recentemente por `nextval` para essa sequência na sessão atual. (Será relatado um erro se `nextval` nunca tiver sido chamado para essa sequência nessa sessão.) Como está sendo exibido um valor de sessão local, essa função fornecerá uma resposta previsível independentemente de outras sessões terem executado `nextval` porque a sessão atual executou. Essa função requer o privilégio `USAGE` ou `SELECT` na sequência.
`lastval () # bigint`	Mostra o valor exibido mais recentemente por `nextval` na transação atual. Essa função é idêntica a `currval`, exceto que, em vez de usar o nome da sequência como argumento, ela se refere a qualquer sequência à qual `nextval` foi aplicado mais recentemente na transação atual. É um erro chamar `lastval` quando `nextval` ainda não foi chamado na transação atual. Essa função requer o privilégio `USAGE` ou `SELECT` na última sequência usada.

> **Warning**
>
> O valor obtido por `nextval` não será recuperado para reutilização se a transação de chamada for cancelada posteriormente. Isso significa que cancelamentos de transações ou falhas no banco de dados podem gerar intervalos na sequência dos valores atribuídos. Isso também pode acontecer sem o cancelamento de uma transação. Por exemplo, um `INSERT` com uma cláusula `ON CONFLICT` calculará a tupla a ser inserida e fará qualquer chamada `nextval` necessária antes de detectar qualquer conflito que faria com que a regra `ON CONFLICT` fosse seguida. Portanto, os objetos de sequência do Aurora DSQL não podem ser usados para obter sequências "sem intervalos".
> Da mesma forma, as alterações de estado de sequência criadas por `setval` podem vistas imediatamente por outras transações e não serão desfeitas se a transação de chamada for revertida.

A sequência a ser operada por uma função de sequência é especificada por um argumento `regclass`, que é simplesmente o identificador de objeto (OID) da sequência no catálogo de sistema `pg_class`. No entanto, não é necessário pesquisar o OID manualmente, porque o conversor de entrada do tipo de dados `regclass` fará isso para você. Consulte a documentação Object Identifier Types do PostgreSQL para ver detalhes.

Colunas de identidade

> **Important**
>
> Ao usar colunas de identidade, é necessário considerar cuidadosamente o valor do cache. Para ter mais informações, consulte o texto explicativo "Importante" na página `CREATE SEQUENCE`.
> Para obter orientações sobre a melhor forma de usar colunas de identidade com base nos padrões de workload, consulte Trabalhar com sequências e colunas de identidade.

Uma coluna de identidade é uma coluna especial gerada automaticamente com base em uma sequência implícita. Ela pode ser usado para gerar valores de chave. Para criar uma coluna de identidade, use a cláusula `GENERATED ... AS IDENTITY` em`CREATE TABLE`. Por exemplo:

```
CREATE TABLE people (
```

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

ou alternativamente:

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

Consulte CREATE TABLE para obter mais detalhes.

Se um comando INSERT for executado na tabela com a coluna de identidade e nenhum valor for especificado explicitamente para a coluna de identidade, será inserido um valor gerado pela sequência implícita. Por exemplo, com as definições anteriores e presumindo colunas adicionais apropriadas, escrever:

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

geraria valores para a coluna id a partir de 1 e os seguintes dados da tabela:

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

Também é possível especificar a palavra-chave DEFAULT, em vez de um valor, para solicitar explicitamente o valor gerado pela sequência:

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

Da mesma forma, a palavra-chave DEFAULT pode ser usada em comandos UPDATE.

Portanto, sob vários aspectos, uma coluna de identidade se comporta como uma coluna com um valor padrão.

As cláusulas ALWAYS e BY DEFAULT na definição da coluna determinam como os valores especificados pelo usuário são tratados explicitamente nos comandos INSERT e UPDATE. Em um

comando `INSERT`, se `ALWAYS` estiver selecionado, um valor especificado pelo usuário só será aceito se a instrução `INSERT` especificar`OVERRIDING SYSTEM VALUE`. Se `BY DEFAULT` estiver selecionado, o valor especificado pelo usuário terá precedência. Portanto, o uso de `BY DEFAULT` gera um comportamento mais semelhante aos valores padrão, em que o valor padrão pode ser substituído por um valor explícito, enquanto `ALWAYS` oferece um pouco mais de proteção contra a inserção acidental de um valor explícito.

O tipo de dados de uma coluna de identidade deve ser um dos tipos de dados aceitos pelas sequências. (Consulte `CREATE SEQUENCE`.) As propriedades da sequência associada podem ser especificadas ao criar uma coluna de identidade (consulte `CREATE TABLE`) ou alteradas posteriormente (consulte `ALTER TABLE`).

Uma coluna de identidade é marcada automaticamente como `NOT NULL`. No entanto, uma coluna de identidade não garante exclusividade. (Uma sequência normalmente exibe valores exclusivos, mas ela pode ser redefinida ou os valores podem ser inseridos manualmente na coluna de identidade, conforme discutido anteriormente.) A exclusividade precisaria ser aplicada usando uma restrição `PRIMARY KEY` ou `UNIQUE`.

Trabalhar com sequências e colunas de identidade

Esta seção ajuda a entender a melhor forma de usar sequências e colunas de identidade com base nos padrões de workload.

> **Important**
>
> Consulte o texto explicativo “Importante” na página `CREATE SEQUENCE` para ver mais detalhes sobre alocação e comportamento de armazenamento em cache.

Escolher tipos de identificador

O Amazon Aurora DSQL permite identificadores baseados em UUID e valores inteiros gerados por meio de sequências ou colunas de identidade. Essas opções diferem com relação à forma como os valores são alocados e à forma como são escalados sob carga.

Os valores de UUID podem ser gerados sem coordenação e são adequados para workloads em que são criados identificadores com frequência ou em várias sessões. Como o Amazon Aurora DSQL foi projetado para operação distribuída, evitar a coordenação geralmente é útil. Por esse motivo, os

UUIDs são recomendados como o tipo de identificador padrão, especialmente para chaves primárias em workloads em que a escalabilidade é importante e a ordenação estrita dos identificadores não é necessária.

Sequências e colunas de identidade geram valores inteiros compactos que são convenientes para identificadores legíveis por humanos, geração de relatórios e interfaces externas. Quando os identificadores numéricos são preferidos por motivos de usabilidade ou integração, considere a possibilidade de usar concomitantemente uma sequência ou coluna de identidade e identificadores baseados em UUID. Quando a sequência de inteiros ou os valores de identidade são necessários, a escolha de um tamanho de cache apropriado se torna uma parte essencial do design da workload. Consulte a seção a seguir para obter orientações sobre como escolher um tamanho de cache.

Escolher o tamanho do cache

A seleção de um valor de cache apropriado é essencial para o uso eficaz de sequências e colunas de identidade. A configuração do cache determina como a alocação de identificadores se comporta sob carga, influenciando o throughput do sistema e até que ponto os valores retratam a ordem de alocação.

Um tamanho de cache maior, como **`CACHE >= 65536`**, é adequado quando:

- Os identificadores são gerados em alta frequência.
- Várias sessões são inseridas simultaneamente.
- A workload consegue tolerar intervalos e efeitos de ordenação visíveis.

Por exemplo, workloads de ingestão de eventos de alto volume (como IoT ou telemetria), bem como identificadores operacionais, como IDs de execução de tarefas, referências de casos de suporte ou números de pedidos internos, geralmente se beneficiam de tamanhos de cache maiores, em que os identificadores são gerados com frequência e a ordenação estrita não é necessária.

Um tamanho de cache de 1 é mais adequado quando:

- As taxas de alocação são relativamente baixas.
- Espera-se que os identificadores sigam a ordem de alocação de maneira mais estreita ao longo do tempo.
- A minimização dos intervalos é mais importante do que o throughput máximo.

Determinadas workloads, como atribuição de números de conta ou de referência, nas quais são gerados identificadores com menor frequência e uma ordenação mais estreita é desejável, se alinham melhor com um tamanho de cache de 1.

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 where job_id = 'wqhu6ewifze5xitg3umt24h5ua';
```

O Aurora DSQL exibe uma resposta semelhante à seguinte:

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

A coluna de status pode ser um dos seguintes valores:

Status	Descrição
`submitted`	A tarefa foi enviada, mas o Aurora DSQL ainda não começou a processá-la.
`processing`	O Aurora DSQL está processando a tarefa.
`failed`	A tarefa falhou. Consulte os detalhes da coluna para ter mais informaçõ es. 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`.
`completed`	O Aurora DSQL concluiu a tarefa com êxito.

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, bem como consultas úteis para encontrar informações sobre o sistema, como a versão.

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.

Name (Nome)	Descrição
`pg_namespace`	Informações sobre todos os esquemas
`pg_tables`	Informações sobre todas as tabelas

Name (Nome)	Descrição
`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.

Nome	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

Nome	Aplicável ao Aurora DSQL
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
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

Nome	Aplicável ao Aurora DSQL
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
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

Nome	Aplicável ao Aurora DSQL
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
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.

Nome	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
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

Nome	Aplicável ao Aurora DSQL
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
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

Nome	Aplicável ao Aurora DSQL
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
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

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

A visualização 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;
```

O Aurora DSQL exibe uma resposta semelhante à seguinte:

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

A tabela a seguir descreve as colunas na visualização `sys.jobs`.

Colunas na visualização sys.jobs

Coluna	Tipo	Descrição
`job_id`	`text`	Um UUID de base 32 que representa o trabalho.
`status`	`text`	O status atual do trabalho. Os valores possíveis são `submitted` , `processing` , `completed` e `failed`. Para obter mais informações, consulte Valores de status de sys.jobs.
`details`	`text`	Qualquer informação relevante sobre o trabalho. Um motivo detalhado é fornecido se o trabalho falha.
`job_type`	`text`	O tipo de trabalho assíncrono. Os valores possíveis são: `INDEX_BUILD` : uma construção de índice assíncrona. `ANALYZE`: um trabalho de análise automática enviado pelo sistema. `DROP`: remove dados físicos após uma operação `DROP TABLE` ou `DROP INDEX`.
`class_id`	`oid`	O OID da tabela de catálogo que contém o objeto.
`object_id`	`oid`	O OID do objeto.
`object_name`	`text`	O nome totalmente qualificado do atuador. Os trabalhos `DROP` não podem fazer referência a

Coluna	Tipo	Descrição
		objetos já descartados. Se um objeto referenciado já tiver sido descartado, `object_name` pode ser NULL.
`start_time`	`timestamp with time zone`	A data e hora em que o trabalho foi enviado.
`update_time`	`timestamp with time zone`	A data e hora em que o trabalho foi atualizado pela última vez.

Valores de status de sys.jobs

Status	Descrição
`submitted`	A tarefa foi enviada, mas o Aurora DSQL ainda não começou a processá-la.
`processing`	O Aurora DSQL está processando a tarefa.
`failed`	A tarefa falhou. Consulte a coluna `details` para obter mais informaçõ es.
`completed`	O Aurora DSQL concluiu a tarefa com êxito.

A visualização 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;
```

Consultas úteis de metadados do sistema

Use essas consultas para acessar estatísticas da tabela e metadados do sistema sem realizar operações caras, como verificações completas da tabela.

Apurar a contagem estimada de linhas de uma tabela

Para apurar a contagem aproximada de linhas em uma tabela sem realizar uma verificação completa, use a seguinte consulta:

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

Esse comando retorna uma saída semelhante à seguinte:

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

Essa abordagem é mais eficiente do que `SELECT COUNT(*)` para tabelas grandes no Aurora DSQL.

Obter a versão principal atual do Aurora DSQL

Para obter a versão principal atual do cluster do Aurora DSQL, use a seguinte consulta:

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

Esse comando retorna uma saída semelhante à seguinte:

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

Isso retorna a versão principal em que a conexão SQL está ativada no Aurora DSQL.

Obter a versão atual do PostgreSQL

Para obter a versão atual do PostgreSQL do cluster do Aurora DSQL, use a seguinte consulta:

```
SHOW server_version;
```

Esse comando retorna uma saída semelhante à seguinte:

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

Isso retorna a versão do PostgreSQL em que a conexão SQL está ativada no Aurora DSQL.

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.

Trabalhar com os planos EXPLAIN do Aurora DSQL

O Aurora DSQL usa uma estrutura de plano EXPLAIN semelhante à do PostgreSQL, mas com adições importantes que refletem a respectiva arquitetura distribuída e modelo de execução.

Nesta documentação, forneceremos uma visão geral dos planos EXPLAIN do Aurora DSQL, destacando as semelhanças e diferenças em comparação ao PostgreSQL. Abordaremos os vários tipos de operações de verificação disponíveis no Aurora DSQL e ajudaremos você a entender o custo da realização de suas consultas.

Planos EXPLAIN do PostgreSQL versus Aurora DSQL

O Aurora SQL é desenvolvido com base no banco de dados PostgreSQL e compartilha a maioria das estruturas de planos com o PostgreSQL, mas tem diferenças arquitetônicas importantes que afetam a realização e a otimização de consultas:

Recurso	PostgreSQL	Aurora DSQL
Armazenamento de dados	Armazenamento heap	Sem heap, todas as linhas são indexadas por um identificador exclusivo.
Chave primária	O índice da chave primária é separado dos dados da tabela.	O índice da chave primária é a tabela com todas as colunas extras como colunas INCLUDE.
Índices secundários	Índices secundários padrão.	Funciona da mesma forma que o PostgreSQL, com a capacidade de incluir colunas que não sejam chave.
Recursos de filtragem	Condição de índice, filtro heap.	Condição de índice, filtro de armazenamento, filtro do processador de consultas.

Recurso	PostgreSQL	Aurora DSQL
Tipos de varredura	Verificação sequencial, verificação de índice, Verificação somente de índice.	Verificação sequencial, verificação de índice, Verificação somente de índice.
Realização de consultas	Local para o banco de dados.	Distribuído (computação e armazenamento são separados).

O Aurora DSQL armazena os dados da tabela diretamente na ordem da chave primária, em vez de em um heap separado. Cada linha é identificada por uma chave exclusiva, normalmente a chave primária, que permite que o banco de dados otimize as pesquisas com maior eficiência. A diferença arquitetônica explica por que o Aurora DSQL geralmente usa verificações somente de índice nos casos em que o PostgreSQL pode escolher a verificação sequencial.

Outra distinção importante é que o Aurora DSQL separa a computação do armazenamento, permitindo que os filtros sejam aplicados mais cedo no caminho de execução a fim de reduzir a movimentação de dados e melhorar a performance.

Para saber mais sobre o uso dos planos EXPLAIN com o PostgreSQL, consulte a documentação sobre o EXPLAIN do PostgreSQL.

Elementos-chave nos planos EXPLAIN do Aurora DSQL

Os planos EXPLAIN do Aurora DSQL fornecem informações detalhadas sobre como as consultas são realizadas, incluindo onde a filtragem ocorre e quais colunas são recuperadas do armazenamento. A compreensão dessa saída ajuda a otimizar a performance da consulta.

Condição de índice

> Condições usadas para navegar no índice. A filtragem mais eficiente que reduz os dados verificados. No Aurora DSQL, as condições de índice podem ser aplicadas em várias camadas do plano de execução.

Projeções

> Colunas recuperadas do armazenamento. Um menor número de projeções significa melhor performance.

Filtros de armazenamento

Condições aplicadas em nível de armazenamento. Mais eficiente do que os filtros do processador de consultas.

Filtros do processador de consultas

Condições aplicadas em nível de processador de consultas. Requer a transferência de todos os dados antes da filtragem, o que causa maior movimentação de dados e sobrecarga de processamento.

Filtros no Aurora DSQL

O Aurora DSQL separa a computação do armazenamento, o que significa que o ponto em que os filtros são aplicados durante a realização da consulta tem um impacto significativo na performance. Os filtros aplicados antes da transferência de grandes volumes de dados reduzem a latência e melhoram a eficiência. Quanto mais cedo um filtro for aplicado, menos dados precisarão ser processados, movidos e verificados, gerando consultas mais rápidas.

O Aurora DSQL pode aplicar filtros em vários estágios no caminho da consulta. Compreender esses estágios é fundamental para interpretar os planos de consulta e otimizar a performance.

Nível	Tipo de filtro	Descrição
1	Condição de índice	Aplicado durante a verificação do índice. Limita o volume de dados lidos do armazenamento e reduz os dados enviados à camada de computação.
2	Filtros de armazenamento	Aplicado depois que os dados são lidos do armazenamento, mas antes de serem enviados à computação. Um exemplo aqui é um filtro em uma coluna de inclusão de um índice. Reduz a transferência de dados, mas não a quantidade lida.
3	Filtros do processador de consultas	Aplicado após os dados chegarem à camada computacional. Todos os dados devem ser transferidos primeiro, o que aumenta a latência e o custo. Atualmente, o Aurora DSQL não pode realizar todas as operações de filtragem e projeção no

Nível	Tipo de filtro	Descrição
		armazenamento, portanto, algumas consultas podem ser forçadas a recorrer a esse tipo de filtragem.

Ler os planos EXPLAIN do Aurora DSQL

Entender como ler os planos EXPLAIN é fundamental para otimizar a performance das consultas. Nesta seção, vamos examinar exemplos reais de planos de consulta do Aurora DSQL, mostrar como os diferentes tipos de verificação se comportam, explicar onde os filtros são aplicados e destacar as oportunidades de otimização.

Tabelas de amostra usadas nesses exemplos

Os exemplos abaixo fazem referência a duas tabelas: `transaction` e `account`.

A tabela `transaction` não tem uma chave primária, o que faz com que o Aurora DSQL realize varreduras completas da tabela ao consultá-la.

A tabela `account` tem um índice no `customer_id`. Esse índice inclui `balance` e `status` como colunas de cobertura, o que permite que determinadas consultas sejam atendidas diretamente do índice sem serem lidas na tabela base. No entanto, o índice não inclui `created_at`, portanto, as consultas que fazem referência a essa coluna exigem acesso adicional à tabela.

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

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

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

Exemplo de verificação completa

O Aurora DSQL tem tanto verificações sequenciais, que são funcionalmente idênticas ao PostgreSQL, quanto verificações completas. A única diferença entre elas é que as verificações completas podem utilizar filtragem extra no armazenamento. Por isso, quase sempre ela é selecionada em detrimento de verificações sequenciais. Devido à semelhança, abordaremos apenas exemplos das verificações completas mais interessantes.

As verificações completas serão usadas principalmente em tabelas sem chave primária. Como as chaves primárias do Aurora DSQL são, por padrão, índices de cobertura completa, o Aurora DSQL provavelmente usará verificações somente de índice na chave primária em muitas situações em que o PostgreSQL usaria uma verificação sequencial. Como acontece com a maioria dos outros bancos de dados, uma tabela sem índices será mal escalada.

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

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

Esse plano mostra dois filtros aplicados em diferentes estágios. A condição `transaction_date > '2025-01-01'` é aplicada na camada de armazenamento, reduzindo o volume de dados exibidos. A condição `description LIKE '%external%'` é aplicada posteriormente no processador de consultas, após a transferência dos dados, tornando-a menos eficiente. Colocar filtros mais seletivos nas camadas de armazenamento ou índice geralmente melhora a performance.

Exemplo de verificação somente de índice

As verificações somente de índice são os melhores tipos de verificação no Aurora DSQL, pois geram o menor número de idas e vindas até a camada de armazenamento e podem realizar a maior parte da filtragem. Mas só porque você vê “verificação somente de índice” não significa que tenha o melhor

plano. Por conta de todos os diferentes níveis de filtragem que podem ocorrer, é essencial ainda prestar atenção aos diferentes locais em que a filtragem pode acontecer.

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

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

Nesse plano, a condição de índice, `customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'`, é avaliada primeiro durante a verificação do índice, que é o estágio mais eficiente porque limita o volume de dados lidos do armazenamento. O filtro de armazenamento, `status = 'pending'`, é aplicado após a leitura dos dados, mas antes de serem enviados para a camada computacional, reduzindo o volume de dados transferidos. Por fim, o filtro do processador de consultas, `balance > 100`, é executado por último, após a movimentação dos dados, tornando-o o menos eficiente. Dessas, a condição de índice oferece a melhor performance porque controla diretamente o volume de dados verificados.

Exemplo de verificação de índice

As verificações de índice são semelhantes às verificações somente de índice, exceto que elas têm a etapa extra de exigir o acesso à tabela base. Como o Aurora DSQL pode especificar filtros de armazenamento, ele pode fazer isso tanto na chamada de índice quanto na chamada de pesquisa.

Para deixar isso claro, o Aurora DSQL apresenta o plano como dois nós. Dessa forma, é possível ver claramente o quanto a adição de uma coluna de inclusão ajudará em termos de linhas exibidas do armazenamento.

```
EXPLAIN SELECT balance FROM account
```

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

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

Esse plano mostra como a filtragem ocorre em vários estágios:

- A condição de índice em `customer_id` filtra os dados com antecedência.
- O filtro de armazenamento em `status` restringe ainda mais os resultados antes que eles sejam enviados para computação.
- O filtro do processador de consultas em `balance` é aplicado posteriormente, após a transferência.
- O filtro de pesquisa em `created_at` é avaliado ao buscar colunas adicionais da tabela base.

Adicionar colunas usadas com frequência como campos `INCLUDE` geralmente pode eliminar essa pesquisa e melhorar a performance.

Práticas recomendadas

- Alinhe os filtros com as colunas indexadas para enviar a filtragem mais cedo.
- Use as colunas INCLUDE para permitir verificações somente de índice e evitar pesquisas.
- Valide as estimativas de linha ao investigar problemas de desempenho. O Aurora DSQL gerencia estatísticas automaticamente executando `ANALYZE` em segundo plano com base nas taxas de alteração de dados. Se as estimativas parecerem imprecisas, você pode executar `ANALYZE` manualmente para atualizar as estatísticas imediatamente.

- Evite consultas não indexadas em tabelas grandes para evitar verificações completas dispendiosas.

Noções básicas sobre DPUs em EXPLAIN ANALYZE

O Aurora DSQL fornece informações da Unidade de Processamento Distribuído (DPU) em nível de instrução na saída do plano `EXPLAIN ANALYZE VERBOSE`, oferecendo maior visibilidade do custo da consulta durante o desenvolvimento. Esta seção explica o que são DPUs e como interpretá-las na saída `EXPLAIN ANALYZE VERBOSE`.

O que é DPU?

Unidade de Processamento Distribuído (DPU) é a medida normalizada do trabalho realizado pelo Aurora DSQL. É composta por:

- ComputeDPU: tempo gasto na execução de consultas SQL.
- ReadDPU: recursos usados para ler dados do armazenamento.
- WriteDPU: recursos usados para gravar dados no armazenamento.
- MultiRegionWriteDPU: recursos usados para replicar gravações em clusters emparelhados em configurações multirregionais.

Uso de DPU em EXPLAIN ANALYZE VERBOSE

O Aurora DSQL estende `EXPLAIN ANALYZE VERBOSE` para incluir uma estimativa de uso da DPU em nível de instrução no final da saída. Isso oferece visibilidade imediata do custo da consulta, ajudando você a identificar os fatores de custo da workload, ajustar a performance da consulta e prever melhor o uso dos recursos.

Os exemplos a seguir mostram como interpretar as estimativas de DPU em nível de instrução incluídas na saída de EXPLAIN ANALYZE VERBOSE.

Exemplo 1: consulta SELECT

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

```
QUERY PLAN
------------------------------------------------
```

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

Neste exemplo, a instrução SELECT executa uma verificação somente de índice, então a maior parte do custo vem da DPU de leitura (0,04312), representando os dados recuperados do armazenamento e da DPU de computação (0,01607), o que reflete os recursos computacionais utilizados para processar e exibir os resultados. Não há DPU de gravação, pois a consulta não modifica os dados. A DPU total (0,05919) é a soma de computação + leitura + gravação.

Exemplo 2: consulta INSERT

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

```
QUERY PLAN
-----------------------------------------------------
Insert on public.test_table  (cost=0.00..0.04 rows=0 width=0) (actual time=0.055..0.056
 rows=0 loops=1)
  ->  Values Scan on "*VALUES*"  (cost=0.00..0.04 rows=3 width=122) (actual
 time=0.003..0.008 rows=3 loops=1)
       Output: "*VALUES*".column1, "*VALUES*".column2
Query Identifier: jtkjkexhjotbo
Planning Time: 0.068 ms
Execution Time: 0.543 ms
Statement DPU Estimate:
  Compute: 0.01550 DPU
  Read: 0.00307 DPU (Transaction minimum: 0.00375)
  Write: 0.01875 DPU (Transaction minimum: 0.05000)
```

```
  Total: 0.03732 DPU
```

Essa instrução executa principalmente gravações, portanto, a maior parte do custo está associada à DPU de gravação. A DPU de computação (0,01550) representa o trabalho realizado para processar e inserir os valores. A DPU de leitura (0,00307) reflete leituras secundárias do sistema (para pesquisas de catálogos ou verificações de índice).

Observe os mínimos de transação mostrados ao lado de DPUs de leitura e gravação. Eles indicam os custos básicos por transação que se aplicam somente quando a operação inclui leituras ou gravações. Eles não significam que cada transação gera automaticamente em uma cobrança de 0,00375 de DPU de leitura ou 0,05 de DPU de gravação. Em vez disso, esses mínimos são aplicados em nível de transação durante a agregação de custos e somente se as leituras ou gravações ocorrerem dentro dessa transação. Devido a essa diferença no escopo, as estimativas em nível de instrução em `EXPLAIN ANALYZE VERBOSE` podem não corresponder exatamente às métricas em nível de transação relatadas no CloudWatch ou nos dados de faturamento.

Usar informações de DPU para otimização

As estimativas de DPU por instrução oferecem uma maneira poderosa de otimizar as consultas além do tempo de execução. Entre os casos de uso comuns estão:

- Consciência de custos: entenda o quanto uma consulta é cara em relação a outras.
- Otimização do esquema: compare o impacto dos índices ou das alterações do esquema na performance e na eficiência dos recursos.
- Planejamento de orçamento: estime o custo da workload com base no uso observado da DPU.
- Comparação de consultas: avalie abordagens de consulta alternativas de acordo com seu consumo relativo de DPU.

Interpretar informações de DPU

Lembre-se das seguintes práticas recomendadas ao usar dados de DPU de `EXPLAIN ANALYZE VERBOSE`:

- Use de forma direcionada: trate a DPU relatada como uma forma de entender o custo relativo de uma consulta, em vez de uma correspondência exata com as métricas ou os dados de faturamento do CloudWatch. As diferenças são esperadas porque `EXPLAIN ANALYZE VERBOSE` indica o custo em nível de instrução, enquanto o CloudWatch agrega atividades em nível de transação. O CloudWatch também inclui operações em segundo plano (como ANALYZE ou

compactações) e sobrecarga de transação (`BEGIN`/`COMMIT`) que `EXPLAIN ANALYZE VERBOSE` exclui intencionalmente.

- A variabilidade da DPU entre as execuções é normal em sistemas distribuídos e não indica erros. Fatores, como armazenamento em cache, alterações no plano de execução, simultaneidade ou mudanças na distribuição de dados, podem fazer com que a mesma consulta consuma recursos diferentes de uma execução para outra.
- Operações pequenas em lotes: se sua workload emitir muitas instruções pequenas, pense em agrupá-las em operações maiores (que não excedam 10 MB). Isso reduz a sobrecarga de arredondamento e produz estimativas de custo mais significativas.
- Use para ajuste, não para cobrança: os dados em `EXPLAIN ANALYZE VERBOSE`foram projetados para reconhecimento de custos, ajuste de consultas e otimização. Não é uma métrica de faturamento. Sempre confie nas métricas do CloudWatch ou nos relatórios de faturamento mensais para acessar dados confiáveis de custo e uso.

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

- Configurar clusters de região única
- Configurar clusters multirregionais
- Configurar clusters do Aurora DSQL usando o AWS CloudFormation
- Ciclo de vida do cluster do Aurora DSQL

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

Configure e gerencie clusters para uma Região da AWS usando a AWS CLI ou a linguagem de programação de sua preferência, como Python, C++, JavaScript, Java, Rust, Ruby, .NET e Golang. A AWS CLI oferece acesso rápido por meio de comandos shell, enquanto os kits de desenvolvimento de software (SDKs) da AWS permitem o controle programático por meio do suporte à linguagem nativa.

Tópicos

- Usar SDKs da AWS
- Usar a AWS CLI

Usar SDKs da AWS

Os SDKs da AWS oferecem acesso programático ao Aurora DSQL na linguagem de programação de sua preferência. As seções a seguir mostram como realizar operações comuns de cluster usando diferentes linguagens de programação.

Criar cluster

Os exemplos a seguir mostram como criar um cluster de região única usando diferentes linguagens de programação.

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()
```

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;
}
```

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();
```

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);
        }
    }
}
```

Rust

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

```
use aws_config::{BehaviorVersion, Region, load_defaults};
```

```
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    let region_provider = Region::new(region);

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

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

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;

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

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

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

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

```
        .await
        .unwrap();

    cluster_output
}

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

    let cluster = create_cluster(region).await;

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

    Ok(())
}
```

Ruby

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

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

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

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

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

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

def main
  region = "us-east-1"

  cluster = create_cluster(region)

  puts "Created single region cluster:"
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

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

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

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

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

        /// <summary>
        /// Create a cluster with deletion protection enabled and a name tag.
        /// </summary>
        public static async Task<CreateClusterResponse> Create(RegionEndpoint
 region)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp single region cluster" }
                };

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

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

                return cluster;
            }
        }

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

            var cluster = await Create(region);

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

Golang

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

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

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

 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

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

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

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

 // Create the waiter with our custom options
 waiter := dsql.NewClusterActiveWaiter(client, func(o
 *dsql.ClusterActiveWaiterOptions) {
```

```
  o.MaxDelay = 30 * time.Second
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })

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

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

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

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

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

}
```

Obter um cluster

Os exemplos a seguir mostram como obter informações sobre um cluster de região única usando diferentes linguagens de programação.

Python

Para ter informações sobre um cluster de região única, 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 obter informações sobre um cluster de região única.

```
#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, 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.

```
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.

```
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.

```
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.

```
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.

```
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)
 }
}
```

Atualização do cluster

Os exemplos a seguir mostram como atualizar um cluster de região única usando diferentes linguagens de programação.

Python

Para atualizar um cluster de região única, 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 de região única.

```
#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 de região única, 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.

```
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.

```
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.

```
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.

```
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.

```
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 cluster

Os exemplos a seguir mostram como excluir um cluster de região única usando diferentes linguagens de programação.

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()
```

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;
}
```

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();
```

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);
        }
    }
}
```

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(())
}
```

Ruby

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

```
require "aws-sdk-dsql"

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

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

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

main if $PROGRAM_NAME == __FILE__
```

.NET

Para 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);
        }
    }
}
```

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 ver mais exemplos e amostras de código, acesse o repositório do GitHub de exemplos do Aurora DSQL.

Usar a AWS CLI

A AWS CLI oferece uma interface de linha de comandos para gerenciar clusters do Aurora DSQL. Os exemplos a seguir demonstram operações comuns de gerenciamento de cluster.

Criar 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

Configure e gerencie clusters em várias Regiões da AWS usando a AWS CLI ou a linguagem de programação de sua preferência, como Python, C++, JavaScript, Java, Rust, Ruby, .NET e Golang. A AWS CLI oferece acesso rápido por meio de comandos shell, enquanto os SDKs da AWS permitem o controle programático por meio do suporte à linguagem nativa.

Tópicos

- Usar SDKs da AWS
- Usar a AWS CLI

Usar SDKs da AWS

Os SDKs da AWS oferecem acesso programático ao Aurora DSQL na linguagem de programação de sua preferência. As seções a seguir mostram como realizar operações comuns de cluster usando diferentes linguagens de programação.

Criar cluster

Os exemplos a seguir mostram como criar um cluster multirregional usando diferentes linguagens de programação.

Python

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

        return (cluster_1, cluster_2)

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


def main():
    region_1 = "us-east-1"
    region_2 = "us-east-2"
    witness_region = "us-west-2"
    (cluster_1, cluster_2) = create_multi_region_clusters(region_1, region_2,
 witness_region)
    print("Created multi region clusters:")
    print("Cluster id: " + cluster_1['arn'])
    print("Cluster id: " + cluster_2['arn'])
```

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

C++

Para 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);

    std::cout << "Creating cluster in " << region1 << std::endl;
```

```
    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

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

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

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

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

    // Create second cluster
    std::cout << "Creating cluster in " << region2 << std::endl;

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

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

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

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

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

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

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

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

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

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

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

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

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

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

```
            Aws::String region2 = "us-east-2";
            Aws::String witnessRegion = "us-west-2";

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

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

JavaScript

Para 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 updateClusterCommand = new UpdateClusterCommand({
            identifier: response1.identifier,
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response2.arn]
            }
        });
        await client1.send(updateClusterCommand);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters they'll
begin the transition to ACTIVE
        console.log(`Waiting for cluster ${response1.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);
```

```
        console.log(`Waiting for cluster ${response2.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

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

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Para 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

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

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__
```

Golang

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)
```

```
 }

}
```

.NET

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}");
        }
    }
}
```

Obter um cluster

Os exemplos a seguir mostram como obter informações sobre um cluster multirregional usando diferentes linguagens de programação.

Python

Para ter informações sobre um cluster 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 exemple a seguir para obter informações sobre um cluster 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 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 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 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 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 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 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)
 }
}
```

Atualização do cluster

Os exemplos a seguir mostram como atualizar um cluster multirregional usando diferentes linguagens de programação.

Python

Para atualizar um cluster 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 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 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 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 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 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 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 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 cluster

Os exemplos a seguir mostram como excluir um cluster multirregional usando diferentes linguagens de programação.

Python

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

```
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 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 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 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 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 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 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 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)
 }
}
```

Para ver mais exemplos e amostras de código, acesse o repositório do GitHub de exemplos do Aurora DSQL.

Usar a AWS CLI

A AWS CLI oferece uma interface de linha de comandos para gerenciar clusters multirregionais do Aurora DSQL. Os exemplos a seguir demonstram como criar, configurar e excluir clusters multirregionais.

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 do Aurora DSQL usando o AWS CloudFormation

Você pode usar o mesmo recurso `AWS::DSQL::Cluster` do 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
```

Ciclo de vida do cluster do Aurora DSQL

Entender o ciclo de vida do cluster do Aurora DSQL ajuda você a gerenciar seus clusters com eficiência. Esta seção aborda as definições de status do cluster e o recurso de ajuste de escala para zero para otimização de custos.

Definir 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 Console de gerenciamento da AWS, a AWS CLI ou a API 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.
Ocioso	Um cluster fica ocioso durante o tempo necessário para o Aurora DSQL reduzir a escala verticalmente dos recursos em execução a fim de reduzir a capacidade e os custos. Quando você se conecta a um cluster ocioso, o Aurora DSQL faz a transição do cluster de volta para o estado Ativo.
Inactive	Um cluster fica ocioso quando não há atividade no cluster por um período prolongado. Nesse estado suspenso, os recursos em execução são escalados para zero enquanto seus dados são preservados. Quando você tenta se conectar a um cluster inativo, o Aurora DSQL faz a transição automática do

Status	Descrição
	cluster de volta para o estado Ativo. O tempo de restauração depende do tamanho do cluster.
Atualização	Um cluster faz a transição para o status Atualizando quando são feitas alterações na respectiva configuração.
Excluindo	Um cluster faz a transição para o status Excluindo quando é enviada uma solicitação para excluí-lo.
Excluído	O cluster foi excluído com êxito.
Failed	O Aurora DSQL não conseguiu 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.

Trabalhar com clusters ociosos e inativos

Quando o Aurora DSQL não detecta nenhuma atividade de conexão em um cluster por algum período, ele faz a transição do cluster para o estado ocioso, reduzindo os recursos em execução a fim de minimizar a capacidade e os custos. Se a atividade de conexão permanecer ausente por um período prolongado, o cluster ocioso passa automaticamente para o estado Inativo, em que os recursos em execução são escalados para zero enquanto seus dados são preservados.

Para retomar as operações normais, basta conectar-se ao cluster normalmente. Quando você se conecta com êxito ao cluster, o Aurora DSQL faz a transição automática do cluster para o estado Ativo.

> **Note**
>
> A primeira tentativa de conexão com um cluster ocioso ou inativo será mais lenta do que o normal.

Operações que exigem o estado Ativo do cluster

Algumas operações exigem que seu cluster esteja em um estado Ativo. Para realizar essas operações em um cluster ocioso ou inativo, você precisa mudar o estado do cluster de volta para ativo conectando-se ao cluster.

Operações de backup

Para fazer um backup, é necessário que o estado do cluster seja ativo. Se o cluster estiver ocioso ou inativo, os backups falharão com o seguinte erro:

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

Para continuar com um backup:

1. Conecte-se ao cluster usando seu cliente de banco de dados preferido ou o console do Aurora DSQL para ativá-lo.
2. Aguarde a transição automática para o estado Ativo.
3. Inicie o backup depois que o cluster estiver totalmente operacional.

> **Note**
>
> Os backups existentes feitos antes de o cluster tornar-se ocioso ou inativo permanecem válidos e inalterados. Novas tentativas de backup no cluster falharão até que o cluster seja conectado para ativação automática.

Visualizar o status do cluster do Aurora DSQL

Para visualizar o status de um cluster, use o Console de gerenciamento da AWS, a AWS CLI ou a API do Aurora DSQL.

Console

Siga estas etapas para visualizar o status de um cluster no Console de gerenciamento da AWS:

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
```

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.

CloudFormation

O AWS::DSQL::Cluster é um recurso do CloudFormation que permite criar e gerenciar clusters do Aurora DSQL como parte de sua infraestrutura como código. O 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 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.

Conectores para o Aurora DSQL

O Aurora DSQL oferece conectores especializados que estendem os drivers de banco de dados existentes para que a autenticação do IAM e a integração com serviços da AWS sejam consistentes.

Esses conectores são projetados para funcionar com linguagens e frameworks de programação conhecidos, mantendo a compatibilidade com os fluxos de trabalho existentes do PostgreSQL.

Conectores adicionais estão planejados para versões futuras. Para ver as informações mais recentes sobre a disponibilidade de conectores, consulte o repositório de exemplos do Aurora DSQL.

Conectar-se a clusters do Aurora DSQL com um conector JDBC

O conector do Aurora DSQL para JDBC foi projetado como um plug-in de autenticação que estende a funcionalidade do driver JDBC do PostgreSQL para permitir que as aplicações se autentiquem com o Aurora DSQL usando credenciais do IAM. O conector não se conecta diretamente ao banco de dados, mas oferece uma autenticação perfeita do IAM no driver JDBC subjacente do PostgreSQL.

O conector do Aurora DSQL para JDBC foi criado para funcionar com o driver JDBC do PostgreSQL e oferece uma integração perfeita com os requisitos de autenticação do IAM do Aurora DSQL.

Em conjunto com o driver JDBC do PostgreSQL, o conector do Aurora DSQL para JDBC permite a autenticação baseada no IAM para o Aurora DSQL. Ele possibilita uma profunda integração com serviços de autenticação da AWS, como o AWS Identity and Access Management (IAM).

Sobre o conector

O Aurora DSQL é um serviço de banco de dados SQL distribuído que oferece alta disponibilidade e escalabilidade para aplicações compatíveis com o PostgreSQL. O Aurora DSQL exige autenticação baseada no IAM com tokens de tempo limitado para os quais os drivers JDBC não oferecem suporte nativo.

A ideia principal por trás do conector do Aurora DSQL para JDBC é adicionar uma camada de autenticação sobre o driver JDBC do PostgreSQL que gerencie a geração de tokens do IAM, permitindo que os usuários se conectem ao Aurora DSQL sem alterar seus fluxos de trabalho JDBC existentes.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de tokens: os tokens de autenticação são gerados usando credenciais da AWS e têm vida útil configurável

O conector do Aurora DSQL para JDBC foi projetado para entender esses requisitos e gerar tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Benefícios do conector do Aurora DSQL para JDBC

Embora o Aurora DSQL ofereça uma interface compatível com o PostgreSQL, os drivers existentes do PostgreSQL no momento não atendem aos requisitos de autenticação do IAM do Aurora DSQL. O conector do Aurora DSQL para JDBC permite que os clientes continuem usando seus fluxos de trabalho existentes do PostgreSQL e, ao mesmo tempo, habilitem a autenticação do IAM por meio de:

- Geração automática de tokens: os tokens do IAM são gerados automaticamente usando credenciais da AWS.
- Integração perfeita: funciona com padrões de conexão JDBC existentes.
- Suporte a credenciais da AWS: aceita vários provedores de credenciais da AWS (padrão, baseado em perfil etc.)

Usar o conector do Aurora DSQL para JDBC com agrupamento de conexões

O conector do Aurora DSQL para JDBC funciona com bibliotecas de agrupamento de conexões, como a HikariCP. O conector gerencia a geração de tokens do IAM durante o estabelecimento da conexão, permitindo que os grupos de conexões operem normalmente.

Atributos principais

Geração automática de tokens

Os tokens do IAM são gerados automaticamente usando credenciais da AWS.

Integração perfeita

Funciona com padrões de conexão JDBC existentes sem exigir alterações no fluxo de trabalho.

Suporte a credenciais da AWS

Aceita vários provedores de credenciais da AWS (padrão, baseado em perfil etc.).

Compatibilidade com o agrupamento de conexões

Funciona perfeitamente com bibliotecas de agrupamento de conexões, como a HikariCP.

Pré-requisitos

Antes de começar, você deve cumprir os seguintes pré-requisitos:

- Criar um cluster no Aurora DSQL.
- Instalar o Java Development Kit (JDK). Ter a versão 17 ou posterior.
- Configurar as permissões apropriadas do IAM para permitir que sua aplicação se conecte ao Aurora DSQL.
- Credenciais da AWS configuradas (por meio da AWS CLI, de variáveis de ambiente ou perfis do IAM).

Usar o conector do Aurora DSQL para JDBC

Para usar o conector do Aurora DSQL para JDBC em sua aplicação Java, siga estas etapas:

1. Adicione as seguintes dependências ao seu projeto Maven:

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

   Para projetos Gradle, adicione esta dependência:

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

2. Crie uma conexão básica com seu cluster do Aurora DSQL usando o formato de conector da AWS do DSQL PostgreSQL:

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

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

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

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

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

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

Propriedades de configuração

O conector do Aurora DSQL para JDBC comporta as seguintes propriedades de conexão:

usuário

Determina o usuário para a conexão e o método de geração de tokens usado. Exemplo: `admin`

token-duration-secs

Duração em segundos para a validade do token. Para ter mais informações sobre limites de token, consulte For more information on token limits, see Generating an authentication token in Amazon Aurora DSQL.

perfil

Usado para instanciar um ProfileCredentialsProvider para geração de tokens com o nome de perfil fornecido.

região

Região da AWS para conexões do Aurora DSQL. Ela é opcional. Quando fornecida, substituirá a região extraída do URL.

banco de dados

O nome do banco de dados ao qual se conectar. O padrão é `postgres`.

Registro em log

Ative o registro em log para solucionar qualquer problema que você possa enfrentar ao usar o conector JDBC do Aurora DSQL.

O conector usa o sistema de registro em log integrado (java.util.logging) do Java. Você pode configurar os níveis de registro em log criando um arquivo `logging.properties`:

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

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

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

# Detailed formatter pattern with timestamp and logger name
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Exemplos

Para ver exemplos e casos de uso mais abrangentes, consulte o repositório do conector do Aurora DSQL para JDBC.

Conector do Aurora DSQL para Python

O conector do Aurora DSQL para Python integra a autenticação do IAM para conectar aplicações Python aos clusters do Amazon Aurora DSQL. Internamente, ele utiliza bibliotecas de cliente psycopg, psycopg2 e asyncpg.

O conector do Aurora DSQL para Python foi projetado como um plug-in de autenticação que estende a funcionalidade das bibliotecas de cliente psycopg, psycopg2 e asyncpg para permitir que as aplicações se autentiquem com o Amazon Aurora DSQL usando credenciais do IAM. O conector não se conecta diretamente ao banco de dados, mas oferece uma autenticação perfeita do IAM nas bibliotecas de cliente subjacentes.

Sobre o conector

O Amazon Aurora DSQL é um serviço de banco de dados SQL distribuído que oferece alta disponibilidade e escalabilidade para aplicações compatíveis com o PostgreSQL. O Aurora DSQL exige autenticação baseada no IAM com tokens de tempo limitado para os quais as bibliotecas do Python existentes não oferecem suporte nativo.

A ideia por trás do conector do Aurora DSQL para Python é adicionar uma camada de autenticação sobre as bibliotecas de cliente psycopg, psycopg2 e asyncpg que lide com a geração de tokens do IAM, permitindo que os usuários se conectem ao Aurora DSQL sem alterar os respectivos fluxos de trabalho existentes.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de tokens: os tokens de autenticação são gerados com o uso de credenciais da AWS e têm vida útil configurável.

O conector do Aurora DSQL para Python foi projetado para entender esses requisitos e gerar tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Recursos

- Autenticação automática do IAM: os tokens do IAM são gerados automaticamente utilizando credenciais da AWS.
- Desenvolvido em psycopg, psycopg2 e asyncpg: utiliza as bibliotecas de cliente psycopg, psycopg2 e asncpg.
- Integração perfeita: funciona com os padrões de conexão psycopg, psycopg2 e asyncpg existentes sem exigir alterações no fluxo de trabalho.
- Descoberta automática de regiões: extrai a região da AWS do nome de host do cluster do DSQL.
- Suporte a credenciais da AWS: aceita vários provedores de credenciais da AWS (padrão, baseado em perfil, personalizado).
- Compatibilidade do grupo de conexões: funciona com o grupo de conexões integrado psycopg, psycopg2 e asyncpg.

Guia de início rápido

Requisitos

- Python 3.10 ou posterior
- Acesso a um cluster do Aurora DSQL
- Configurar as permissões apropriadas do IAM para permitir que sua aplicação se conecte ao Aurora DSQL.
- Credenciais da AWS configuradas (por meio da AWS CLI, de variáveis de ambiente ou perfis do IAM).

Instalação

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

Instale a psycopg, a psycopg2 ou a asyncpg separadamente.

O instalador do conector do Aurora DSQL para Python não instala as bibliotecas subjacentes. É necessário instalá-las separadamente, por exemplo:

```
# Install psycopg and psycopg pool
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
```

```
pip install psycopg2-binary
```

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

Observação:

Somente a biblioteca necessária deve ser instalada. Portanto, se o cliente for usar a psycopg, somente ela precisará ser instalada. Se o cliente for usar a psycopg2, somente ela precisará ser instalada. Se o cliente for usar a asyncpg, somente ela precisará ser instalada.

Se o cliente precisar de mais de uma, todas as bibliotecas necessárias precisarão ser instaladas.

Uso básico

psycopg

```
    import aurora_dsql_psycopg as dsql

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

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

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

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

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

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

Usar apenas o host

psycopg

```
    import aurora_dsql_psycopg as dsql

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql
```

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

Usar apenas o ID do cluster

psycopg

```
    import aurora_dsql_psycopg as dsql

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

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

Observação:

No cenário “usar apenas o ID do cluster”, a região que foi definida anteriormente na máquina é usada, por exemplo:

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

Se a região não tiver sido definida ou o ID do cluster fornecido estiver em uma região diferente, a conexão falhará. Para que funcione, forneça a região como parâmetro, como no exemplo abaixo:

psycopg

```
    import aurora_dsql_psycopg as dsql

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

```
    }

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

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

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

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

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

String de conexão

psycopg

```
    import aurora_dsql_psycopg as dsql

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

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

Configuração avançada

psycopg

```
    import aurora_dsql_psycopg as dsql

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

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

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

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

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

```
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

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

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

Opções de configuração

Opção	Tipo	Obrigatório	Descrição
`host`	`string`	Sim	Nome do host ou ID do cluster do DSQL
`user`	`string`	Não	Nome de usuário do DSQL. Padrão: admin
`dbname`	`string`	Não	Database name. Padrão: postgres
`region`	`string`	Não	Região da AWS (detectada automaticamente por meio do nome do host, se não for fornecido).
`port`	`int`	Não	O padrão é 5432.
`custom_credentials_provider`	`CredentialProvider`	Não	Provedor de credenciais personalizadas da AWS

Opção	Tipo	Obrigatório	Descrição
`profile`	`string`	Não	O nome do perfil do IAM. Padrão: padrão.
`token_duration_secs`	`int`	Não	Tempo de expiração do token em segundos.

Todas as opções de conexão padrão das bibliotecas psycopg, psycopg2 e asyncpg subjacentes também são aceitas, com exceção dos parâmetros asyncpg krbsrvname e gsslib, que não são aceitos pelo DSQL.

Usar o conector do Aurora DSQL para Python com um grupo de conexões

O conector do Aurora DSQL para Python funciona com o grupo de conexões integrado psycopg, psycopg2 e asyncpg. O conector gerencia a geração de tokens do IAM durante o estabelecimento da conexão, permitindo que os grupos de conexões operem normalmente.

psycopg

Em relação à psycopg, o conector implementa uma classe de conexão chamada DSQLConnection que pode ser transmitida diretamente ao construtor psycopg_pool.ConnectionPool. Para operações assíncronas, também há uma versão assíncrona da classe chamada DSQLaSyncConnection.

```
from psycopg_pool import ConnectionPool as PsycopgPool

...
pool = PsycopgPool(
    "",
    connection_class=dsql.DSQLConnection,
    kwargs=conn_params,
    min_size=2,
    max_size=8,
    max_lifetime=3300
)
```

Observação: configuração da conexão max_lifetime

O parâmetro max_lifetime deve ser definido como menos de 3.600 segundos (uma hora), pois essa é a duração máxima da conexão permitida pelo banco de dados Aurora DSQL. Definir um max_lifetime

inferior permite que o grupo de conexões gerencie de modo proativo a reciclagem de conexões, o que é mais eficiente do que lidar com erros de tempo limite de conexão do banco de dados.

psycopg2

Para psycopg2, o conector fornece uma classe chamada AuroraDSQLThreadedConnectionPool que herda de psycopg2.pool.ThreadedConnectionPool. A classe AuroraDSQLThreadedConnectionPool só substitui o método interno _connect. O restante da implementação é fornecido pelo psycopg2.pool.ThreadedConnectionPool inalterado.

```
import aurora_dsql_psycopg2 as dsql

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

asyncpg

Para asyncpg, o conector fornece uma função create_pool que exibe uma instância de asyncpg.Pool.

```
import asyncio
import os

import aurora_dsql_asyncpg as dsql

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

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

Autenticação

O conector processa automaticamente a autenticação do DSQL gerando tokens com o uso do gerador de tokens do cliente do DSQL. Se a região da AWS não for fornecida, ela será automaticamente analisada por meio do nome do host fornecido.

Para acessar mais informações sobre autenticação no Aurora DSQL, consulte o guia do usuário.

Admin versus usuários regulares

- Usuários chamados `"admin"` utilizam automaticamente tokens de autenticação de admin.
- Todos os outros usuários utilizam tokens de autenticação diferentes de admin.
- Os tokens são gerados dinamicamente para cada conexão.

Exemplos

Para ver o código de exemplo completo, consulte os exemplos conforme indicado nas seções abaixo. Para receber instruções sobre como executar os exemplos, consulte os arquivos LEIAME dos exemplos.

psycopg

LEIAME dos exemplos

Descrição	Exemplos
Usar o conector do Aurora DSQL para Python para conexões básicas	Exemplo básico de conexão
Usar o conector do Aurora DSQL para Python para conexões assíncronas básicas	Exemplo básico de conexão assíncrona
Usar o conector do Aurora DSQL para Python com grupo de conexões	Exemplo básico de conexão com grupo de conexões
	Exemplo de conexões simultâneas com grupo de conexões
Usar o conector do Aurora DSQL para Python com grupo de conexões assíncronas	Exemplo básico de conexão com grupo de conexões assíncronas

psycopg2

LEIAME dos exemplos

Descrição	Exemplos
Usar o conector do Aurora DSQL para Python para conexões básicas	Exemplo básico de conexão
Usar o conector do Aurora DSQL para Python com grupo de conexões	Exemplo básico de conexão com grupo de conexões
	Exemplo de conexões simultâneas com grupo de conexões

asyncpg

LEIAME dos exemplos

Descrição	Exemplos
Usar o conector do Aurora DSQL para Python para conexões básicas	Exemplo básico de conexão
Usar o conector do Aurora DSQL para Python com grupo de conexões	Exemplo básico de conexão com grupo de conexões
	Exemplo de conexões simultâneas com grupo de conexões

Conectar-se a clusters do Aurora DSQL com um conector Go

O conector do Aurora DSQL para Go encapsula o pgx com a autenticação automática do IAM. O conector gerencia a geração de tokens, a configuração SSL e o gerenciamento de conexões para que você se concentre na lógica da aplicação.

Sobre o conector

O Aurora DSQL exige autenticação baseada no IAM com tokens de tempo limitado para os quais os drivers Go do PostgreSQL não oferecem suporte nativo. O conector do Aurora DSQL para Go adiciona uma camada de autenticação sobre o driver pgx que gerencia a geração de tokens do IAM, permitindo que você se conecte ao Aurora DSQL sem alterar seus fluxos de trabalho existentes do pgx.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de token: o conector gera tokens de autenticação usando credenciais da AWS, e esses tokens têm vida útil configurável

O conector do Aurora DSQL para Go foi projetado para entender esses requisitos e gerar tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Benefícios do conector do Aurora DSQL para Go

O conector do Aurora DSQL para Go permite que você continue usando seus fluxos de trabalho existentes do pgx e, ao mesmo tempo, habilite a autenticação do IAM por meio de:

- Geração automática de tokens: o conector gera tokens do IAM automaticamente para cada conexão
- Agrupamento de conexões: suporte integrado para `pgxpool` com geração automática de tokens por conexão
- Configuração flexível: suporte a endpoints completos ou IDs de cluster com detecção automática de região.
- Suporte a credenciais da AWS: aceita perfis e provedores de credenciais personalizados da AWS.

Atributos principais

Gerenciamento automático de tokens

O conector gera tokens do IAM automaticamente para cada nova conexão usando credenciais pré-resolvidas.

Agrupamento de conexões

Agrupamento de conexões por meio de `pgxpool` com geração automática de tokens por conexão.

Configuração flexível de hosts

Oferece suporte a endpoints completos de cluster e IDs de cluster com detecção automática de região.

Segurança SSL

O SSL está sempre habilitado com o modo de verificação completa e a negociação direta de TLS.

Pré-requisitos

- Go 1.24 ou posterior.
- Credenciais da AWS configuradas.
- Um cluster do Aurora DSQL.

O conector usa a cadeia de credenciais padrão do AWS SDK para Go v2, que resolve as credenciais na seguinte ordem:

1. Variáveis de ambiente (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
2. Arquivo de credenciais compartilhadas (~/.aws/credentials)
3. Arquivo de configuração compartilhado (~/.aws/config)
4. Perfil do IAM para o Amazon EC2/ECS/Lambda

Instalação

Instale o conector usando os módulos do Go:

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

Início rápido

O seguinte exemplo mostra como criar um grupo de conexões e executar uma consulta:

```
package main

import (
    "context"
    "log"

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

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

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

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

Opções de configuração

O conector aceita as seguintes opções de configuração:

Campo	Tipo	Padrão	Descrição
Host	string	(obrigatório)	Endpoint do cluster ou ID do cluster
Região	string	(detectado automatic amente)	Região da AWS; obrigatória se o host for um ID de cluster
Usuário	string	admin	Usuário do banco de dados
Banco de dados	string	“postgres”	Nome do banco de dados
Porta	int	5432	Database port
Perfil	string	""	Nome do perfil da AWS para credenciais
TokenDura tionSecs	int	900 (15 min)	Duração da validade do token em segundos (máximo permitido: 1 semana, padrão: 15 min)
MaxConns	int32	0	Máximo de conexões do grupo (0 = padrão pgxpool)
MinConns	int32	0	Mínimo de conexões do grupo (0 = padrão pgxpool)
MaxConnLi fetime	time.Duration	55 minutos	Vida útil máxima da conexão

Formato de string da conexão

O conector é compatível com os formatos de string de conexão do PostgreSQL e do DSQL:

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

Parâmetros de consulta compatíveis:

- `region`: região da AWS
- `profile`: nome do perfil da AWS
- `tokenDurationSecs`: duração da validade do token em segundos

Exemplos:

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

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

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

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

Uso avançado

Configuração do host

O conector aceita dois formatos de host:

Endpoint completo (região detectada automaticamente):

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

ID do cluster (região obrigatória):

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

Ajuste da configuração do grupo

Configure o grupo de conexões para a workload:

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

Uso de conexão única

Para scripts simples ou quando o agrupamento de conexões não é necessário:

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

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

Usar perfis da AWS

Especifique um perfil da AWS para credenciais:

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

Nova tentativa de OCC

O Aurora DSQL usa o controle de simultaneidade otimista (OCC). Quando duas transações modificam os mesmos dados, a primeira a confirmar vence e a segunda recebe um erro de OCC.

O pacote `occretry` fornece ajudantes para nova tentativa automática com recuo exponencial e instabilidade. Instale-o com:

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

Use `WithRetry` para gravações transacionais:

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

Para declarações DDL ou únicas, use `ExecWithRetry`:

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

> **Important**
>
> `WithRetry` gerencia `BEGIN`/`COMMIT`/`ROLLBACK` internamente. Seu retorno de chamada recebe uma transação e deve conter somente operações de banco de dados e ser seguro para tentar novamente.

Exemplos

Para ver exemplos e casos de uso mais abrangentes, consulte exemplos do conector do Aurora DSQL para Go.

Exemplo	Descrição
example_preferred	Recomendado: grupo de conexões com consultas simultâneas

Exemplo	Descrição
transaction	Tratamento de transações com BEGIN/COMMIT/ROLLBACK
occ_retry	Tratamento de conflitos de OCC com recuo exponencial
connection_string	Uso de strings de conexão para configuração

Conectores do Aurora DSQL para Node.js

O conector do Aurora DSQL para node-postgres e o conector do Aurora DSQL para Postgres.js são plug-ins de autenticação que estendem a funcionalidade dos clientes node-postgres e Postgres.js para permitir que as aplicações se autentiquem com o Aurora DSQL usando credenciais do IAM.

Conector do Aurora SQL para node-postgres

O conector do Aurora DSQL para node-postgres é um conector Node.js desenvolvido em node-postgres que integra a autenticação do IAM para conectar aplicações JavaScript/TypeScript aos clusters do Amazon Aurora DSQL.

O conector do Aurora DSQL foi projetado como um plug-in de autenticação que estende a funcionalidade do cliente e do grupo do node-postgres para permitir que as aplicações se autentiquem com o Amazon Aurora DSQL utilizando credenciais do IAM.

Sobre o conector

O Amazon Aurora DSQL é um banco de dados distribuído nativo da nuvem com compatibilidade com o PostgreSQL. Embora ele exija autenticação do IAM e tokens com limite de tempo, os drivers de banco de dados Node.js tradicionais não têm esse suporte integrado.

O conector do Aurora DSQL para node-postgres preenche essa lacuna implementando um middleware de autenticação que funciona perfeitamente com o node-postgres. Essa abordagem permite que os desenvolvedores mantenham seu código node-postgres existente e, ao mesmo tempo, tenham acesso seguro baseado em IAM aos clusters do Aurora DSQL por meio do gerenciamento automatizado de tokens.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de tokens: os tokens de autenticação são gerados com o uso de credenciais da AWS e têm vida útil configurável.

O conector do Aurora DSQL para node-postgres foi projetado para entender esses requisitos e gerar tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Recursos

- Autenticação automática do IAM: trata da geração e atualização de tokens do DSQL.
- Desenvolvido em node-postgres: utiliza o conhecido cliente PostgreSQL para Node.js.
- Integração perfeita: funciona com padrões existentes de conexão do node-postgres.
- Descoberta automática de regiões: extrai a região da AWS do nome de host do cluster do DSQL.
- Suporte completo ao TypeScript: oferece segurança total de tipos.
- Suporte a credenciais da AWS: aceita vários provedores de credenciais da AWS (padrão, baseado em perfil, personalizado).
- Compatibilidade com agrupamento de conexões: funciona perfeitamente com o agrupamento de conexões integrado.

Aplicação de exemplo

Há uma aplicação de exemplo incluída em exemplo, que mostra como usar o conector do Aurora DSQL para node-postgres. Para executar o exemplo incluído, consulte o respectivo LEIAME

Guia de início rápido

Requisitos

- Node.js 20
- Acesso a um cluster do Aurora DSQL
- Configurar as permissões apropriadas do IAM para permitir que sua aplicação se conecte ao Aurora DSQL.
- Credenciais da AWS configuradas (por meio da AWS CLI, de variáveis de ambiente ou perfis do IAM).

Instalação

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

Dependências de pares

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

Usage

Conexão do cliente

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

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

Conexão do grupo

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

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

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

Uso avançado

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

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

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

Opções de configuração

Opção	Tipo	Obrio	Descrição
`host`	`string`	Sim	Nome do host do cluster do DSQL
`username`	`string`	Sim	Nome de usuário do DSQL
`database`	`string`	Não	Nome do banco de dados
`region`	`string`	Não	Região da AWS (detectada automaticamente por meio do nome do host, se não for fornecido).
`port`	`number`	Não	O padrão é 5432.
`customCredentialsProvider`	`AwsCredentialIdentity / AwsCredentialIdentityProvider`	Não	Provedor de credenciais personalizadas da AWS
`profile`	`string`	Não	O nome do perfil do IAM. Tem como padrão “padrão”.
`tokenDurationSecs`	`number`	Não	Tempo de expiração do token em segundos.

Todos os outros parâmetros de Cliente/Grupo são aceitos.

Autenticação

O conector processa automaticamente a autenticação do DSQL gerando tokens com o uso do gerador de tokens do cliente do DSQL. Se a região da AWS não for fornecida, ela será automaticamente analisada por meio do nome do host fornecido.

Para acessar mais informações sobre autenticação no Aurora DSQL, consulte o guia do usuário.

Admin versus usuários regulares

- Usuários chamados “admin” utilizam automaticamente tokens de autenticação de admin.
- Todos os outros usuários utilizam tokens de autenticação regulares.
- Os tokens são gerados dinamicamente para cada conexão.

Conector do Aurora SQL para Postgres.js

O conector do Aurora DSQL para Postgres.js é um conector Node.js desenvolvido em Postgres.js que integra a autenticação do IAM para conectar aplicações Javascript aos clusters do Amazon Aurora DSQL.

O conector do Aurora DSQL para Postgres.js foi projetado como um plug-in de autenticação que estende a funcionalidade do cliente do Postgres.js para permitir que as aplicações se autentiquem com o Amazon Aurora DSQL utilizando credenciais do IAM. O conector não se conecta diretamente ao banco de dados, mas oferece uma autenticação perfeita do IAM no driver subjacente do Postgres.js.

Sobre o conector

O Amazon Aurora DSQL é um serviço de banco de dados SQL distribuído que oferece alta disponibilidade e escalabilidade para aplicações compatíveis com o PostgreSQL. O Aurora DSQL exige autenticação baseada no IAM com tokens de tempo limitado que não são aceitos pelos drivers Node.js.

A ideia por trás do conector do Aurora DSQL para Postgres.js é adicionar uma camada de autenticação sobre o cliente do Postgres.js que gerencia a geração de tokens do IAM, permitindo que os usuários se conectem ao Aurora DSQL sem alterar os respectivos fluxos de trabalho do Postgres.js existentes.

O conector do Aurora DSQL para Postgres.js funciona com a maioria das versões do Postgres.js. Os usuários fornecem a própria versão instalando o Postgres.js diretamente.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de tokens: os tokens de autenticação são gerados com o uso de credenciais da AWS e têm vida útil configurável.

O conector do Aurora DSQL para Postgres.js foi projetado para entender esses requisitos e gerar tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Recursos

- Autenticação automática do IAM: trata da geração e atualização de tokens do DSQL.
- Desenvolvido em Postgres.js: utiliza o rápido cliente do PostgreSQL para Node.js.
- Integração perfeita: funciona com padrões existentes de conexão do Postgres.js.
- Descoberta automática de regiões: extrai a região da AWS do nome de host do cluster do DSQL.
- Suporte completo ao TypeScript: oferece segurança total de tipos.
- Suporte a credenciais da AWS: aceita vários provedores de credenciais da AWS (padrão, baseado em perfil, personalizado).
- Compatibilidade com agrupamento de conexões: funciona perfeitamente com o agrupamento de conexões integrado do Postgres.js.

Guia de início rápido

Requisitos

- Node.js 20
- Acesso a um cluster do Aurora DSQL
- Configurar as permissões apropriadas do IAM para permitir que sua aplicação se conecte ao Aurora DSQL.
- Credenciais da AWS configuradas (por meio da AWS CLI, de variáveis de ambiente ou perfis do IAM).

Instalação

```
npm install @aws/aurora-dsql-postgresjs-connector
```

```
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Uso básico

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

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

});

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

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

Utilizar o ID do cluster em vez do host

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

});
```

String de conexão

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

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

Configuração avançada

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

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

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

Opções de configuração

Opção	Tipo	Obrigatório	Descrição
`host`	`string`	Sim	Nome do host ou ID do cluster do DSQL
`database`	`string?`	Não	Nome do banco de dados
`username`	`string?`	Não	Nome de usuário do banco de dados (usará admin se não for fornecido)
`region`	`string?`	Não	Região da AWS (detectada automaticamente por meio do nome do host, se não for fornecido).
`customCredentialsProvider`	`AwsCredentialIdentityProvider?`	Não	Provedor de credenciais personalizadas da AWS
`tokenDurationSecs`	`number?`	Não	Tempo de expiração do token em segundos.

Todas as opções padrão do Postgres.js também são aceitas.

Autenticação

O conector processa automaticamente a autenticação do DSQL gerando tokens com o uso do gerador de tokens do cliente do DSQL. Se a região da AWS não for fornecida, ela será automaticamente analisada por meio do nome do host fornecido.

Para acessar mais informações sobre autenticação no Aurora DSQL, consulte o guia do usuário.

Admin versus usuários regulares

- Usuários chamados “admin” utilizam automaticamente tokens de autenticação de admin.
- Todos os outros usuários utilizam tokens de autenticação regulares.
- Os tokens são gerados dinamicamente para cada conexão.

Exemplo de uso

Exemplos de JavaScript usando o conector do Aurora DSQL para Postgres.js estão disponíveis no GitHub. Para ver instruções sobre como executar os exemplos, consulte o diretório de exemplos.

Descrição	Exemplo
Agrupamento de conexões com consultas simultâneas, incluindo criação de tabelas, inserções e leituras em vários operadores	Exemplo de grupo de conexões (preferencial)
Operações CRUD (criar tabela, inserir, selecionar, excluir) sem grupo de conexões	Exemplo sem grupo de conexões

Conectar-se a clusters do Aurora DSQL com um conector Ruby

O conector do Aurora DSQL para Ruby é um conector Ruby desenvolvido em pg que integra a autenticação do IAM para conectar aplicações Ruby aos clusters do Amazon Aurora DSQL.

O conector gerencia a geração de tokens, a configuração SSL e o agrupamento de conexões para que você se concentre na lógica da aplicação.

Sobre o conector

O Amazon Aurora DSQL exige autenticação do IAM com tokens de tempo limitado para os quais os drivers PostgreSQL do Ruby existentes não oferecem suporte nativo. O conector do Aurora DSQL para Ruby adiciona uma camada de autenticação sobre o driver pg que gerencia a geração de tokens do IAM, permitindo que você se conecte ao Aurora DSQL sem alterar seus fluxos de trabalho existentes do pg.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de token: o conector gera tokens de autenticação usando credenciais da AWS, e esses tokens têm vida útil configurável

O conector do Aurora DSQL para Ruby entende esses requisitos e gera tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Recursos

- Autenticação automática do IAM: lida com a geração e atualização de tokens do Aurora DSQL
- Desenvolvido em pg: agrupa o popular gem do PostgreSQL para Ruby
- Integração perfeita: funciona com fluxos de trabalho de pg gem existentes
- Agrupamento de conexões: suporte integrado por meio do `connection_pool` gem com aplicação de max_lifetime
- Detecção automática de região: extrai a região da AWS do nome de host do cluster do Aurora DSQL
- Suporte a credenciais da AWS: aceita perfis e provedores de credenciais personalizados da AWS
- Nova tentativa de OCC: opte por uma nova tentativa de controle de simultaneidade otimista com recuo exponencial

Aplicativo de exemplo

Para ver um exemplo completo, consulte a aplicação de exemplo no GitHub.

Guia de início rápido

Requisitos

- Ruby 3.1 ou posterior
- Acesso a um cluster do Aurora DSQL
- Credenciais da AWS configuradas (por meio da CLI da AWS, de variáveis de ambiente ou perfis do IAM)

Instalação

Adicione a seu Gemfile:

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

Ou instale diretamente:

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

Usage

Conexão do grupo

```
require "aurora_dsql_pg"

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

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

```
end

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

pool.shutdown
```

Conexão única

Para scripts simples ou quando o agrupamento de conexões não é necessário:

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

Uso avançado

Configuração do host

O conector é compatível com endpoints de cluster completos (região detectada automaticamente) e IDs de cluster (região obrigatória):

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

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

AWS Perfis do

Especifique um perfil da AWS para credenciais:

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

Formato de string da conexão

O conector é compatível com os formatos de string de conexão do PostgreSQL:

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

Parâmetros de consulta compatíveis: `region`, `profile`, `tokenDurationSecs`.

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

Nova tentativa de OCC

O Aurora DSQL usa o controle de simultaneidade otimista (OCC). Quando duas transações modificam os mesmos dados, a primeira a confirmar vence e a segunda recebe um erro de OCC.

A nova tentativa do OCC é opcional. Defina `occ_max_retries` ao criar o agrupamento para ativar a nova tentativa automática com recuo exponencial e instabilidade em `pool.with`:

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

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

> **Warning**
>
> O `pool.with` NÃO agrupa automaticamente seu bloco em uma transação. Você deve chamar `conn.transaction` por conta própria para operações de gravação. Em caso de conflito de OCC, o conector reexecuta o bloco inteiro, portanto, ele deve conter somente operações de banco de dados e ser seguro tentar novamente.

Para pular a nova tentativa de chamadas individuais, passe `retry_occ: false`:

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

Opções de configuração

Campo	Tipo	Padrão	Descrição
host	String	(obrigatório)	Endpoint do cluster ou ID do cluster
region	String	(detectado automatic amente)	Região da AWS; obrigatória se o host for um ID de cluster
usuário	String	admin	Usuário do banco de dados
banco de dados	String	“postgres”	Nome do banco de dados
porta	Inteiro	5432	Database port
perfil	String	nulo	Nome do perfil da AWS para credenciais
token_duration	Inteiro	900 (15 min)	Duração da validade do token em segundos (máximo permitido: 1 semana, padrão: 15 min)
credentia ls_provider	Aws::Credentials	nulo	Provedor de credenciais personali zadas

Campo	Tipo	Padrão	Descrição
max_lifetime	Inteiro	3300 (55 min)	Vida útil máxima da conexão em segundos
applicati on_name	String	nulo	Prefixo ORM para application_name
Logger	Logger	nulo	Registrador para avisos de nova tentativa de OCC
occ_max_retries	Inteiro	nulo (desabili tado)	Máximo de novas tentativas de OCC em `pool.with` ; habilita a nova tentativa quando definido

`create_pool` também aceita uma palavra-chave `pool:` com um hash de opções que você passa diretamente para `ConnectionPool.new`. Se você omitir `pool:`, o conector assume `{size: 5, timeout: 5}` como padrão. As chaves fornecidas substituem somente esses padrões específicos.

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

Autenticação

O conector gerencia automaticamente a autenticação do Aurora DSQL gerando tokens com o uso de credenciais da AWS. Se você não fornecer a região da AWS, o conector a analisará a partir do nome do host.

Para obter mais informações sobre autenticação no Aurora DSQL, consulte Autenticação e autorização para o Aurora DSQL.

Admin versus usuários regulares

- Usuários chamados “admin” utilizam automaticamente tokens de autenticação de admin.
- Todos os outros usuários utilizam tokens de autenticação regulares.
- O conector gera tokens dinamicamente para cada conexão

Conectar-se a clusters do Aurora DSQL com um conector .NET

O conector do Amazon Aurora DSQL para .NET é um conector .NET desenvolvido em Npgsql que integra a autenticação do IAM para conectar aplicações .NET aos clusters do Amazon Aurora DSQL.

O conector gerencia a geração de tokens, a configuração SSL e o agrupamento de conexões para que você se concentre na lógica da aplicação.

Sobre o conector

O Amazon Aurora DSQL exige autenticação do IAM com tokens de tempo limitado para os quais os drivers PostgreSQL do .NET existentes não oferecem suporte nativo. O conector do Aurora DSQL para .NET adiciona uma camada de autenticação sobre o Npgsql que gerencia a geração de tokens do IAM, permitindo que você se conecte ao Aurora DSQL sem alterar seus fluxos de trabalho existentes do Npgsql.

O que é a autenticação do Aurora DSQL?

Na autenticação do Aurora DSQL, a autenticação envolve:

- Autenticação do IAM: todas as conexões usam autenticação baseada no IAM com tokens de tempo limitado.
- Geração de token: o conector gera tokens de autenticação usando credenciais da AWS, e esses tokens têm vida útil configurável

O conector do Aurora DSQL para .NET entende esses requisitos e gera tokens de autenticação do IAM automaticamente ao estabelecer conexões.

Recursos

- Autenticação automática do IAM: lida com a geração e atualização de tokens do Aurora DSQL
- Desenvolvido em Npgsql: agrupa o popular driver PostgreSQL para.NET
- Integração perfeita: funciona com fluxos de trabalho de Npgsql existentes
- Agrupamento de conexões: suporte integrado por meio do `NpgsqlDataSource` com aplicação de vida útil máxima
- Detecção automática de região: extrai a região da AWS do nome de host do cluster do Aurora DSQL
- Suporte a credenciais da AWS: aceita perfis e provedores de credenciais personalizados da AWS

- Nova tentativa de OCC: opte por uma nova tentativa de controle de simultaneidade otimista com recuo exponencial
- Aplicação de SSL: sempre usa SSL com o modo `verify-full` e negociação direta de TLS

Aplicativo de exemplo

Para ver um exemplo completo, consulte a aplicação de exemplo no GitHub.

Guia de início rápido

Requisitos

- .NET 8.0 ou posterior
- Acesso a um cluster do Aurora DSQL
- Credenciais da AWS configuradas (por meio da CLI da AWS, de variáveis de ambiente ou perfis do IAM)

Instalação

Adicione o pacote ao seu projeto:

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

Usage

Conexão do grupo

```
using Amazon.AuroraDsql.Npgsql;

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

// Read
await using (var conn = await ds.OpenConnectionAsync())
{
```

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

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

Conexão única

Para scripts simples ou quando você não tem o agrupamento de conexões:

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

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

Nova tentativa de OCC

O Aurora DSQL usa o controle de simultaneidade otimista (OCC). Quando duas transações modificam os mesmos dados, a primeira a confirmar vence e a segunda recebe um erro de OCC.

A nova tentativa do OCC é opcional. Defina `OccMaxRetries` na configuração para habilitar nova tentativa automática com recuo exponencial e instabilidade. Use `WithTransactionRetryAsync` para gravações transacionais:

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

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

```
    await cmd.ExecuteNonQueryAsync();

    cmd.CommandText = "UPDATE accounts SET balance = balance + 100 WHERE id = @to";
    cmd.Parameters.Clear();
    cmd.Parameters.AddWithValue("to", toId);
    await cmd.ExecuteNonQueryAsync();
});
```

Para declarações DDL ou únicas, use `ExecWithRetryAsync`:

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

> **Important**
>
> `WithTransactionRetryAsync` gerencia `BEGIN`/`COMMIT`/`ROLLBACK` internamente e abre uma nova conexão para cada tentativa. Seu retorno de chamada deve conter somente operações de banco de dados e ser seguro tentar novamente.

Opções de configuração

O conector também aceita strings de conexão `postgres://` e `postgresql://` com os parâmetros de consulta `region` e `profile`.

Campo	Tipo	Padrão	Descrição
`Host`	`string`	(obrigatório)	Endpoint do cluster ou ID do cluster de 26 caracteres
`Region`	`string?`	(detectado automatic amente)	Região da AWS; obrigatória se o `Host` for um ID de cluster
`User`	`string`	`"admin"`	Usuário do banco de dados
`Database`	`string`	`"postgres"`	Nome do banco de dados
`Port`	`int`	`5432`	Database port

Campo	Tipo	Padrão	Descrição
`Profile`	`string?`	`null`	Nome do perfil da AWS para credenciais
`CustomCredentialsProvider`	`AWSCredentials?`	`null`	Provedor de credenciais da AWS personalizadas
`TokenDurationSecs`	`int?`	`null` (SDK padrão, 900s)	Duração da validade do token em segundos
`OccMaxRetries`	`int?`	`null` (desabilitado)	Máximo padrão de novas tentativas de OCC para métodos de nova tentativa na fonte de dados
`OrmPrefix`	`string?`	`null`	Prefixo ORM prefixado a `application_name`
`LoggerFactory`	`ILoggerFactory?`	`null`	Fábrica de loggers para novas tentativas, avisos e diagnósticos
`ConfigureConnectionString`	`Action<NpgsqlConnectionStringBuilder>?`	`null`	Retorno de chamada para substituir as configurações do agrupamento ou definir propriedades adicionais da string de conexão Npgsql. SSL e `Enlist` são invariantes de segurança e não podem ser substituídos.

Autenticação

O conector gerencia automaticamente a autenticação do Aurora DSQL gerando tokens com o uso de credenciais da AWS. Se você não fornecer a região da AWS, o conector a analisará a partir do nome do host.

Para obter mais informações sobre autenticação no Aurora DSQL, consulte Autenticação e autorização para o Aurora DSQL.

Admin versus usuários regulares

- Usuários chamados “admin” utilizam automaticamente tokens de autenticação de admin.
- Todos os outros usuários utilizam tokens de autenticação regulares.
- O conector gera tokens dinamicamente para cada conexão

Acessar o Aurora DSQL com clientes compatíveis com o PostgreSQL

O Aurora DSQL usa o protocolo wire do PostgreSQL. Você pode se conectar ao PostgreSQL usando uma série de ferramentas e clientes, como AWS CloudShell, psql, DBeaver e DataGrip. A tabela a seguir resume como o Aurora DSQL associa parâmetros comuns de conexão do PostgreSQL:

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 o perfil `admin` para associá-los aos perfis do IAM para autenticação ao se conectar ao cluster. Para obter mais informações, consulte Usar perfis de banco de dados e autenticação do IAM.
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 habilitad	No Aurora DSQL, o Aurora DSQL é compatível com o modo SSL `require`.

PostgreSQL	Aurora DSQL	Observações
	o do lado do servidor.	Conexões sem SSL são rejeitadas pelo Aurora DSQL.
Senha	Token de autentica ção	O Aurora DSQL exige tokens de autentica ção temporários em vez de senhas de longa duração. Para saber mais, consulte Gerar um token de autenticação no Amazon Aurora DSQL.

Ao se conectar, o Aurora DSQL exige um token de autenticação do IAM assinado no lugar de uma senha tradicional. Esses tokens temporários são gerados usando o AWS Signature Versão 4 e são usados somente durante o estabelecimento da conexão. Depois de conectada, a sessão permanece ativa até que ela termine ou o cliente se desconecte.

Se você tentar abrir uma nova sessão com um token expirado, a solicitação de conexão falhará e um novo token deverá ser gerado. Para obter mais informações, consulte Gerar um token de autenticação no Amazon Aurora DSQL.

Acessar o Aurora DSQL usando clientes SQL

O Aurora DSQL comporta vários clientes compatíveis com PostgreSQL para conexão com seu cluster. As seções a seguir descrevem como se conectar usando o PostgreSQL com AWS CloudShell ou sua linha de comandos local, bem como ferramentas baseadas em GUI, como DBeaver e JetBrains DataGrip. Cada cliente exige um token de autenticação válido, conforme descrito na seção anterior.

Tópicos

- Use o DBeaver para acessar o Aurora DSQL
- Usar o JetBrains DataGrip para acessar o Aurora DSQL
- Usar o terminal interativo do PostgreSQL (psql) para acessar o Aurora DSQL
- Usar o driver do Aurora DSQL para SQLTools
- Solução de problemas

Use o DBeaver para acessar o Aurora DSQL

O DBeaver é um cliente SQL universal que pode ser usado para gerenciar qualquer banco de dados que tenha um driver JDBC. Ele é amplamente usado entre desenvolvedores e administradores de banco de dados por conta de seus recursos robustos de visualização, edição e gerenciamento de dados. Usando as opções de conectividade em nuvem do DBeaver, é possível conectar o DBeaver ao Aurora DSQL de forma nativa.

DBeaver Pro

Os produtos do DBeaver PRO oferecem integração nativa com o Aurora DSQL a partir da versão 25.3. Siga as instruções da documentação do DBeaver para se conectar a um cluster do Aurora DSQL.

DBeaver Community Edition

O DBeaver Community Edition é a versão gratuita e de código aberto. Acesse a página de download para ver instruções de instalação. Para se conectar ao DSQL usando o DBeaver Community Edition, é necessário instalar o plug-in do Aurora DSQL para DBeaver.

O plug-in do Aurora DSQL para DBeaver foi projetado com base no conector do Aurora DSQL para JDBC e permite a autenticação do IAM em clusters do Aurora DSQL. Ele é convenientemente instalado por meio da interface de usuário do DBeaver e elimina a necessidade de escrever código de geração de tokens ou fornecer manualmente um token válido do IAM, simplificando a autenticação e eliminando os riscos de segurança associados a senhas tradicionais geradas pelo usuário.

Recursos

- Suporte à autenticação do IAM: conecte-se aos clusters do Aurora DSQL usando as credenciais do AWS IAM para realizar uma autenticação segura e sem senha.
- Gerenciamento automático de drivers: instala e configura perfeitamente o conector do Aurora DSQL para JDBC.
- Opções flexíveis de conexão: escolha entre configuração de conexão baseada em host ou no URL JDBC.

Instalação do plug-in do Aurora SQL para DBeaver

1. Com o DBeaver aberto, acesse o menu suspenso Ajuda → Instalar novo software.

2. Clique em Adicionar para adicionar um novo repositório.
3. Insira:
   - Nome: `Aurora DSQL Plugin`
   - Local: `https://awslabs.github.io/aurora-dsql-dbeaver-plugin/update-site/`.
4. Marque Conector do Aurora DSQL para JDBC.
5. Clique em Próximo, aceite a licença e conclua a instalação
6. Reinicie o DBeaver quando solicitado.

Crie uma conexão do Aurora DSQL.

1. Clique em Nova conexão de banco de dados.
2. Selecione Aurora DSQL.
3. Em Servidor, selecione uma das seguintes opções para a configuração Conectar usando:
   - Host
     - Para habilitar as entradas de texto da interface de usuário para os seguintes campos:
       - Endpoint: endpoint de cluster do DSQL.
       - Nome de usuário: nome de usuário do DSQL (p. ex., admin).
       - Perfil da AWS: p. ex., padrão. O perfil padrão usado quando nenhum perfil específico é estipulado.
       - Região da AWS (opcional): deve corresponder à região em que o cluster do DSQL se encontra. Do contrário, a autenticação falhará.
   - URL do
     - URL JDBC neste formato:

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

     - Observação: nesse modo, somente a entrada de URL está habilitada. Para adicionar parâmetros à string de conexão JDBC, use o formato de parâmetros de consulta de URL começando com ? como o primeiro parâmetro e acrescente um & para os parâmetros subsequentes.
4. Clique em Testar conexão para verificar se a conexão do Aurora DSQL está funcionando.
5. Clique em Concluir.

Solução de problemas

Problema no Windows Trust Store

Os usuários do Windows podem encontrar problemas ao baixar o conector do Aurora DSQL para o driver JDBC por meio do Maven Central.

Causa: o Windows Trust Store pode não incluir os certificados necessários para acessar o repositório Maven Central.

Solução:

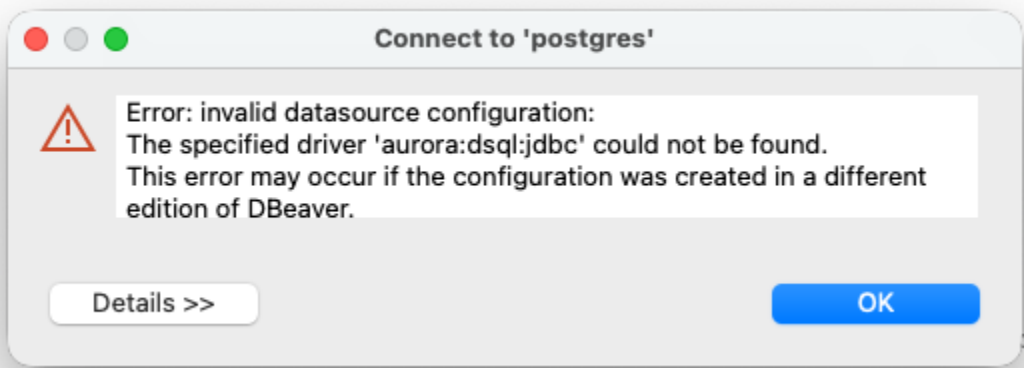
1. Execute o DBeaver como “Administrador”.
2. Desmarque esta configuração: Windows > Preferências > Conexões > “Usar o Windows Trust Store”.

Erro de driver ausente

Se você vir um ícone de driver ausente ou erros de conexão, o Aurora DSQL (plug-in do Community) pode não estar instalado em sua versão atual do DBeaver. Confira abaixo alguns exemplos de erros e como corrigi-los:

- Criar uma conexão sem ter o driver instalado:

- Tentar realizar uma conexão sem o driver:



Causa: quando várias versões do DBeaver são instaladas, as configurações de conexão são compartilhadas, mas os drivers são instalados por aplicação.

Solução: reinstale o Aurora DSQL (plug-in do Community) seguindo as etapas de instalação acima.

> **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 bancos de dados do Aurora DSQL devido à respectiva arquitetura exclusiva. Embora acessíveis, essas telas não fornecem informações confiáveis sobre a integridade ou o status do banco de dados.

Usar o JetBrains DataGrip para acessar o Aurora DSQL

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 and PGSSLMODE=verify-full`. O Aurora DSQL impõe a comunicação SSL do lado do servidor e rejeita conexões que não sejam SSL. Para a opção `verify-full`, você precisará instalar os certificados SSL localmente. Para acessar mais informações, consulte Certificados SSL/TLS.

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

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

Usar o terminal interativo do PostgreSQL (psql) para acessar o Aurora DSQL

Use o AWS CloudShell para acessar o Aurora DSQL com o terminal interativo do PostgreSQL (psql).

Use o procedimento a seguir para acessar o Aurora DSQL com o terminal interativo do PostgreSQL do AWS CloudShell. Para obter mais informações, consulte O que é o AWS CloudShell.

Como se conectar usando AWS CloudShell

1. Faça login no console do Aurora DSQL.
2. Escolha o cluster que você deseja abrir no CloudShell. Se você ainda não criou um cluster, siga as etapas em Etapa 1: criar um cluster do Aurora DSQL de região única ou Criar um cluster multirregional.
3. Escolha Conectar com o Editor de consultas e, depois, selecione Conectar com o CloudShell.
4. Escolha se você deseja se conectar como administrador ou com um perfil de banco de dados personalizado.
5. Escolha Iniciar no CloudShell e selecione Executar na caixa de diálogo do CloudShell a seguir.

Use a CLI local para acessar o Aurora DSQL com o terminal interativo do PostgreSQL (psql).

Use `psql`, um frontend baseado em terminal para o utilitário PostgreSQL, para inserir consultas interativamente, emiti-las para o PostgreSQL e visualizar os resultados da consulta.

> **Note**
>
> Para melhorar os tempos de resposta das consultas, use o cliente PostgreSQL versão 17. Se você usar a CLI em um ambiente diferente, configure manualmente o Python versão 3.8+ e o psql versão 14+.

Baixe o instalador do seu sistema operacional na página PostgreSQL Downloads. Para acessar mais informações sobre `psql`, consulte Aplicações cliente do PostgreSQL no site do PostgreSQL.

Se você já tem a AWS CLI instalada, use o exemplo a seguir para se conectar ao cluster.

```
# 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
```

Usar o driver do Aurora DSQL para SQLTools

O driver do Aurora DSQL para SQLTools é uma extensão do Visual Studio Code para o Amazon Aurora DSQL que se integra ao SQLTools. Ele permite que os desenvolvedores se conectem e consultem bancos de dados do Aurora DSQL diretamente do VS Code. O driver está disponível para instalação no Visual Studio Marketplace e no Open VSX Registry. O Kiro, o Cursor e outros IDEs baseados no VSCode podem usar o Open VSX Registry para instalar o driver seguindo o procedimento de instalação padrão descrito na respectiva página.

Recursos

- Autenticação automática do IAM
- Operações de banco de dados padrão, como procurar esquemas e tabelas e executar consultas SQL.

Instalação

1. Abra a visualização “Extensões”.
2. Pesquise “Driver do Aurora DSQL para SQLTools”.
3. Clique em “Instalar”.

Observação:

Se a extensão do SQLTools ainda não tiver sido instalada, ela será instalada automaticamente.

Autenticação

No Aurora DSQL, todas as conexões usam a autenticação baseada no IAM com tokens de tempo limitado. O driver manipula automaticamente a autenticação do Aurora DSQL usando o conector do Aurora DSQL para node-postgres.

Para acessar mais informações sobre autenticação no Aurora DSQL, consulte o guia do usuário.

Crie uma conexão do Aurora DSQL.

Pré-requisitos

- Credenciais da AWS configuradas (por meio da AWS CLI, de variáveis de ambiente ou perfis do IAM).

Steps

1. Clique no ícone do SQLTools na barra lateral à esquerda.
2. No painel do SQLTools, passe o mouse sobre CONEXÕES e clique no ícone “Adicionar nova conexão”.
3. Na guia “Configurações do SQLTools”, selecione “Driver do Aurora DSQL” na lista.
4. Preencha os parâmetros de conexão.
   - Região da AWS
     - Opcional: a região será analisada no endpoint do cluster do Aurora DSQL.
     - Obrigatório quando somente um ID de cluster é especificado no campo “Cluster do DSQL”.
   - Perfil da AWS
     - Usado para geração de tokens.
     - Se nenhum perfil for especificado, será o usado o padrão.
5. Clique no link “Testar conexão” para testá-la.
6. Clique em “Salvar conexão”.

Solução de problemas

Expiração das credenciais de autenticação para os clientes SQL

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 da conexão. Quando você tenta abrir uma nova sessão (por exemplo, para listar novas tabelas ou abrir 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. Tenha isso em mente ao escolher a duração do token de autenticação do IAM com a opção `expires-in` que pode ser definida como 15 minutos por padrão e como um valor máximo de sete dias.

Além disso, consulte a seção Solução de problemas da documentação do Aurora DSQL.

Ferramentas de conectividade com clusters do Amazon Aurora DSQL

O Aurora DSQL é compatível com muitos drivers de banco de dados e bibliotecas de ORM de terceiros. A AWS oferece dois tipos de ferramenta para simplificar o trabalho com o Aurora DSQL:

- Conectores: plug-ins de autenticação que estendem os drivers do banco de dados para lidar automaticamente com a geração de tokens do IAM. Use conectores ao trabalhar diretamente com drivers de banco de dados.
- Adaptadores e dialetos: extensões para frameworks específicos de ORM que oferecem autenticação do IAM e compatibilidade aprimorada com o Aurora DSQL. Use adaptadores ao trabalhar com um framework de ORM compatível.

Adaptadores e dialetos do Aurora DSQL

A tabela a seguir mostra os adaptadores e dialetos disponíveis para o Aurora DSQL.

Linguagem de programação	ORM/Framework	Link de repositório
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate

Linguagem de programação	ORM/Framework	Link de repositório
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy
Python	Tortoise ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm

Exemplos de driver de banco de dados

A tabela a seguir mostra um exemplo de código para estabelecer conexão com o Aurora DSQL usando drivers de banco de dados de terceiros.

Linguagem de programação	Driver	Exemplo de link de repositório
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	HikariCP + pgJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc

Linguagem de programação	Driver	Exemplo de link de repositório
JavaScript	node-postgres (AWS Lambda)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/lambda
JavaScript	node-postgres	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/node-postgres
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	asyncpg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/asyncpg
Python	Psycopg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg2	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg2
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx

Exemplos de ORM e framework

A tabela a seguir mostra um exemplo de código para usar bibliotecas e frameworks de ORM de terceiros com o Aurora DSQL.

Linguagem de programação	ORM/Framework	Exemplo de link de repositório
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate/examples/pet-clinic-app
Java	Liquibase	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/liquibase
Java	Spring Boot	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/spring_boot
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django/examples/pet-clinic-app
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy/examples/pet-clinic-app
Python	Tortoise ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm/example
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails

Linguagem de programação	ORM/Framework	Exemplo de link de repositório
TypeScript	Prisma	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/prisma
TypeScript	Sequelize	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	TypeORM	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/type-orm

Carregar dados no Aurora DSQL

Seja migrando a partir de um banco de dados existente, importando arquivos de um Amazon Simple Storage Service ou carregando dados de seu sistema local, o Aurora DSQL fornece várias abordagens para inserir seus dados. Esta seção aborda as ferramentas e técnicas recomendadas para carregamentos de dados de todos os tamanhos, de poucos gigabytes a centenas de terabytes.

Escolher uma abordagem de migração

O Aurora DSQL oferece suporte aos comandos padrão de carregamento de dados do PostgreSQL, mas carregar dados de forma eficiente em grande escala exige lidar com paralelização, gerenciamento de conexões e recuperação de erros. A seguinte tabela resume suas opções:

Abordagem	Melhor para	Considerações
Aurora DSQL Loader: utilitário de código aberto que facilita a paralelização de inserções ao usar o Aurora DSQL	A maioria dos cenários de carregamento de dados, especialmente migrações e importações em massa	Lida automaticamente com paralelização, agrupamento de conexões, resolução de conflitos e autenticação do IAM. Disponível como código-fonte ou binário.

Abordagem	Melhor para	Considerações
PostgreSQL **`\copy`**: meta-comando `psql` do lado do cliente	Carregamentos simples quando há conexão pelo `psql`	Lê arquivos no cliente e transmite dados pela conexão; você gerencia a paralelização
Transações **`INSERT`**: `SQL DML` padrão	Pequenos conjuntos de dados ou inserções orientadas por aplicações	Abordagem mais simples, mas mais lenta para dados em massa

Para a maioria das tarefas de carregamento de dados, use o Aurora DSQL Loader. Ele lida com a complexidade operacional de carregar dados em um banco de dados distribuído, incluindo a execução paralela em várias conexões e a repetição automática de operações com falha.

Aurora DSQL Loader

O Aurora DSQL Loader é um utilitário de linha de comando de código aberto projetado para carregar dados com eficiência nos clusters do Aurora DSQL. Ele gerencia o agrupamento de conexões, paraleliza a transferência de dados entre vários trabalhadores e lida com conflitos e novas tentativas automaticamente.

Atributos principais

O Aurora DSQL Loader fornece os seguintes recursos:

Carregamento paralelo

: Os threads de trabalho configuráveis permitem o carregamento simultâneo de dados em várias conexões para melhorar o desempenho.

Agrupamento de conexões

: Gerencia um agrupamento de conexões ao cluster Aurora DSQL, gerenciando automaticamente a autenticação do IAM e o ciclo de vida da conexão.

Suporte ao novo formato de arquivo

Compatível com CSV (valores separados por vírgula), TSV (valores separados por tabulação) e formato colunar do Apache Parquet. O carregador detecta automaticamente o formato do arquivo com base na extensão do URI de origem.

Geração automática de esquemas

Quando usado com o sinalizador `--if-not-exists`, o carregador pode criar automaticamente tabelas com os tipos de coluna apropriados com base nos dados.

Criação automática de conflitos

Quando sua tabela de destino tiver restrições exclusivas, configure como o carregador lida com conflitos usando a opção `--on-conflict`: ignorar duplicatas, atualizar registros ou retornar um erro.

Tolerância a falhas

As novas tentativas automáticas e os recursos de retomada do trabalho garantem que as cargas interrompidas possam continuar a partir do ponto de parada em vez de serem reiniciadas.

Fontes locais e do S3

Carregue dados dos caminhos do sistema de arquivos local ou diretamente dos buckets do Amazon S3 usando URIs do S3.

Pré-requisitos

Antes de usar o Aurora DSQL Loader, verifique se tem o seguinte:

- Um cluster ativo do Aurora DSQL com um endpoint válido.
- Credenciais da AWS configuradas por meio da AWS CLI (aws configure), do AWS Single Sign-On (aws sso login) ou perfis do IAM.
- Permissões do IAM: `dsql:DbConnectAdmin` ou `dsql:DbConnect` em seu cluster Aurora DSQL.
- Para fontes do S3, permissões adequadas para ler da fonte do bucket.

Instalação

Baixe a versão mais recente na página de versões do GitHub. Binários pré-criados estão disponíveis para plataformas comuns. Para obter instruções sobre como criar a partir do código-fonte, consulte o repositório do Aurora DSQL Loader.

Exemplos de uso

Os exemplos a seguir demonstram casos de uso comuns do Aurora DSQL Loader.

Example Carregar um arquivo CSV local

Este exemplo carrega um arquivo CSV do seu sistema de arquivos local em uma tabela existente:

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

Example Carregar dados do Amazon S3

Este exemplo carrega um arquivo Parquet de um bucket do Amazon S3:

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

Example Criar esquemas automaticamente

Este exemplo cria uma tabela automaticamente com base no esquema de dados:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --if-not-exists
```

Example Validar antes do carregamento

Este exemplo valida sua configuração sem realmente carregar dados:

```
aurora-dsql-loader load \
```

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

Example Retomar uma instância interrompida

Se uma operação de carregamento for interrompida, você poderá retomá-la usando o ID do trabalho da execução anterior:

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

> **Note**
>
> Ao retomar, o carregador ignora a maior parte do trabalho já concluído, mas pode tentar novamente alguns registros. Se sua tabela de destino tiver restrições exclusivas, use a opção `--on-conflict` para lidar com duplicatas. Por exemplo, use `DO NOTHING` para ignorá-las ou `DO UPDATE` para atualizar.

Opções de linha de comando

O Aurora DSQL Loader é compatível com as seguintes opções de linha de comando:

`--endpoint`

(Obrigatório) O endpoint do cluster do Aurora DSQL. Exemplo: *cluster-id*`.dsql.`*region*`.on.aws`

`--source-uri`

(Obrigatório) O caminho para o arquivo de dados. Pode ser um caminho de arquivo local ou um URI do S3 (por exemplo, `s3://`*bucket-name*`/`*file.parquet*).

`--table`

(Obrigatório) O nome da tabela de destino em seu banco de dados do Aurora DSQL.

`--if-not-exists`

(Opcional) Cria automaticamente uma tabela de destino se ela não existir. O carregador infere o esquema a partir dos dados.

`--dry-run`

(Opcional) Valide a configuração e os dados sem precisar carregá-los no banco de dados.

`--resume-job-id`

(Opcional) Retomar uma operação de carregamento interrompida anteriormente usando o ID do trabalho especificado.

`--manifest-dir`

(Opcional) Diretório para armazenar o estado e os manifestos do trabalho, usado para retomada do trabalho.

`--on-conflict`

(Opcional) Especifica como lidar com conflitos ao inserir linhas que violam restrições exclusivas na tabela de destino. Os valores válidos são `error` (retornar um erro), `do-nothing` (ignorar linhas duplicadas) ou `do-update` (atualizar as linhas existentes com novos valores).

Para obter uma lista completa de opções e parâmetros adicionais de configuração, execute:

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

Práticas recomendadas

- Usar dry-run para validação: sempre teste sua configuração de carga com `--dry-run` antes de carregar os dados nas tabelas de produção.
- Definir restrições exclusivas para retomada: se você precisar retomar cargas interrompidas, defina restrições exclusivas em suas tabelas de destino e use a opção `--on-conflict` para lidar com registros já carregados.
- Usar o Parquet para grandes conjuntos de dados: o formato colunar do Parquet normalmente fornece melhor compactação e carregamento mais rápido para grandes conjuntos de dados em comparação com CSV ou TSV.

- Preservar diretórios do manifesto: mantenha o diretório do manifesto para trabalhos de carregamento até confirmar que o carregamento foi concluído com êxito, permitindo a retomada, se necessário.
- Pré-criar tabelas quando possível: defina a tabela de destino com tipos de dados de coluna explícitos e chaves primárias antes de carregar os dados. Os esquemas pré-criados oferecem controle sobre a precisão e a indexação do tipo, o que normalmente resulta em melhor desempenho de consulta em comparação aos esquemas inferidos automaticamente.

Solução de problemas

Erros de autenticação

Verifique se suas credenciais da AWS estão configuradas corretamente e se sua identidade do IAM tem as permissões `dsql:DbConnect` ou `dsql:DbConnectAdmin` necessárias no cluster de destino.

Erros de acesso do S3

Verifique se sua identidade do IAM tem as permissões de leitura apropriadas do S3 para o bucket e os objetos de origem.

Erros de validação de esquema

Ao usar `--if-not-exists`, verifique se o arquivo de dados tem tipos de coluna consistentes. Tipos mistos em uma coluna podem fazer com que a inferência do esquema falhe.

Erros de chave duplicados na retomada

Se você encontrar erros de chave duplicada ao retomar um carregamento, adicione restrições exclusivas à sua tabela de destino para que o carregador possa usar `ON CONFLICT DO NOTHING` para ignorar registros já carregados.

Para obter informações adicionais sobre solução de problemas, consulte o repositório do GitHub do Aurora DSQL Loader.

Caminhos de migração

As seções a seguir descrevem como migrar dados de sistemas de origem comuns para o Aurora DSQL.

Migrar do PostgreSQL

Para migrar dados de um banco de dados do PostgreSQL existente para o Aurora DSQL:

1. Exporte seus dados do PostgreSQL para o formato CVS ou Parquet. Você pode usar o comando `COPY` do PostgreSQL para exportar cada tabela:

```
COPY my_table TO '/path/to/my_table.csv' WITH (FORMAT csv, HEADER true);
```

2. Crie a tabela de destino no Aurora DSQL. Você pode criar o esquema manualmente ou usar o sinalizador `--if-not-exists` do carregador para inferir o esquema a partir dos dados.
3. Carregue os dados exportados usando o Aurora DSQL Loader:

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

> **Tip**
>
> Para migrações maiores, considere exportar para o formato Parquet para obter melhor compactação e carregamento mais rápido. Ferramentas como o DuckDB podem converter arquivos CSV em Parquet de forma eficiente.

Migrar do MySQL

Para migrar dados do MySQL para o Aurora DSQL:

1. Exporte seus dados do MySQL para o formato CSV usando `SELECT INTO OUTFILE` ou uma ferramenta como mysqldump com a opção `--tab`:

```
SELECT * FROM my_table
INTO OUTFILE '/path/to/my_table.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Crie a tabela de destino no Aurora DSQL com os tipos de dados apropriados compatíveis com o PostgreSQL.
3. Carregue os dados exportados usando o Aurora DSQL Loader:

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

> **Note**
>
> O MySQL e o PostgreSQL têm sistemas de tipos de dados diferentes. Revise seu esquema e ajuste os tipos de dados conforme necessário ao criar tabelas no Aurora DSQL.

Carregar do Amazon S3

Se os dados já estão no Amazon S3, você pode armazená-los diretamente sem baixar para o sistema local. O Aurora DSQL Loader oferece suporte nativo a URIs do S3:

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

Verifique se sua identidade do IAM tem a permissão `s3:GetObject` nos objetos de origem.

Usar \copy do PostgreSQL

Se já estiver conectado ao Aurora DSQL por meio de uma sessão do `psql` que gerencia a autenticação do IAM, você pode usar o meta-comando `\copy` do lado do cliente para carregar dados do seu sistema de arquivos local. Ao contrário da declaração `COPY` do lado do servidor, `\copy` lê o arquivo na máquina cliente e transmite os dados pela conexão existente, portanto, nenhum acesso ao arquivo do lado do servidor é necessário. Essa abordagem funciona bem para cargas simples e de thread único.

Example Carregar um arquivo CSV com \copy

```
\copy my_table FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

Ao usar `\copy` diretamente, você é responsável por:

- Gerenciar a paralelização ao carregar vários arquivos ou grandes conjuntos de dados
- Gerenciar o gerenciamento de conexões e a atualização do token de autenticação
- Implementar a lógica de nova tentativa em operações com falha

Práticas recomendadas para transações INSERT

Ao usar declarações `INSERT` para carregar dados no Aurora DSQL, siga estas práticas para melhorar a taxa de throughput e a confiabilidade:

- Agrupe linhas em INSERTs de várias linhas: agrupe várias linhas em uma única declaração `INSERT` para reduzir viagens de ida e volta. Por exemplo, `INSERT INTO my_table VALUES (1, 'a'), (2, 'b'), (3, 'c')` é mais eficiente do que três declarações separadas.
- Use consultas parametrizadas: use declarações preparadas com associação de parâmetros em vez de concatenação de strings. Isso evita riscos de injeção de SQL e permite que o banco de dados reutilize planos de consulta.
- Mantenha as transações pequenas: o Aurora DSQL usa um controle otimista de simultaneidade, portanto, transações grandes que tocam muitas linhas têm maior probabilidade de encontrar conflitos. Use transações de algumas centenas de linhas em vez de milhares.
- Implemente a lógica de repetição: erros transitórios, como conflitos de Controle de simultaneidade otimista (OCC), são esperados em um sistema distribuído. Implemente um recuo exponencial com novas tentativas para transações com falha.
- Paralelize as conexões: abra várias conexões e distribua as inserções entre elas. Cada conexão pode processar um subconjunto diferente de dados simultaneamente.

Para a maioria dos casos de uso, o Aurora DSQL Loader fornece uma abordagem mais simples e robusta para o carregamento de dados.

Recursos adicionais

- Aurora DSQL Loader no GitHub: código-fonte, documentação e rastreamento de problemas

- Gerar um token de autenticação no Amazon Aurora DSQL: saiba mais sobre tokens de autenticação do IAM para o Aurora DSQL
- Acessar o Aurora DSQL com clientes compatíveis com o PostgreSQL: conecte-se ao Aurora DSQL usando vários clientes e ferramentas

IA generativa para o Aurora DSQL

Esta seção fornece instruções detalhadas sobre como usar as ferramentas de IA generativa com o Aurora DSQL

Servidor MCP para AWS Labs Aurora DSQL

Um servidor de protocolo de contexto para modelos (MCP) do AWS Labs para Aurora DSQL

Recursos

- Converter perguntas e comandos legíveis por humanos em consultas SQL estruturadas compatíveis com o Postgres e executá-las no banco de dados Aurora DSQL configurado.
- Somente leitura por padrão, transações habilitadas com `--allow-writes`
- Reutilização de conexão entre solicitações para melhorar a performance
- Acesso integrado à documentação, pesquisa e recomendações de práticas recomendadas do Aurora DSQL

Ferramentas disponíveis

Operações de banco de dados

- readonly_query: execute consultas SQL somente leitura em seu cluster DSQL.
- transact: execute operações de gravação em uma transação (requer `--allow-writes`).
- get_schema: recupera informações do esquema da tabela.

Documentação e recomendações

- dsql_search_documentation: pesquise a documentação do Aurora DSQL.
  - Parâmetros: `search_phrase` (obrigatório), `limit` (opcional).
- dsql_read_documentation: leia páginas específicas de documentação do DSQL.

  - Parâmetros: `url` (obrigatório), `start_index` (opcional), `max_length` (opcional).
- dsql_recommend: receba recomendações sobre as práticas recomendadas do DSQL.
  - Parâmetros: `url` (obrigatórios)

Pré-requisitos

1. Uma conta da AWS com um cluster do Aurora DSQL.
2. Esse servidor MCP só pode ser executado localmente no mesmo host que o seu cliente LLM.
3. Configurar credenciais da AWS com acesso aos serviços da AWS
   - Você precisa de uma conta da AWS com um perfil que inclua essas permissões:
     - `dsql:DbConnectAdmin`: conecte-se aos clusters do DSQL como usuário administrador.
     - `dsql:DbConnect`: conecte-se a clusters do DSQL com perfis de banco de dados personalizados (necessário somente se estiver usando usuários que não sejam administradores).
   - Configurar credenciais da AWS com `aws configure` ou variáveis de ambiente

Instalação

Para a maioria das ferramentas, atualizar a configuração seguindo as instruções de instalação padrão deve ser suficiente.

Há instruções separadas para o Claude Code e o Codex.

Instalação padrão: atualizar o arquivo de configuração do MCP relevante

Usar o **uv**

1. Instale `uv` por meio do Astral ou do LEIAME do GitHub.
2. Instale o Python usando `uv python install 3.10`.

Configure o servidor MCP na configuração do seu cliente MCP (Encontrar o arquivo de configuração do MCP)

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "command": "uvx",
```

```
      "args": [
        "awslabs.aurora-dsql-mcp-server@latest",
        "--cluster_endpoint",
        "[your dsql cluster endpoint, e.g. abcdefghijklmnopqrst234567.dsql.us-
east-1.on.aws]",
        "--region",
        "[your dsql cluster region, e.g. us-east-1]",
        "--database_user",
        "[your dsql username, e.g. admin]",
        "--profile",
        "[your aws profile, e.g. default]"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

Instalação do Windows

Para usuários do Windows, o formato de configuração do servidor MCP é um pouco diferente:

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "disabled": false,
      "timeout": 60,
      "type": "stdio",
      "command": "uv",
      "args": [
        "tool",
        "run",
        "--from",
        "awslabs.aurora-dsql-mcp-server@latest",
        "awslabs.aurora-dsql-mcp-server.exe"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR",
        "AWS_PROFILE": "your-aws-profile",
        "AWS_REGION": "us-east-1"
```

```
      }
    }
  }
}
```

Localizar o arquivo de configuração do cliente do MCP

Para algumas das ferramentas de desenvolvimento de agentes mais comuns, você pode encontrar as configurações do seu cliente do MCP nos seguintes caminhos de arquivo:

- Kiro:
  - Configuração do usuário: `~/.kiro/settings/mcp.json`
  - Configuração do espaço de trabalho: `/path/to/workspace/.kiro/settings/mcp.json`
- Claude Code: consulte Instalação do Claude Code para receber ajuda detalhada sobre a configuração.
  - Configuração do usuário: `~/.claude.json` em `"mcpServers"`
  - Configuração do projeto: `/path/to/project/.mcp.json`
  - Configuração local: `~/.claude.json` em `"projects" -> "path/to/project" -> "mcpServers"`
- Cursor:
  - Global: `~/.cursor/mcp.json`
  - Projeto: `/path/to/project/.cursor/mcp.json`
- Codex: `~/.codex/config.toml`
  - Cada servidor MCP é configurado com uma tabela `[mcp_servers.<server-name>]` no arquivo de configuração. Consulte as instruções de instalação do Custom Codex.
- Warp:
  - Edição de arquivos: `~/.warp/mcp_settings.json`
  - Editor de aplicações: `Settings > AI > Manage MCP Servers` e cole o json.
- CLI do Amazon Q Developer: `~/.aws/amazonq/mcp.json`
- Cline: normalmente, um caminho aninhado do VS Code: `~/.vscode-server/path/to/cline_mcp_settings.json`

Claude Code

Pré-requisitos

Importante: o gerenciamento do servidor MCP só está disponível por meio da experiência do terminal da CLI do Claude Code, não do modo de painel nativo do VS Code.

Instale primeiro a CLI do Claude Code seguindo o processo de instalação nativo recomendado do Claude.

Escolher o escopo correto

O Claude Code oferece três escopos diferentes: local (padrão), projeto e usuário, e detalha qual escopo escolher com base na sensibilidade da credencial e na necessidade de compartilhamento. Consulte a documentação do Claude Code sobre os escopos de instalação do MCP para receber mais detalhes.

1. Os servidores com escopo local representam o nível de configuração padrão e são armazenados em `~/.claude.json` no caminho do seu projeto. Ambos são privados para você e só podem ser acessados no diretório atual do projeto. Esse é o `scope` padrão ao criar servidores MCP.
2. Os servidores com escopo de projeto permitem a colaboração em equipe e, ao mesmo tempo, só podem ser acessados em um diretório de projetos. Servidores com escopo de projeto incluem um arquivo `.mcp.json` na raiz do seu projeto. Esse arquivo foi projetado para ser verificado no controle de versão, garantindo que todos os membros da equipe tenham acesso às mesmas ferramentas e serviços do MCP. Quando você adiciona um servidor com escopo de projeto, o Claude Code cria ou atualiza automaticamente esse arquivo com a estrutura de configuração apropriada.
3. Os servidores com escopo de usuário são armazenados em `~/.claude.json` e oferecem acessibilidade entre projetos. Eles os disponibilizam em todos os projetos em sua máquina, permanecendo privados da sua conta de usuário.

Usar a CLI do Claude (recomendado)

O uso de uma sessão de CLI `claude` interativa permite uma melhor experiência de solução de problemas, portanto, esse é o caminho recomendado.

```
claude mcp add amazon-aurora-dsql \
  --scope [one of local, project, or user] \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Solução de problemas: usar o Claude Code com o Bedrock em uma conta diferente da AWS

Se você configurou o Claude Code com uma conta ou perfil da AWS do Bedrock diferente do perfil necessário para se conectar ao seu cluster dsql, você precisará fornecer argumentos de ambiente adicionais:

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Modificação direta no arquivo de configuração

O Claude Code requer nomenclatura alfanumérica, então recomendamos nomear seu servidor: `aurora-dsql-mcp-server`.

Escopo local

Atualize `~/.claude.json` no campo específico do projeto `mcpServers`:

```
{
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Escopo do projeto

Atualize `/path/to/project/root/.mcp.json` no campo `mcpServers`:

```
{
  "mcpServers": {}
}
```

Escopo do usuário

Atualize `~/.claude.json` no campo específico do projeto `mcpServers`:

```
{
  "mcpServers": {}
}
```

Codex

Opção 1: CLI do Codex

Se você tiver a CLI do Codex instalada, poderá usar o comando codex mcp para configurar seus servidores MCP.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Opção 2: config.toml

Para um controle mais refinado sobre as opções do servidor MCP, você pode editar manualmente o arquivo de configuração `~/.codex/config.toml`. Cada servidor MCP é configurado com uma tabela `[mcp_servers.<server-name>]` no arquivo de configuração.

```
[mcp_servers.amazon-aurora-dsql]
command = "uvx"
args = [
  "awslabs.aurora-dsql-mcp-server@latest",
  "--cluster_endpoint", "<DSQL_CLUSTER_ID>.dsql.<AWS_REGION>.on.aws",
  "--region", "<AWS_REGION>",
  "--database_user", "<DATABASE_USERNAME>"
]

[mcp_servers.amazon-aurora-dsql.env]
FASTMCP_LOG_LEVEL = "ERROR"
```

Verificar a instalação

Para CLI do Amazon Q Developer, CLI do Kiro, CLI/TUI do Claude ou CLI/TUI do Codex, execute `/mcp` para ver o status do servidor MCP.

Para o IDE do Kiro, você também pode navegar até a guia `MCP SERVERS` do Painel do Kiro, que mostra todos os servidores MCP configurados e seus indicadores de status de conexão.

Opções de configuração do servidor

`--allow-writes`

Por padrão, o servidor mcp do dsql não permite operações de gravação ("modo somente leitura"). Todas as invocações da ferramenta de transação falharão nesse modo. Para usar a ferramenta de transação, permita gravações transmitindo o parâmetro `--allow-writes`.

Recomendamos usar o acesso com privilégio mínimo ao se conectar ao DSQL. Por exemplo, os usuários devem usar um perfil que seja somente leitura quando possível. O modo somente leitura tem uma aplicação no lado do cliente, em regime de melhor esforço, para rejeitar mutações.

`--cluster_endpoint`

Esse é um parâmetro obrigatório para especificar o cluster ao qual se conectar. Esse deve ser o endpoint completo do seu cluster, por exemplo, `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.

`--database_user`

Esse é um parâmetro obrigatório para especificar o usuário com o qual se conectar. Por exemplo: `admin` ou `my_user`. Observe que as credenciais da AWS que você está usando devem ter permissão para fazer login como esse usuário. Para acessar mais informações sobre como configurar e usar perfis de banco de dados no DSQL, consulte Usar perfis de banco de dados com perfis do IAM.

`--profile`

Você pode especificar o perfil da aws a ser usado para suas credenciais. Observe que isso não é aceito na instalação do docker.

O uso da variável de ambiente `AWS_PROFILE` em sua configuração do MCP também é aceito:

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Se nenhum for fornecido, o servidor MCP usa como padrão o perfil “padrão” em seu arquivo de configuração da AWS.

`--region`

Esse é um parâmetro obrigatório para especificar a região do seu banco de dados DSQL.

`--knowledge-server`

Parâmetro opcional para especificar o endpoint remoto do servidor MCP para ferramentas de conhecimento do DSQL (pesquisa, leitura e recomendações de documentação). Por padrão, ele é pré-configurado.

Exemplo:

```
--knowledge-server https://custom-knowledge-server.example.com
```

Observação: por questões de segurança, use somente endpoints confiáveis do servidor de conhecimento. O servidor deve ser um endpoint HTTPS.

`--knowledge-timeout`

Parâmetro opcional para especificar o tempo limite em segundos para solicitações ao servidor de conhecimento.

Padrão: `30.0`

Exemplo:

```
--knowledge-timeout 60.0
```

Aumente esse valor se você tiver tempos limite ao acessar a documentação em redes lentas.

Direcionamento do Aurora DSQL: skills e powers

Esta seção descreve como configurar o direcionamento da IA para o Aurora DSQL usando skills e powers. Esses arquivos de configuração baseados em Markdown oferecem contexto e orientações que os assistentes de IA aplicam automaticamente ao gerar código para melhorar a qualidade do desenvolvimento agêntico.

Visão geral

Skills e powers são recursos modulares que ampliam a funcionalidade do assistente de IA para o Aurora DSQL. Eles reúnem instruções, metadados e recursos que os assistentes de IA usam automaticamente ao trabalhar com bancos de dados do Aurora DSQL.

Por que usar skills e powers

Os recursos de skills e powers oferecem vários benefícios importantes para o desenvolvimento do Aurora DSQL:

- Especialização de assistentes de IA: esses recursos oferecem conhecimentos de domínios específicos para o Aurora DSQL, como práticas recomendadas, padrões SQL compatíveis com Postgres e otimizações distribuídas de bancos de dados.
- Redução da repetição: cria-se uma vez e usa-se automaticamente. Isso elimina a necessidade de fornecer repetidamente a mesma orientação em várias conversas.
- Eficiência contextual: em vez de consumir o contexto antecipadamente, as skills são carregadas sob demanda. A IA carrega as informações em etapas, conforme necessário.
- Aprendizado contínuo: à medida que os recursos do Aurora DSQL evoluem e as skills são atualizadas, os assistentes de IA acessam padrões atualizados automaticamente.

Caminhos de configuração recomendados

Escolha o caminho de configuração que corresponda ao seu ambiente de desenvolvimento:

- the section called “CLI de Skills” (independente do agente)
- the section called “Kiro Power”
- the section called “Claude Skill”
- the section called “Gemini Skill”
- the section called “Codex Skill”

A skill do DSQL também pode ser usada com outros agentes de codificação de IA copiando a pasta skill no diretório `rules` ou `skills` da ferramenta.

CLI de Skills

A skill do DSQL pode ser instalada usando a CLI de skills. Esse método de configuração independente de agente funciona com a maioria dos assistentes de codificação de IA e permite que você instale a skill em vários agentes ao mesmo tempo.

Configuração

Execute o seguinte comando para instalar a skill do Aurora DSQL:

```
npx skills add awslabs/mcp --skill dsql
```

A CLI oferecerá orientações para você:

- Selecionar agentes: escolha em quais agentes instalar (Kiro, Claude Code, Cursor, Copilot, Gemini, Codex, Roo, Cline, OpenCode, Windsurf etc.)
- Definir o escopo da instalação: escolha entre:
  - Projeto: instale no diretório atual (dedicado ao seu projeto).
  - Global: instale no diretório inicial (disponível em todos os projetos).
- Método de instalação: escolha entre:
  - Link simbólico (recomendado): única fonte confiável, atualizações fáceis.
  - Cópia para todos os agentes: cópias independentes para cada agente.

Gerenciar habilidades

Verifique e atualize as skills a qualquer momento usando:

```
npx skills check
npx skills update
```

Kiro Power

Os powers do Kiro são pacotes unificados que reúnem ferramentas MCP com experiência em framework e instruções de direcionamento. Cada power inclui um documento de ponto de entrada que explica as ferramentas MCP e os acionadores de ativação disponíveis, a configuração do servidor MCP e orientações adicionais específicas do fluxo de trabalho, carregadas sob demanda.

Os powers são ativados dinamicamente com base no contexto do usuário. Em vez de carregar todas as ferramentas antecipadamente, os powers mantêm um uso básico próximo de zero até que as palavras-chave relevantes acionem a ativação.

Configuração

Para configurar um power do Kiro para o Aurora DSQL:

1. Instale diretamente do Kiro Powers Registry.

2. Isso conduzirá você ao power no IDE, onde você pode:
   - Selecionar o botão Try Power. Sugerido para usuários que desejam que a IA oriente a configuração do servidor MCP ou uma experiência de integração interativa com o Aurora DSQL para criar um cluster.
   - Abrir um novo chat do Kiro e pergunte qualquer coisa relacionada ao Aurora DSQL. Opcionalmente, atualize a configuração de MCP com os detalhes do cluster existente para testar a conexão do servidor MCP e possibilitar que ela seja usada imediatamente com o power. O agente do Kiro ativará automaticamente o power se considerar que ele é importante para concluir a tarefa do usuário.

Claude Skill

As skills do Claude são capacidades modulares que ampliam a funcionalidade do Claude. Cada skill inclui instruções, metadados e recursos opcionais que o Claude usa automaticamente quando relevante. As skills são baseadas no sistema de arquivos e são carregadas sob demanda para minimizar o uso de contexto.

Configuração simples com a CLI de Skills

A skill pode ser instalada no Claude Code usando a the section called “CLI de Skills”. Para especificar somente o Claude Code como o agente a ser instalado, use:

```
npx skills add awslabs/mcp --skill dsql --agent claude-code
```

Alternativa: configuração direta usando um clone do Git

A configuração alternativa usa um clone esparso do diretório dsql-skill e cria um link simbólico desse clone na pasta `~/.claude/skills/`. Isso permite que alterações na skill sejam removidas sempre que for necessário atualizar a skill.

Pré-requisitos

- Git instalado

Etapas da configuração

1. Criar um diretório de repositórios base

```
mkdir -p .dsql_skill_repos
```

2. Usar um clone esparso da skill do repositório MCP

Clone somente a pasta `dsql-skill` (nenhum outro arquivo):

```
cd .dsql_skill_repos
git clone --filter=blob:none --no-checkout https://github.com/awslabs/mcp.git
cd mcp
git sparse-checkout init --cone
git sparse-checkout set src/aurora-dsql-mcp-server/skills/dsql-skill
git checkout
cd ../..
```

3. Criar um link simbólico da skill no diretório de skills

Adicione o diretório de skills (padrão: escopo global/de usuário):

```
mkdir -p ~/.claude/skills
```

> **Note**
>
> Se você quiser definir o escopo da skill para um projeto, use o diretório raiz `.claude/skills/` do projeto em vez disso.

Criar um link simbólico:

```
ln -s "$(pwd)/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/dsql-skill"
 ~/.claude/skills/dsql-skill
```

4. Verifique a configuração

```
# Should show SKILL.md and other skill files
ls -la ~/.claude/skills/dsql-skill/
```

5. Verificar o uso da skill

Assim que a skill estiver configurada, você terá um novo comando de skill: `/dsql`. Talvez seja necessário reiniciar o Claude Code depois de adicionar a skill, para que ela seja detectada. Você pode usar esse comando na CLI ou no painel do Claude Code, conforme desejado.

Atualizar a skill

Para obter as alterações mais recentes do repositório:

```
cd .dsql_skill_repos/mcp
git pull
```

Estrutura de diretório

Depois de configurar uma skill global, você deve ver estes diretórios:

```
.dsql_skill_repos/
### mcp/                              # Sparse git checkout
    ### src/
        ### aurora-dsql-mcp-server/
            ### skills/
                ### dsql-skill/
                    ### SKILL.md
                    ### ...

~/.claude/
### skills/
    ### dsql-skill -> /path/to/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/
dsql-skill
```

> **Note**
>
> Adicione `.dsql_skill_repos/` ao seu `.gitignore` se você não quiser rastreá-lo. O checkout esparso mantém apenas a pasta de skills, minimizando o uso do disco.

Gemini Skill

Para adicionar a skill do Aurora DSQL diretamente no Gemini, escolha um escopo: `workspace` (contido no projeto) ou `user` (padrão, global) e use o instalador de skills.

Configuração

```
gemini skills install https://github.com/awslabs/mcp.git --path src/aurora-dsql-mcp-
server/skills/dsql-skill --scope $SCOPE
```

Substitua `$SCOPE` por `workspace` ou `user`.

Você pode então usar o comando de skill `/dsql` com o Gemini, e o Gemini detectará automaticamente quando a skill deve ser usada.

Codex Skill

Use o instalador de skill da CLI ou TUI do Codex usando a skill `$skill-installer`.

Configuração

```
$skill-installer install dsql skill: https://github.com/awslabs/mcp/tree/main/src/
aurora-dsql-mcp-server/skills/dsql-skill
```

Reinicie o Codex para adquirir a skill. A skill pode então ser ativada usando `$dsql`.

Comece a usar o Editor de consultas do Aurora DSQL

Com o Editor de consultas do Aurora DSQL, você pode se conectar com segurança aos seus clusters do Aurora DSQL e executar consultas SQL diretamente do Console de Gerenciamento da AWS sem instalar nem configurar clientes externos. Ele oferece um espaço de trabalho intuitivo com destaque de sintaxe integrado, preenchimento automático e assistência inteligente de código. É possível explorar rapidamente objetos de esquema, desenvolver e executar consultas SQL e visualizar resultados, tudo em uma única interface.

Este tópico mostra as etapas para se conectar a um cluster, executar consultas, visualizar resultados e explorar recursos avançados, como planos de execução.

> **Note**
>
> O Editor de consultas está disponível em todas as regiões onde o Aurora DSQL é aceito. Para ter mais informações sobre disponibilidade regional, consulte Serviços regionais da AWS.

Pré-requisitos

Antes de começar, certifique-se de que os seguintes requisitos são atendidos:

- Você tem pelo menos um cluster do Aurora DSQL disponível. Para ver mais detalhes sobre como criar clusters, consulte Etapa 1: criar um cluster do Aurora DSQL de região única.
- O endpoint do cluster pode ser acessado publicamente. O Editor de Consultas não aceita clusters com acesso público bloqueado por políticas baseadas em recursos ou clusters gerenciados por meio de endpoints da VPC. Para ver mais detalhes sobre restrições de acesso, consulte Bloquear o acesso público com políticas baseadas em recursos no Aurora DSQL e Gerenciar e conectar-se com clusters do Amazon Aurora DSQL usando o AWS PrivateLink.
- Seu usuário ou perfil do IAM tem as permissões necessárias para acessar e se conectar ao cluster. Para obter detalhes sobre permissões, consulte Usar perfis de banco de dados e autenticação do IAM.

Trabalhar com o Editor de consultas

Abra o editor de consulta

Como abrir o Editor de consultas

1. Abra o console do Aurora DSQL.
2. No painel de navegação, selecione Query Editor (Editor de consultas).

Como alternativa, na página Clusters, selecione o cluster que você deseja consultar e escolha Conectar com o Editor de consultas para iniciar o editor diretamente.

> **Note**
>
> O trabalho e o estado da conexão não são salvos. Se você sair do console do Aurora DSQL, fechar a guia do navegador ou sair, suas conexões, o texto da consulta e os resultados serão perdidos.

Conectar-se a um cluster

Como se conectar a um cluster

1. Se não houver nenhuma conexão de cluster, o editor exibirá Nenhum cluster foi conectado. Escolha Conectar ou selecione + (Adicionar) no painel Explorador de clusters para se conectar a um cluster existente.

2. (Opcional) Conecte-se a vários clusters ou ao mesmo cluster usando perfis diferentes.

Explorar objetos de cluster

O Explorador de clusters exibe todas as conexões de cluster disponíveis e permite que você procure objetos, como bancos de dados, esquemas, tabelas e visualizações. Ele também fornece ações comuns, como Atualizar, Criar tabela e outras opções específicas do contexto.

Executar consultas

Para executar uma consulta

1. No painel da guia do Editor de consultas, insira sua instrução SQL. Por exemplo:

```
SELECT * FROM public.orders LIMIT 10;
```

2. Verifique o contexto do cluster ativo exibido no canto superior direito da guia de consulta. Isso indica a conexão do cluster associada à guia de consulta atual.
3. (Opcional) Use o menu suspenso de conexão para revisar todas as conexões disponíveis ou alternar para um cluster diferente. Alterar as atualizações de conexão onde suas consultas nessa guia são executadas.
4. Para executar a consulta, escolha Executar.

> **Note**
>
> Cada consulta pode exibir até 10 mil linhas no painel de resultados. Para conjuntos de dados maiores, refine sua consulta com filtros ou limites.

Analise os resultados e os planos de execução

Depois que a consulta for executada, revise a saída no painel Resultados na parte inferior do editor. Por padrão, cada execução de consulta exibe a guia Resultados (Tabela), mostrando a saída da consulta tabular.

Para recuperar o plano de execução da consulta, execute `EXPLAIN ANALYZE` ou `EXPLAIN ANALYZE VERBOSE` para ter insights adicionais sobre a performance da consulta. Para ter mais informações sobre planos de execução, consulte Ler os planos EXPLAIN do Aurora DSQL.

> **Tip**
>
> O comando `EXPLAIN ANALYZE VERBOSE` apresenta estimativas de uso de DPU, como valores de computação, leitura, gravação e total de DPU, e oferece visibilidade imediata dos recursos consumidos por instruções SQL individuais.

Editores de consultas: usar o JupyterLab com o Aurora DSQL

Este guia fornece instruções passo a passo sobre como conectar e consultar o Amazon Aurora DSQL usando o JupyterLab com o Python. O JupyterLab é um ambiente computacional interativo conhecido que combina código, texto e visualizações em um único documento. É amplamente utilizado para aplicações de pesquisa e ciência de dados.

As instruções abaixo abordarão os conceitos básicos do uso do Aurora DSQL em uma instalação local do JupyterLab e no uso do Amazon SageMaker AI, um serviço de machine learning totalmente gerenciado que oferece um ambiente hospedado com uma interface de usuário para fluxos de trabalho de dados.

Introdução

Requisitos

- Um cluster do Aurora DSQL
- Credenciais da AWS configuradas (somente instalação local)
- Python versão 3.9 ou posterior (somente instalação local)

Usar o JupyterLab local

Para começar a utilizar o JupyterLab, os usuários devem primeiro instalar a aplicação usando o pip do Python:

```
pip install jupyterlab
```

O JupyterLab pode então ser aberto executando **`jupyter lab`**. Isso abrirá a aplicação JupyterLab em localhost:8888, acessível em um navegador. Tenha as credenciais da AWS configuradas em seu ambiente local antes de continuar.

Usar o Amazon SageMaker AI

No console da AWS, vá para a página do console do Amazon SageMaker AI e depois para a seção Bloco de anotações em Aplicações e IDEs. Nela, você pode selecionar Criar instância do bloco de anotações para começar a criar um ambiente do SageMaker. Selecione um tipo de instância e uma plataforma antes de clicar em Criar instância do bloco de anotações.

Consulte a documentação de configuração do Amazon SageMaker AI para acessar mais informações sobre opções de configuração e instância.

> **Note**
>
> Aviso: o uso do Amazon SageMaker AI pode gerar cobranças na conta da AWS.

Depois que a instância do SageMaker estiver ativa, você poderá abri-la na seção Instâncias do bloco de anotações com Abrir JupyterLab. Antes de começar a usar o Aurora DSQL no seu bloco de anotações, você deve conceder acesso ao seu cluster do DSQL no perfil do IAM da instância do SageMaker. A maneira mais simples de fazer isso é seguir o link para o perfil do IAM na página da instância do bloco de anotações. A partir daí, você pode editar as políticas anexadas ao seu perfil do IAM do SageMaker. Consulte Autenticação e autorização para acessar mais informações sobre como configurar uma política do IAM para permitir acesso ao Aurora DSQL.

Conectar-se ao Aurora DSQL usando o JupyterLab

Depois de configurar uma instância do JupyterLab, as etapas para se conectar ao Aurora DSQL são as mesmas localmente e no SageMaker AI. Crie um caderno do Python 3 vazio, no qual você pode incluir células com código Python.

Em uma célula Python, baixe o certificado raiz da Amazon da loja oficial de confiança:

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Para se conectar ao Aurora DSQL, primeiro instale o conector do Aurora DSQL para Python e o driver Psycopg em uma célula Python e, depois, importe-o:

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Com o conector importado, você pode criar uma configuração do DSQL e se conectar. O conector do Aurora DSQL para Python processará automaticamente a criação de um token de autenticação em cada conexão.

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

conn = dsql.connect(**config)
```

Ao executar o código, agora você deve ter uma conexão Psycopg com o Aurora DSQL. Depois, você poderá executar consultas usando o cursor Psycopg e fornecendo sua consulta SQL. Consulte a documentação do Psycopg para acessar mais informações sobre como usar a Psycopg com um banco de dados compatível com Postgres. Essa consulta vai gerar uma lista de tuplas em `results_list`.

```
with conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

Depois, você pode usar frameworks Python, como o Pandas, para analisar ou visualizar os resultados da consulta, por exemplo:

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

Caderno de exemplo

Um caderno de amostra usando o Aurora DSQL está disponível no repositório de exemplos do Aurora DSQL.

Outras fontes de leitura

Documentação de configuração do Amazon SageMaker AI

Conector do Aurora DSQL para Python

Documentação do Pandas

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 Console de gerenciamento da AWS, 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. Configurar 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.

Restaurar clusters de região única

Para restaurar um cluster de região única do Aurora DSQL, use o console: 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. Para obter instruções detalhadas, consulte Restaurar um cluster Aurora DSQL de região única.

Restaurar clusters multirregionais

A restauração de um cluster multirregional do Aurora DSQL tem suporte tanto pelo console: https://console.aws.amazon.com/backup quanto pelo AWS CLI. Para obter instruções detalhadas, consulte Restaurar um cluster multirregional do 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, você deverá garantir que haja uma cópia idêntica do seu backup em todas as Regiões da AWS para seus clusters multirregionais. Se você ainda não tiver essas cópias, deverá primeiro copiar o backup para outra Região da AWS que seja compatível com clusters multirregionais.

Recomendamos criar cópias de backup em Regiões da AWS importantes para permitir opções robustas de recuperação de desastres e atender aos requisitos de conformidade. Para ver as versões disponíveis Regiões da AWS para o Aurora DSQL, consulte the section called “Disponibilidade do Região da AWS”.

Para obter instruções detalhadas sobre essas etapas, consulte a documentação de restauração do Amazon Aurora DSQL.

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.

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. Analise 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
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.
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

Nome da métrica do CloudWatch	Métrica	Dimensão: ResourceId	Unidade	Descrição
				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 Console de gerenciamento da AWS 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 da 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 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 terceirizados 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

- Políticas gerenciadas pela AWS para o Amazon Aurora DSQL
- Proteção de dados no Amazon Aurora DSQL
- Criptografia de dados para o Amazon Aurora DSQL
- Gerenciamento de identidade e acesso para o Aurora DSQL
- Políticas baseadas em recursos para o Aurora DSQL
- Usar perfis vinculados ao serviço no Aurora DSQL
- Usar chaves de condição do IAM com o Amazon Aurora DSQL
- Resposta a incidentes no Amazon Aurora DSQL
- Validação de conformidade para o Amazon Aurora DSQL

- Resiliência no Amazon Aurora DSQL
- 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 autônoma 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.

Não é possível 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 saber mais, consulte AWSPolíticas gerenciadas pela no Guia 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 com essas permissões podem:

- Criar, excluir e atualizar clusters do Aurora DSQL, inclusive clusters multirregionais.

- Gerenciar políticas em linha para o cluster (criar, visualizar, atualizar e excluir políticas)
- Adicionar e remover tags dos clusters.
- Listar clusters e visualizar informações sobre clusters individuais.
- Ver as tags anexadas aos clusters do Aurora DSQL.
- Conectar-se ao banco de dados como qualquer usuário, inclusive administrador.
- Realizar operações de backup e restauração para clusters do Aurora DSQL, como iniciar, interromper e monitorar tarefas de backup e restauração.
- Usar chaves do AWS KMS gerenciadas pelo cliente para criptografia de cluster.
- Visualizar qualquer métrica do CloudWatch para a conta
- Usar o AWS Fault Injection Service (AWS FIS) para injetar falhas em clusters do Aurora DSQL para testes de tolerância a falhas.
- Criar perfis vinculados ao serviço para o serviço `dsql.amazonaws.com`, 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.
- `fis`: concede permissões ao AWS Fault Injection Service (AWS FIS) para injetar falhas em clusters do Aurora DSQL para testes de tolerância a falhas.

Você pode encontrar a política `AmazonAuroraDSQLFullAccess` no console do IAM e 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. Elas podem ver as marcações anexadas aos clusters do Aurora DSQL e visualizar políticas em linha para cluster. 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 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 Console de gerenciamento da AWS. As entidades principais com essas permissões podem:

- Criar, excluir e atualizar clusters do Aurora DSQL, inclusive clusters multirregionais, com o console.
- Gerenciar políticas em linha para o cluster por meio do console (criar, visualizar, atualizar e excluir políticas)
- Listar clusters e visualizar informações sobre clusters individuais.
- Ver tags em qualquer recurso em sua conta.
- Conectar-se ao banco de dados como qualquer usuário, inclusive administrador.

- Realizar operações de backup e restauração para clusters do Aurora DSQL, como iniciar, interromper e monitorar tarefas de backup e restauração.
- Usar chaves do AWS KMS gerenciadas pelo cliente para criptografia de cluster.
- Iniciar o AWS CloudShell por meio do Console de gerenciamento da AWS.
- Visualizar quaisquer métricas do CloudWatch em sua conta.
- Usar o AWS Fault Injection Service (AWS FIS) para injetar falhas em clusters do Aurora DSQL para testes de tolerância a falhas.
- Criar perfis vinculados ao serviço para o serviço `dsql.amazonaws.com`, 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 Console de gerenciamento da AWS.
- `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.
- `fis`: concede permissões para usar o AWS FIS para injetar falhas em clusters do Aurora DSQL para testes de tolerância a falhas.

- A `access-analyzer:ValidatePolicy` concede permissão para o linter no editor de políticas, que fornece feedback em tempo real sobre erros, avisos e problemas de segurança na política atual.
- `fis`: concede permissões ao AWS Fault Injection Service (AWS FIS) para injetar falhas em clusters do Aurora DSQL para testes de tolerância a falhas.

Você pode encontrar a política `AmazonAuroraDSQLConsoleFullAccess` no console do IAM e 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 e AmazonAuroraDSQLConsoleFullAccess	Suporte adicionado para integração do AWS Fault Injection Service (AWS FIS) com o Aurora DSQL. Isso permite que você injete falhas em clusters do Aurora DSQL de região única e multirregionais para testar a tolerânci	19 de agosto de 2025

Alteração	Descrição	Data
	a a falhas de suas aplicaçõe s. Você pode criar modelos de experimentos no console do AWS FIS para definir cenários de falha e direcionar clusters específicos do Aurora DSQL para testes. Para ter mais informações sobre essas políticas, consulte AmazonAuroraDSQLFu llAccess e AmazonAur oraDSQLConsoleFullAccess.	
Atualização do AmazonAur oraDSQLFullAccess, do AmazonAuroraDSQLRe adOnlyAccess e do AmazonAuroraDSQLCo nsoleFullAccess	Adição de suporte à política baseada em recursos (RBP) com novas permissões: `PutClusterPolicy` , `GetClusterPolicy` e `DeleteClusterPolicy` . Essas permissões possibilitam o gerenciamento de políticas em linha anexadas clusters do Aurora DSQL para controle de acesso refinado. Para acessar mais informaçõ es, consulte AmazonAur oraDSQLFullAccess, AmazonAuroraDSQLRe adOnlyAccess e AmazonAur oraDSQLConsoleFullAccess.	15 de outubro de 2025

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 Usar perfis vinculados ao serviço no 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 saber mais sobre a privacidade de dados, consulte as Data Privacy FAQ. Para obter mais informações sobre a proteção de dados na Europa, consulte o artigo do blog Shared Responsibility Model and GDPR no Blog de segurança da .

Para fins de proteção de dados, é recomendável que você proteja as credenciais e configure usuários individuais com o Centro de Identidade do AWS IAM 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 sensíveis 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 Processo de assinatura do Signature versão 4 na 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.

Conformidade com os FIPS

Os endpoints de plano de dados do Aurora DSQL (endpoints de cluster usados para conexões de banco de dados) usam módulos criptográficos validados pelo FIPS 140-2 por padrão. Não são necessários endpoints FIPS separados para conexões de cluster.

Para operações de ambiente de gerenciamento, o Aurora DSQL fornece endpoints FIPS exclusivos em regiões aceitas. Para acessar mais informações sobre endpoints FIPS do ambiente de gerenciamento, consulte Aurora DSQL endpoints and quotas, na Referência geral da AWS.

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 Direct Connect. Para ter mais informações, consulte O que é o 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.

Proteção de dados em regiões testemunhas

Quando você cria um cluster multirregional, uma região testemunha ajuda a permitir a recuperação automática de falhas participando da replicação síncrona de transações criptografadas. Se um cluster emparelhado ficar indisponível, a região testemunha permanecerá disponível para validar e processar gravações no banco de dados, garantindo que não haja perda de disponibilidade.

As regiões testemunhas protegem seus dados por meio desses recursos de design:

- A região testemunha recebe e armazena somente logs de transações criptografadas. Ela não hospeda, armazena nem transmite suas chaves de criptografia.
- A região testemunha se concentra exclusivamente em registrar transações de gravação e funções de quórum. Por padrão, ela não consegue ler seus dados.
- A região testemunha opera sem endpoints de conexão de cluster ou processadores de consulta. Isso impede o acesso ao banco de dados do usuário.

Para obter mais informações sobre as regiões testemunhas, consulte Configurar clusters multirregionais.

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
- Criptografia em repouso no Aurora DSQL
- Usar chaves do AWS KMS e de dados com o Aurora DSQL
- Autorizar o uso da AWS KMS key para o Aurora DSQL
- Contexto de criptografia do Aurora DSQL
- Monitorar a interação do Aurora DSQL com o AWS KMS
- Criar um cluster criptografado do Aurora DSQL
- Remover ou atualizar uma chave para um cluster do Aurora DSQL
- Considerações sobre criptografia com o Aurora DSQL

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
- Usar chaves de cluster com o Aurora DSQL
- Armazenamento de chaves de cluster em cache

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.

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, é necessário 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 Console de gerenciamento da AWS ou da AWS CLI.

Console

Como criar um cluster criptografado usando o Console de gerenciamento da AWS

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 de propriedade da 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 Console de gerenciamento da AWS 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 Console de gerenciamento da AWS

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 Console de gerenciamento da AWS 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 Console de gerenciamento da AWS, 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

- Público
- Autenticação com identidades
- 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

A forma como você usa o AWS Identity and Access Management (IAM) difere com base em seu perfil:

- Usuário do serviço: solicite permissões ao seu administrador se você não conseguir acessar os atributos (consulte Solução de problemas de identidade e acesso do Amazon Aurora DSQL).
- Administrador do serviço: determine o acesso do usuário e envie solicitações de permissão (consulte Como o Amazon Aurora DSQL funciona com o IAM)
- Administrador do IAM: escreva políticas para gerenciar o acesso (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. Você precisa se autenticar como o Usuário raiz da conta da AWS, como um usuário do IAM ou assumindo um perfil do IAM.

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade, como o Centro de Identidade do AWS IAM (Centro de Identidade do IAM), autenticação única ou credenciais do Google/Facebook. Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS.

Para acesso programático, a AWS oferece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Usuário-raiz Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login única chamada usuário-raiz da Conta da AWS, que tem acesso total a todos os recursos e Serviços da AWS. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em Tarefas que exigem credenciais de usuário-raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários humanos usem a federação com um provedor de identidades para acessar a Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório empresarial, de um provedor de identidades da web ou do Directory Service que acessa Serviços da AWS usando credenciais de uma fonte de identidade. As identidades federadas assumem funções que oferecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos Centro de Identidade do AWS IAM. Para saber mais, consulte O que é o IAM Identity Center? no Guia do usuário do Centro de Identidade do AWS IAM.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para saber mais, consulte Exigir que os usuários humanos usem a

federação com um provedor de identidade para acessar a AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso de usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. É possível assumir um perfil alternando de um usuário para um perfil do IAM (console) ou chamando uma operação de API AWS CLI ou AWS. Para saber mais, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em Acesso a recursos entre contas no IAM 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 define permissões quando associada a uma identidade ou a um recurso. A AWS avalia essas políticas quando a entidade principal faz uma solicitação. A maioria das políticas é armazenada na AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você anexa a uma identidade (usuário, grupo ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais 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 políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas 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. Entre os exemplos estão políticas de confiança de perfil do IAM e 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. É necessário especificar uma entidade principal em uma política baseada em recursos.

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.

Outros tipos de política

A AWS permite outros tipos de política capazes de definir o máximo de permissões concedidas por tipos de política mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de Controle de Serviços (SCPs): as SCPs especificam o número máximo de permissões para uma organização ou uma unidade organizacional no AWS Organizations. Para saber mais, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations.
- Políticas de controle de recursos (RCPs): definem o número máximo de permissões disponíveis para recursos em suas contas. Consulte mais informações em Resource control policies (RCPs) no Guia do usuário do AWS Organizations.
- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, 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

Recurso do IAM	Suporte ao Aurora DSQL
Políticas baseadas em identidade	Sim
Políticas baseadas em atributos	Sim
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
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 podem ser anexados 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. 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

Compatível com políticas baseadas em recursos: sim

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. É necessário especificar uma entidade principal em uma política baseada em recursos. Os principais podem incluir contas, usuários, funções, usuários federados ou serviços da AWS. Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar as políticas gerenciadas da AWS do IAM em uma política baseada em recursos.

Para saber como criar e gerenciar políticas baseadas em recursos para clusters do Aurora DSQL, consulte Políticas baseadas em recursos para o Aurora DSQL.

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. 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. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Para ações que não oferecem compatibilidade com permissões em nível de recurso, 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` especifica quando as instruções são executadas com base em critérios definidos. 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. 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 permissões com base em atributos chamados de tags. Você pode anexar tags a entidades do IAM e recursos da

AWS e, em seguida, projetar políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag no recurso.

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 saber mais 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 por atributo (ABAC) no Guia do usuário do IAM.

Usar credenciais temporárias com o Aurora DSQL

Compatível com credenciais temporárias: sim

As credenciais temporárias dão acesso de curto prazo aos recursos da AWS e são criadas automaticamente quando você usa a federação ou alterna os perfis. A AWS recomenda a você gerar credenciais temporárias dinamicamente, em vez de usar chaves de acesso de longo prazo. Para ter mais informações, consulte Credenciais de segurança temporárias no IAM e Serviços da Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Permissões de entidade principal entre serviços para o Aurora DSQL

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

As sessões de acesso direto (FAS) usam as permissões da entidade principal chamando um AWS service (Serviço da AWS), bem como o AWS service (Serviço da AWS) solicitante, para fazer solicitações a serviços subsequentes. 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 saber

mais, 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 vinculados 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 ao 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. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM.

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
- Usar o console do Aurora DSQL
- Permitir que os usuários visualizem suas próprias permissões

- Permitir o gerenciamento de clusters e a conexão com o banco de dados
- Acessar recursos do Aurora DSQL com base em tags

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 saber mais, 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 saber mais 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: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível 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 CloudFormation. Para saber mais, 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 saber mais, 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

saber mais, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para saber mais 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": "*"
        }
    ]
}
```

Permitir o gerenciamento de clusters e a conexão com o banco de dados

A política a seguir concede a um usuário do IAM permissão para gerenciar e se conectar a um cluster específico do Aurora DSQL. A política define o escopo das ações de gerenciamento e conexão de clusters para um único cluster de nome do recurso da Amazon (ARN), ao mesmo tempo em que permite `dsql:ListClusters` em todos os recursos, já que esta ação não é compatível com permissões no nível do recurso.

Este exemplo usa `dsql:DbConnectAdmin` para se conectar com o perfil de `admin`. Para se conectar a um perfil de banco de dados personalizado, substitua `dsql:DbConnectAdmin` por `dsql:DbConnect`. Para obter mais informações, consulte Autenticação e autorização para o Aurora DSQL.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowClusterManagement",
            "Effect": "Allow",
            "Action": [
                "dsql:GetCluster",
                "dsql:UpdateCluster",
                "dsql:DeleteCluster",
                "dsql:DbConnectAdmin",
                "dsql:TagResource",
                "dsql:ListTagsForResource",
                "dsql:UntagResource"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/my-cluster-id"
        },
        {
            "Sid": "AllowListClusters",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        }
    ]
}
```

Acessar recursos do Aurora DSQL com base em tags

É possível utilizar condições na política baseada em identidade para controlar o acesso aos recursos do Aurora DSQL com base em tags. O exemplo a seguir mostra como criar uma política que permite visualizar um cluster. No entanto, a política concede permissão somente se a tag de cluster `Owner` tiver o valor do nome desse usuário. Essa política também concede as permissões necessárias concluir essa ação no console.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListClustersInConsole",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
```

```
        },
        {
            "Sid": "ViewClusterIfOwner",
            "Effect": "Allow",
            "Action": "dsql:GetCluster",
            "Resource": "arn:aws:dsql:*:*:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

É possível anexar essa política aos usuários do IAM na sua conta. Se um usuário chamado `richard-roe` tentar visualizar um cluster do Aurora DSQL, o cluster deverá estar marcado como `Owner=richard-roe` ou `owner=richard-roe`. Caso contrário, o IAM nega o acesso. A chave da tag de condição `Owner` corresponde a `Owner` e a `owner` porque os nomes das chaves de condição não fazem distinção entre maiúsculas e minúsculas. Para obter mais informações, consulte Elementos de política JSON do IAM: condição no Guia do usuário do IAM.

A política a seguir permite que um usuário crie clusters somente se marcar o cluster com seu próprio nome de usuário como o `Owner`. Ela também permite a marcação somente em clusters que o usuário já possui.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedCluster",
            "Effect": "Allow",
            "Action": "dsql:CreateCluster",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
```

```
            }
        },
        {
            "Sid": "AllowTagOwnedClusters",
            "Effect": "Allow",
            "Action": "dsql:TagResource",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

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

É possível 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), é possível 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.

Políticas baseadas em recursos para o Aurora DSQL

Use políticas baseadas em recursos para o Aurora DSQL com o objetivo de restringir ou conceder acesso aos seus clusters por meio de documentos de política JSON que são anexados diretamente aos recursos do cluster. Essas políticas oferecem controle refinado sobre quem pode acessar seu cluster e em quais condições.

Os clusters do Aurora DSQL podem ser acessados pela Internet pública por padrão, com a autenticação do IAM como controle de segurança principal. As políticas baseadas em recursos permitem que você adicione restrições de acesso, principalmente para bloquear o acesso da Internet pública.

As políticas baseadas em recursos são políticas baseadas em identidade do IAM. A AWS avalia os dois tipos de política para determinar as permissões finais para qualquer solicitação de acesso ao seu cluster. Por padrão, os clusters do Aurora DSQL são acessíveis em uma conta. Se um usuário ou perfil do IAM tiver permissões do Aurora DSQL, ele poderá acessar clusters sem anexar uma política baseada em recursos.

> **Note**
>
> As alterações nas políticas baseadas em recursos acabam sendo consistentes e normalmente entram em vigor em um minuto.

Para acessar mais informações sobre as diferenças entre as políticas baseadas em identidade e as baseadas em recursos, consulte Políticas baseadas em identidade e em recurso no Guia do usuário do IAM.

Quando usar políticas baseadas em recursos

As políticas baseadas em recursos são políticas úteis nestes cenários:

- Controle de acesso baseado em rede: restrinja o acesso com base na VPC ou no endereço IP de origem das solicitações ou bloqueie totalmente o acesso à Internet pública. Use chaves de condição, como `aws:SourceVpc` e `aws:SourceIp`, para controlar o acesso à rede.
- Várias equipes ou aplicações: conceda acesso ao mesmo cluster para várias equipes ou aplicações. Em vez de gerenciar políticas individuais do IAM para cada entidade principal, você define as regras de acesso uma vez no cluster.
- Acesso condicional complexo: controle o acesso com base em vários fatores, como atributos de rede, contexto da solicitação e atributos do usuário. É possível combinar várias condições em uma única politica.
- Governança de segurança centralizada: permita que os proprietários do cluster controlem o acesso usando uma sintaxe de política da AWS conhecida, que se integra às suas práticas de segurança existentes.

> **Note**
>
> O acesso entre contas ainda não é aceito pelas políticas baseadas em recursos do Aurora DSQL, mas estará disponível em versões futuras.

Quando tentam se conectar ao seu cluster do Aurora DSQL, a AWS avalia sua política baseada em recursos como parte do contexto de autorização, junto com todas as políticas relevantes do IAM, para decidir se vai permitir ou rejeitar a solicitação.

Políticas baseadas em recursos podem conceder acesso às entidades principais na mesma conta da AWS que do cluster. Em relação a clusters multirregionais, cada cluster regional tem sua própria política baseada em recursos, permitindo controles de acesso específicos da região quando necessário.

> **Note**
>
> As chaves de contexto de condição podem variar entre as regiões (como IDs de VPC).

Tópicos

- Criar clusters com políticas baseadas em recursos
- Adicionar e editar políticas baseadas em recursos para clusters
- Visualizar políticas baseadas em recursos
- Remover políticas baseadas em recursos
- Exemplos de políticas baseadas em recursos comuns
- Bloquear o acesso público com políticas baseadas em recursos no Aurora DSQL
- Operações da API do Aurora DSQL e políticas baseadas em recursos

Criar clusters com políticas baseadas em recursos

Ao criar um cluster, você pode anexar políticas baseadas em recursos a fim de garantir que controles de acesso sejam implementados desde o início. Cada cluster pode ter uma única política em linha anexada diretamente ao cluster.

AWSManagement Console da

Como adicionar uma política baseada em recursos durante a criação de um cluster

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. Selecione Criar cluster.
3. Defina o nome do cluster, as marcações e as configurações multirregionais, conforme necessário.
4. Na seção Configurações do cluster, localize a opção Política baseada em recursos.
5. Ative Adicionar política baseada em recursos.
6. Insira seu documento de política no editor JSON. Por exemplo, para bloquear o acesso público à Internet:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Você pode usar Editar declaração ou Adicionar nova instrução para criar sua política.
8. Conclua a configuração restante do cluster e escolha Criar cluster.

AWS CLI

Use o parâmetro `--policy` ao criar um cluster para anexar uma política em linha:

```
aws dsql create-cluster --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}'
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}

response = client.create_cluster(
    policy=json.dumps(policy)
)

print(f"Cluster created: {response['identifier']}")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
```

```
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
    }
  }]
}
""";

CreateClusterRequest request = CreateClusterRequest.builder()
    .policy(policy)
    .build();

CreateClusterResponse response = client.createCluster(request);
System.out.println("Cluster created: " + response.identifier());
```

Adicionar e editar políticas baseadas em recursos para clusters

AWSManagement Console da

Como adicionar uma política baseada em recursos a um cluster existente

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. Escolha seu cluster na lista para abrir a página de detalhes do cluster.
3. Escolha a aba Permissões.
4. Na seção Política baseada em recursos, escolha Adicionar política.
5. Insira seu documento de política no editor JSON. Você pode usar Editar declaração ou Adicionar nova instrução para criar sua política.
6. Escolha Add policy.

Como editar uma política baseada em recursos existente

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. Escolha seu cluster na lista para abrir a página de detalhes do cluster.
3. Escolha a aba Permissões.
4. Na seção Política baseada em recursos, escolha Editar.

5. Modifique o documento de política no editor JSON. Você pode usar Editar instrução ou Adicionar nova instrução para atualizar sua política.
6. Escolha Salvar alterações.

AWS CLI

Use o comando `put-cluster-policy` para anexar uma nova política ou atualizar uma política existente em um cluster:

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": {"aws:SourceVpc": "true"}
```

```
        }
    }]
}

response = client.put_cluster_policy(
    identifier='your_cluster_id',
    policy=json.dumps(policy)
)
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.PutClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "Null": {"aws:SourceVpc": "true"}
    }
  }]
}
""";

PutClusterPolicyRequest request = PutClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .policy(policy)
    .build();

client.putClusterPolicy(request);
```

Visualizar políticas baseadas em recursos

Você pode visualizar as políticas baseadas em recursos anexadas aos seus clusters para entender os controles de acesso atuais implementados.

AWSManagement Console da

Como visualizar políticas baseadas em recursos

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. Escolha seu cluster na lista para abrir a página de detalhes do cluster.
3. Escolha a aba Permissões.
4. Veja a política anexada na seção Política baseada em recursos.

AWS CLI

Use o comando `get-cluster-policy` para visualizar uma política baseada em recursos:

```
aws dsql get-cluster-policy --identifier your_cluster_id
```

AWS SDKs

Python

```
import boto3
import json

client = boto3.client('dsql')

response = client.get_cluster_policy(
    identifier='your_cluster_id'
)

# Parse and pretty-print the policy
policy = json.loads(response['policy'])
print(json.dumps(policy, indent=2))
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyRequest;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyResponse;

DsqlClient client = DsqlClient.create();

GetClusterPolicyRequest request = GetClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

GetClusterPolicyResponse response = client.getClusterPolicy(request);
System.out.println("Policy: " + response.policy());
```

Remover políticas baseadas em recursos

É possível remover as políticas baseadas em recursos dos clusters para alterar os controles de acesso.

> **Important**
>
> Quando você remove todas as políticas baseadas em recursos de um cluster, o acesso será controlado inteiramente por políticas baseadas em identidade do IAM.

AWSManagement Console da

Como remover uma política baseada em recursos

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. Escolha seu cluster na lista para abrir a página de detalhes do cluster.
3. Escolha a aba Permissões.
4. Na seção Política baseada em recursos, escolha Excluir.
5. Na caixa de diálogo de confirmação, digite **`confirm`** para confirmar a exclusão.

6. Escolha Excluir.

AWS CLI

Use o comando `delete-cluster-policy` para remover uma política de um cluster.

```
aws dsql delete-cluster-policy --identifier your_cluster_id
```

AWS SDKs

Python

```
import boto3

client = boto3.client('dsql')

response = client.delete_cluster_policy(
    identifier='your_cluster_id'
)

print("Policy deleted successfully")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

DeleteClusterPolicyRequest request = DeleteClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

client.deleteClusterPolicy(request);
System.out.println("Policy deleted successfully");
```

Exemplos de políticas baseadas em recursos comuns

Estes exemplos mostram padrões comuns para controlar o acesso aos seus clusters do Aurora DSQL. Você pode combinar e modificar esses padrões para atender aos seus requisitos específicos de acesso.

Bloqueio de acesso à internet pública

Essa política bloqueia conexões com seus clusters do Aurora DSQL pela Internet pública (não VPC). A política não especifica de qual VPC os clientes podem se conectar, apenas que eles devem se conectar por meio de uma VPC. Para limitar o acesso a uma VPC específica, use `aws:SourceVpc` com o operador de condição `StringEquals`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> Este exemplo usa somente `aws:SourceVpc` para conferir as conexões de VPC. As chaves de condição `aws:VpcSourceIp` e `aws:SourceVpce` fornecem granularidade adicional, mas não são necessárias para o controle de acesso básico somente de VPC.

Para abrir uma exceção para funções específicas, em vez disso, use esta política:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromOutsideVPC",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        },
        "StringNotEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::123456789012:role/ExceptionRole",
            "arn:aws:iam::123456789012:role/AnotherExceptionRole"
          ]
        }
      }
    }
  ]
}
```

Restringir o acesso à organização da AWS

Essa política restringe o acesso às entidades principais em uma organização da AWS:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "o-exampleorgid"
        }
      }
    }
  ]
}
```

Restringir acesso a uma unidade organizacional específica

Essa política restringe o acesso às entidades principais em uma unidade organizacional (UO) específica em uma organização da AWS, oferecendo um controle mais detalhado do que o acesso em toda a organização:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotLike": {
          "aws:PrincipalOrgPaths": "o-exampleorgid/r-examplerootid/ou-exampleouid/*"
        }
      }
    }
  ]
}
```

Políticas para clusters multirregionais

Em relação a clusters multirregionais, cada cluster regional mantém a própria política de recursos, permitindo controles de acesso específicos da região. Veja um exemplo com diferentes políticas por região:

política us-east:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

política us-east:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
```

```
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> As chaves de contexto de condição podem variar entre Regiões da AWS (como IDs de VPC).

Bloquear o acesso público com políticas baseadas em recursos no Aurora DSQL

O Bloqueio de Acesso Público (BPA) é um recurso que identifica e impede a anexação de políticas baseadas em recursos que concedem acesso público aos clusters do Aurora DSQL entre suas contas da AWS. Com o BPA, é possível impedir o acesso público aos recursos do Aurora DSQL. O BPA realiza verificações durante a criação ou a modificação de uma política baseada em recursos e ajuda a melhorar o procedimento de segurança com o Aurora DSQL.

O BPA usa raciocínio automatizado para analisar o acesso concedido por sua política baseada em recursos e alerta você se essas permissões forem encontradas no momento da administração de uma política baseada em recursos. A análise verifica o acesso a todas as declarações de políticas baseadas em recursos, ações e ao conjunto de chaves de condição usadas nas políticas.

> **Important**
>
> O BPA ajuda a proteger os recursos impedindo que o acesso público seja concedido por meio de políticas baseadas em recursos que estão diretamente anexadas aos recursos do Aurora DSQL, como clusters. Além de usar o BPA, inspecione com cuidado as seguintes políticas para garantir que elas não concedam acesso público:
>
> - Políticas baseadas em identidade vinculadas a entidades principais da AWS associadas (por exemplo, perfis do IAM).

- Políticas baseadas em recursos anexadas a recursos da AWS associados, por exemplo, chaves do AWS Key Management Service (KMS).

Você deve garantir que a entidade principal não inclua uma entrada * ou que uma das chaves de condição especificadas restrinja o acesso das entidades principais ao recurso. Se a política baseada em recursos conceder acesso público ao seu cluster entre contas da AWS, o Aurora DSQL impedirá você de criar ou modificar a política até que a especificação dentro da política seja corrigida e considerada não pública.

É possível tornar uma política não pública especificando uma ou mais entidades principais no bloco `Principal`. O exemplo de política baseada em recursos a seguir bloqueia o acesso público ao especificar duas entidades principais.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Políticas que restringem o acesso especificando determinadas chaves de condição também não são consideradas públicas. Junto com a avaliação da entidade principal especificada na política baseada em recursos, as seguintes chaves de condição confiáveis são usadas para concluir a avaliação de uma política baseada em recursos para acesso não público:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`

- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

Além disso, para que uma política baseada em recursos não seja pública, os valores de nome do recurso da Amazon (ARN) e das chaves de string não devem conter curingas nem variáveis. Se a política baseada em recursos usa a chave `aws:PrincipalIsAWSService`, você deve garantir que tenha definido o valor da chave como verdadeiro.

A seguinte política limita o acesso ao usuário `Ben` na conta especificada. A condição faz com que a `Principal` seja restrita e não seja considerada pública.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

O exemplo a seguir de uma política baseada em recursos não pública restringe `sourceVPC` usando o operador `StringEquals`.

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operações da API do Aurora DSQL e políticas baseadas em recursos

As políticas baseadas em recursos no Aurora DSQL controlam o acesso a operações específicas da API. As seções a seguir listam todas as operações da API do Aurora DSQL organizadas por categoria, com uma indicação de quais comporta políticas baseadas em recursos.

A coluna Comporta RBP indica se a operação da API está sujeita à avaliação da política baseada em recursos quando é anexada ao cluster.

APIs de marcações

Operação de API	Descrição	Comporta RBP
ListTagsForResource	Lista as marcações de um recurso do Aurora DSQL	Sim
TagResource	Adiciona marcações a um recurso do Aurora DSQL.	Sim
UntagResource	Remove as marcações de um recurso do Aurora DSQL.	Sim

APIs de gerenciamento de clusters

Operação de API	Descrição	Comporta RBP
CreateCluster	Cria um novo cluster	Não
DeleteCluster	Exclui um cluster.	Sim
GetCluster	Recupera as informações sobre um cluster.	Sim
GetVpcEndpointServiceName	Recupera o nome do serviço de endpoint da VPC para um cluster.	Sim
ListClusters	Lista os clusters na conta.	Não
UpdateCluster	Atualiza a configuração de um cluster.	Sim

APIs de propriedades multirregionais

Operação de API	Descrição	Comporta RBP
AddPeerCluster	Adiciona um cluster par a uma configuração multirregional.	Sim
PutMultiRegionProperties	Define propriedades multirregionais para um cluster.	Sim
PutWitnessRegion	Define a região testemunha para um cluster multirregional.	Sim

APIs de políticas baseadas em recursos

Operação de API	Descrição	Comporta RBP
DeleteClusterPolicy	Exclui a política baseada em recursos de um cluster.	Sim

Operação de API	Descrição	Comporta RBP
GetClusterPolicy	Recupera a política baseada em recursos para um cluster.	Sim
PutClusterPolicy	Cria ou atualiza a política baseada em recursos para um cluster	Sim

AWS Fault Injection ServiceAPIs do

Operação de API	Descrição	Comporta RBP
InjectError	Injeta erros para testes de injeção de falhas.	Não

APIs de backup e restauração

Operação de API	Descrição	Comporta RBP
GetBackupJob	Recupera informações sobre um trabalho de backup.	Não
GetRestoreJob	Recupera informações sobre um trabalho de restauração.	Não
StartBackupJob	Inicia um trabalho de backup para um cluster.	Sim
StartRestoreJob	Inicia um trabalho de restauração por meio de um backup.	Não
StopBackupJob	Interrompe um trabalho de backup em execução.	Não
StopRestoreJob	Interrompe um trabalho de restauração em execução.	Não

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 serviços compatíveis com perfis vinculados ao serviço, consulte Serviços da AWS que funcionam com o IAM e procure os serviços que contêm Sim na coluna Perfil vinculado 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**
>
> É necessário configurar as permissões para permitir que uma entidade do IAM (como um usuário, grupo ou perfil) crie, edite ou exclua uma função vinculada ao 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:
>
> JSON
>
> ```
> {
> ```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "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 atributo 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
```

```
        }
    ]
}
```

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "dsql:CreateCluster",
          "dsql:PutMultiRegionProperties",
          "dsql:PutWitnessRegion",
          "dsql:AddPeerCluster"
        ],
        "Resource": [
           "arn:aws:dsql:us-east-1:123456789012:cluster/*",
           "arn:aws:dsql:us-east-2:123456789012:cluster/*"
        ]
      }
    ]
}
```

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": [
                        "us-west-2"
                    ]
                }
            }
        }
    ]
}
```

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:

- AWS Guia de resposta a incidentes de segurança da
- AWS Práticas recomendadas de segurança, identidade e conformidade da
- 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 escolha o programa de conformidade no qual tem interesse. Para obter informações gerais, consulte Programas de Conformidade da AWS.

É possível baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para saber mais, 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. Para ter mais informações sobre sua responsabilidade pela conformidade ao usar Serviços da AWS, consulte a Documentação de segurança da AWS.

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 saber mais 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. Se a região testemunha ficar comprometida, não haverá impacto na disponibilidade do cluster. As transações de gravação podem sofrer um pequeno aumento na latência até que a região testemunha se recupere.

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.

Teste de injeção de falhas

O Amazon Aurora DSQL se integra ao AWS Fault Injection Service (AWS FIS), um serviço totalmente gerenciado para executar experimentos de injeção de falhas controladas para melhorar a resiliência de uma aplicação. Usando o AWS FIS, você pode:

- Criar modelos de experimentos que definam cenários de falha específicos.
- Injetar falhas (taxas elevadas de erro de conexão de cluster) para validar os mecanismos de tratamento e recuperação de erros da aplicação.
- Testar o comportamento da aplicação em várias regiões para validar a mudança de tráfego entre Regiões da AWS quando há altas taxas de erro de conexão em uma Região da AWS.

Por exemplo, em um cluster multirregional que abrange o Leste dos EUA (Norte da Virgínia) e Leste dos EUA (Ohio), você pode realizar um experimento no Leste dos EUA (Ohio) para testar falhas lá, enquanto o Leste dos EUA (Norte da Virgínia) continua com as operações normais. Esse teste controlado ajuda você a identificar e resolver possíveis problemas antes que eles afetem as workloads de produção.

Consulte Action targets no Guia do usuário do AWS FIS para ver uma lista completa das ações permitidas pelo AWS FIS.

Para ter informações sobre as ações do Amazon Aurora DSQL disponíveis no AWS FIS, consulte Aurora DSQL actions reference no Guia do usuário do AWS FIS.

Para começar a realizar experimentos de injeção de falhas, consulte Planning your AWS FIS experiments no Guia do usuário do AWS FIS.

Segurança da infraestrutura no Amazon Aurora DSQL

Como um serviço gerenciado, o Amazon Aurora DSQL é protegido pelos procedimentos de segurança de rede global da AWS 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 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 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 obter mais informações, consulte Acessar um serviço da AWS usando um endpoint da VPC de interface e Cotas do AWS PrivateLink 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.dsql.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-xxxxxxxxxxxxxxxx \
    --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.

JSON

```
{
  "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);
```

Configuração adicional ao se conectar via Direct Connect ou emparelhamento da Amazon VPC

Pode ser necessária alguma configuração adicional para conectar-se aos clusters do Aurora DSQL usando um endpoint de conexão do AWS PrivateLink de dispositivos on-premises via emparelhamento da Amazon VPC ou Direct Connect. Essa configuração só não será necessária se a sua aplicação estiver em execução na mesma Amazon VPC que o endpoint do AWS PrivateLink. As entradas de DNS privadas criadas acima não serão resolvidas corretamente fora da Amazon VPC do endpoint, mas você pode criar seus próprios registros de DNS privados que serão resolvidos no endpoint de conexão do AWS PrivateLink.

Crie um registro DNS CNAME privado que aponte para o nome de domínio totalmente qualificado do endpoint do AWS PrivateLink. O nome de domínio do registro de DNS criado deve ser construído por meio dos seguintes componentes:

1. O identificador do serviço do nome do serviço. Por exemplo, .: `dsql-fnh4`
2. O Região da AWS

Crie o registro DNS CNAME com um nome de domínio no seguinte formato: `*.`*`service-identifier`*`.`*`region`*`.on.aws`

O formato do nome de domínio é importante por dois motivos:

1. O nome do host usado para se conectar ao Aurora DSQL deve corresponder ao certificado do servidor do Aurora DSQL ao usar o modo SSL `verify-full`. Isso garante o maior nível de segurança da conexão.
2. O Aurora DSQL usa a parte do ID do cluster do nome do host usado para se conectar ao Aurora DSQL para identificar o cluster de conexão.

Se não for possível criar registros DNS privados, você ainda poderá se conectar ao Aurora DSQL. Consulte Conectar-se a um cluster do Aurora DSQL usando um endpoint do AWS PrivateLink sem DNS privado.

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. O Região da AWS. Por exemplo, .: `us-east-1`

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
```

Conectar-se a um cluster do Aurora DSQL usando um endpoint do AWS PrivateLink sem DNS privado

As instruções de conexão acima dependem de registros DNS privados. Se sua aplicação estiver sendo executada na mesma Amazon VPC que seu endpoint do AWS PrivateLink, os registros DNS

serão criados para você. Como alternativa, se você estiver se conectando a partir de dispositivos on-premises via emparelhamento da Amazon VPC ou Direct Connect, poderá criar seus próprios registros DNS privados. No entanto, a configuração do registro DNS nem sempre é possível devido às restrições de rede impostas por suas equipes de segurança. Se sua aplicação precisar se conectar usando o Direct Connect ou a partir de uma Amazon VPC emparelhada e a configuração do registro DNS não for possível, você ainda poderá se conectar ao Aurora DSQL.

O Aurora DSQL usa a parte do ID do cluster do seu nome de host para identificar o cluster de conexão, mas se a configuração do registro DNS não for possível, o Aurora DSQL aceita a especificação do cluster de destino usando a opção de conexão `amzn-cluster-id`. Com essa opção, é possível usar o nome de domínio totalmente qualificado do seu endpoint do AWS PrivateLink como seu nome de host ao se conectar.

> **Important**
>
> Ao se conectar ao nome de domínio ou endereço IP totalmente qualificado do seu endpoint do AWS PrivateLink, o modo SSL `verify-full` não é aceito. Por esse motivo, é preferível configurar o DNS privado.

Exemplo: especificar a opção de conexão do ID do cluster usando o PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export HOSTNAME=vpce-04037adb76c111221-d849uc2p.dsql-fnh4.us-east-1.vpce.amazonaws.com
 # This should match your endpoint's fully-qualified domain name

# Construct the hostname used to generate the authentication token
export AUTH_HOSTNAME="$CLUSTERID.dsql.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $AUTH_HOSTNAME)

# Specify the amzn-cluster-id connection option
export PGOPTIONS="-c amzn-cluster-id=$CLUSTERID"

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

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 saber mais, consulte os seguintes recursos:

- Guia do usuário do Amazon Aurora DSQL
- AWS PrivateLink documentação da
- Access AWS services through AWS PrivateLink

Análise de configuração e vulnerabilidade no Amazon Aurora DSQL

AWSA 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 recursos a seguir:

- 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:dsql:*: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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "backup.amazonaws.com"
    },
    "Action": "dsql:GetCluster",
    "Resource": [
      "arn:aws:dsql:*:123456789012:cluster/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:backup:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

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
- Práticas recomendadas de segurança preventiva para o Aurora DSQL

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 ultrapassar um limite especificado, 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 Console de gerenciamento da AWS, 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.

- Os cálculos de limite de armazenamento podem levar algum tempo para exibir o armazenamento liberado após a execução do comando `DROP TABLE`. Se você precisar de capacidade de armazenamento adicional, consulte Cotas de cluster para solicitar atualizações de cota.
- Para tabelas grandes no Aurora DSQL, use o catálogo do sistema para recuperar contagens de linhas da tabela em vez de operações `COUNT(*)`. Para ter mais informações, consulte Usar tabelas de sistema e comandos no Aurora DSQL.
- O Aurora DSQL gerencia permissões por meio de concessões em nível de esquema. Os usuários administradores criam esquemas usando `CREATE SCHEMA` e concedem acesso usando `GRANT USAGE ON SCHEMA`. Os usuários administradores gerenciam objetos no esquema público, enquanto os usuários não administradores criam objetos em esquemas criados pelo usuário. O perfil de administrador pode se autoconceder qualquer outro perfil para obter permissões em objetos criados pelo usuário. Para obter mais informações, consulte Autorizar perfis de banco de dados a usar SQL no banco de dados.
- Quando os drivers chamam `PG_PREPARED_STATEMENTS`, o Aurora DSQL oferece uma visualização ampla do cluster com relação às instruções preparadas que estão armazenadas em cache. Talvez você veja mais instruções preparadas por conexão do que o esperado para o mesmo cluster e perfil do IAM. O Aurora DSQL gerencia o nome das instruções dinamicamente durante a preparação.
- Ao se conectar por meio de instâncias somente IPv4, verifique se o cliente está configurado para conexões IPv4. Alguns clientes PostgreSQL tentam conexões IPv4 e IPv6 no modo de pilha dupla. Se a conexão IPv4 sofrer controle de utilização, o cliente poderá tentar IPv6 e exibir o erro `NetworkUnreachable` em hosts somente IPv4. Configure o cliente para usar IPv4 explicitamente e evitar esse comportamento.
- Depois que um usuário administrador cria um esquema, as alterações `GRANT` e `REVOKE` propagam-se para as conexões existentes durante a vida útil da conexão (até uma hora). Para obter um efeito imediato, estabeleça uma nova conexão após alterações de permissão.
- Durante cenários raros de recuperação de clusters vinculados a várias regiões, as operações automatizadas de recuperação de cluster mantêm alta disponibilidade, mas pode haver erros

transitórios de controle de simultaneidade ou de conexão. Na maioria dos casos, somente uma porcentagem da workload é afetada. Quando se deparar com esses erros transitórios, tente novamente a transação ou se reconecte com o cliente.

- Alguns clientes SQL, como o DataGrip, solicitam amplos metadados do sistema para preencher as informações do esquema. O Aurora DSQL fornece metadados essenciais para a funcionalidade de consulta SQL. A exibição do esquema nesses clientes pode mostrar informações limitadas em comparação com o conjunto completo de recursos.
- Para garantir que as consultas reconheçam esquemas e tabelas recém-criados, atualize a conexão depois de criar ou eliminar objetos do banco de dados. Isso inclui cenários em que são exibidos erros `Schema Already Exists` após a remoção de um esquema ou de consultas a objetos criados em outra conexão. Desconecte e reconecte ou execute `SET search_path` novamente para atualizar o cache do catálogo.
- Em consultas complexas, use `EXPLAIN ANALYZE VERBOSE` para identificar operações de alta latência e otimizar os planos de consulta. Os índices de cobertura podem reduzir significativamente os custos de DPU ao permitir verificações somente de índice em vez de verificações de tabelas completas. Para obter mais informações, consulte Trabalhar com os planos EXPLAIN do Aurora DSQL.
- Os limites de conexão são gerenciados em nível de cluster. Consulte Cotas de cluster para solicitar atualizações de cota.

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 256 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 reabastecimento 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	A 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	A 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 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	A 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

Descrição	Limite-padrão	Configurável?	Código de erro do Aurora DSQL	A mensagem de erro
Número máximo de sequências	5.000	Não	`54000`	`ERROR: creating more than 500 not allowed`

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 Console de gerenciamento da AWS 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
- Solucionar erros de autenticação
- Solucionar erros de autorização
- Solucionar de erros de SQL
- Solucionar erros de OCC
- Solucionar problemas em conexões SSL/TLS

Solução de problemas de erros de conexão

erro: código de erro SSL não reconhecido: 6 ou não foi possível aceitar a conexão, o sni não foi recebido.

Você pode 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.

Mapeamento incorreto de usuário para ação

Esse erro ocorre quando o tipo de token de autenticação não corresponde ao perfil do banco de dados. O Aurora DSQL usa dois tipos de token: `DbConnectAdmin` para o perfil `admin` e `DbConnect` para perfis personalizados do banco de dados.

- Se você ver `Wrong user to action mapping. user: admin, action: DbConnect`, use `generate-db-connect-admin-auth-token` em vez de `generate-db-connect-auth-token`.
- Se você ver `Wrong user to action mapping. user: myusername, action: DbConnectAdmin`, use `generate-db-connect-auth-token` em vez de `generate-db-connect-admin-auth-token`.

Solucionar erros de autorização

Role <role> not supported

O Aurora DSQL não aceita a operação `GRANT`. Consulte Subconjuntos de comandos SQL compatíveis no Aurora DSQL.

Cannot establish trust with role <role>

O Aurora DSQL não aceita a operação `GRANT`. Consulte Subconjuntos de comandos SQL compatíveis no 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: 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”

Essa transação tentou modificar as mesmas tuplas que de outra transação simultânea. Isso indica contenção nas tuplas modificadas. Para saber mais, consulte Controle de simultaneidade no Aurora DSQL.

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.

Fornecer feedback sobre o Amazon Aurora DSQL

Se você encontrar recursos essenciais para sua migração, mas que atualmente não são aceitos no Aurora DSQL, a AWS vai oferecer alguns canais para compartilhar feedback:

Canais de feedback

Servidor do Aurora DSQL Discord

: Junte-se ao servidor Aurora DSQL Discord para se conectar com a equipe e a comunidade da AWS. Compartilhe solicitações de recursos, discuta os desafios da migração e receba feedback em tempo real.

AWS Support

: Se você tiver um plano do AWS Support, crie um caso de suporte para discutir seus requisitos específicos e necessidades de cronograma.

AWS re:Post

: Use o AWS re:Post para fazer perguntas e compartilhar feedback com a comunidade e os especialistas da AWS.

Solicitações efetivas de recursos

Ao solicitar recursos, forneça:

- Descrição do caso de uso: explique o que você está tentando realizar e por quê.
- Solução alternativa atual: descreva todas as alternativas que você já tentou.
- Impacto nos negócios: explique como o recurso ausente afeta o cronograma de migração ou a funcionalidade da aplicação.
- Nível de prioridade: indique se isso está bloqueando sua migração ou se seria uma melhoria interessante.

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
Novo conteúdo: Conector do Aurora DSQL para .NET Npgsql	Documentação adicionada para o conector do Aurora DSQL para .NET Npgsql, que encapsula o Npgsql com a autenticação automática do IAM. O conector gerencia a geração de tokens, a configuração SSL e o gerenciamento de conexões para aplicações .NET. Para obter mais informações, consulte Conectar-se a clusters do Aurora DSQL com um conector .NET Npgsql.	20 de março de 2026
Conteúdo atualizado: Referência de comandos SQL e consultas de sistema	Adição de START TRANSACTION e ROLLBACK à referência dos comandos de controle de transação, com END e ABORT como aliases. Consultas úteis do sistema adicionadas para recuperar informações sobre a versão do Aurora DSQL e do PostgreSQL. Para obter mais informações, consulte Referência de compatibilidade do PostgreSQL.	13 de março de 2026

Conteúdo atualizado: Carregar dados no Aurora DSQL	Guia de carregamento de dados atualizado com o uso de `\copy` do lado do cliente, as práticas recomendadas de INSERT e orientações para pré-criar tabelas antes do carregamento. Para obter mais informações, consulte Carregar dados no Aurora DSQL.	13 de março de 2026
Novo conteúdo: Conector do Aurora DSQL para Ruby pg	Documentação adicionada para o conector do Aurora DSQL para Ruby pg, que encapsula o pg gem com a autenticação automática do IAM. O conector gerencia a geração de tokens, a configuração SSL e o gerenciamento de conexões para aplicações Ruby. Para obter mais informações, consulte Conectar-se a clusters do Aurora DSQL com um conector Ruby pg.	12 de março de 2026
Conteúdo atualizado: Trabalhos DDL assíncronos	Documentação sobre `sys.jobs` atualizada com detalhes expandidos sobre monitoramento e gerenciamento de operações assíncronas de DDL. Para obter mais informações, consulte Referência de compatibilidade do PostgreSQL.	6 de março de 2026

Conteúdo atualizado: Geração de tokens de autenticação PHP	Guia adicionado sobre o SDK de PHP à página de geração de tokens de autenticação. Para obter mais informaçõ es, consulte Gerar tokens de autenticação.	5 de março de 2026
Novo conteúdo: Carregar dados no Aurora DSQL	Guia adicionado sobre carregar dados em clusters do Aurora DSQL, incluindo o uso do utilitário Aurora DSQL loader. Para obter mais informações, consulte Carregar dados no Aurora DSQL.	5 de março de 2026
Novo conteúdo: “Sequências e colunas de identidade	Suporte adicionado a sequências e colunas de identidade. Novas páginas de referência de comandos SQL para CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE e funções de manipulação de sequência . Atualização de CREATE TABLE e ALTER TABLE para incluir a sintaxe de coluna de identidade. Um novo guia foi adicionado para escolher tipos de identificador e tamanhos de cache. Para ter mais informações, consulte Sequências e colunas de identidade.	11 de fevereiro de 2026

Novo conteúdo: “Conector do Aurora DSQL para Go	Documentação adicionada para o conector do Aurora DSQL para Go, que encapsula o pgx com a autentica ção automática do IAM. O conector gerencia a geração de tokens, a configuração SSL e o gerenciamento de conexões para aplicações Go. Para ter mais informaçõ es, consulte Conectar-se a clusters do Aurora DSQL com um conector Go.	5 de fevereiro de 2026
Conteúdo atualizado: “Ferramentas de conectivi dade com clusters do Amazon Aurora DSQL	Reorganização da documenta ção das ferramentas de conectividade de clusters para esclarecer a distinção entre conectores e adaptador es fornecidos pela AWS e ferramentas de terceiros . Adição de links ausentes aos exemplos de código. Para ter mais informações, consulte Ferramentas de conectividade com clusters do Amazon Aurora DSQL.	26 de janeiro de 2026

Novo conteúdo: “Plug-in do Aurora DSQL para o DBeaver Community Edition”	Documentação adicionada para o plug-in do Aurora DSQL para o DBeaver Community Edition, que permite a autenticação do IAM e a configuração simplificada da conexão para clusters do Aurora DSQL. Inclui instruções de instalação, configuração de conexão e orientação para solução de problemas. Para ter mais informações, consulte Usar o DBeaver para acessar o Aurora DSQL.	26 de janeiro de 2026
Novo conteúdo: “Driver do Aurora DSQL para SQLTools”	Documentação adicionada para o driver do Aurora DSQL para SQLTools, uma extensão do Visual Studio Code que permite que os desenvolvedores se conectem e consultem bancos de dados do Aurora DSQL diretamente do VS Code com autenticação automática do IAM. Para ter mais informações, consulte Usar o driver do Aurora DSQLpara SQLTools.	26 de janeiro de 2026

Conteúdo atualizado: “Direcionamento do Aurora DSQL: skills e powers”	Documentação adicionada para facilitar a instalação usando a CLI de skills e atender a qualquer tipo de agente. A CLI de skills oferece um método de configuração simplificado que funciona com vários assistentes de codificação de IA, como Claude Code, Cursor, Copilot, Gemini e outros. Para ter mais informações, consulte Direcionamento do Aurora DSQL: skills e powers.	23 de janeiro de 2026
Novo conteúdo: “Adaptador do Aurora DSQL para o Tortoise ORM”	Foi adicionado suporte ao Tortoise ORM, um framework de ORM assíncrono do Python. O adaptador do Aurora DSQL para o Tortoise ORM permite que os desenvolvedores usem o Tortoise ORM com clusters do Aurora DSQL. Para ter mais informações, consulte Adaptadores e dialetos do Aurora DSQL.	23 de janeiro de 2026

Novo conteúdo: “Direcionamento do Aurora DSQL: skills e powers	Nova documentação adicionada para configurar o direcionamento da IA com o Aurora DSQL usando skills e powers. Inclui instruções de configuração para Kiro Powers, Claude Skills, Gemini Skills e Codex Skills. Para ter mais informações, consulte Direcionamento do Aurora DSQL: skills e powers.	16 de janeiro de 2026
Suporte a índice de tipo de dados numérico	Suporte adicionado para índice de tipo de dados `numeric` no Aurora DSQL. Agora é possível usar colunas `numeric` como chaves primárias e em índices secundários. Para ter mais informações, consulte Tipos de dados compatíveis no Aurora DSQL.	13 de janeiro de 2026
Conteúdo atualizado: “Subconjuntos de comandos SQL compatíveis	Reorganização da documentação de comandos SQL em páginas separadas para melhorar a navegação e a clareza. Cada comando (CREATE TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW) agora tem uma página dedicada. Para ter mais informações, consulte Subconjuntos de comandos SQL compatíveis.	6 de janeiro de 2026

Conteúdo atualizado: “Servidor MCP para AWS Labs Aurora DSQL”	Atualização da documentação do servidor MCP com abordagens detalhadas de instalação para Claude Code e Codex, bem como exemplos de arquivos de configuração e instalação baseados em CLI. Foi adicionada uma orientação abrangente para encontrar arquivos de configuração de cliente MCP em diferentes ferramentas de desenvolvimento. Para ter mais informações, consulte Servidor MCP para AWS Labs Aurora DSQL.	19 de dezembro de 2025
Conteúdo atualizado: “Migrar do PostgreSQL para o Aurora DSQL	A seção de compatibilidade do PostgreSQL foi reformulada como um guia abrangente de migração. Inclui informações sobre compatibilidade do framework, padrões comuns de migração, diferenças de arquitetura e diretrizes de migração assistida por IA. Foi adicionado um novo capítulo para fornecer feedback sobre o Aurora DSQL. Para ter mais informações, consulte Migração do PostgreSQL para o Aurora DSQL.	16 de dezembro de 2025

Conteúdo atualizado: “Conectar-se ao Aurora DSQL usando o AWS PrivateLink	Documentação adicionada para configuração de DNS privado e a opção de conexão de ID de cluster para atender clientes que usam o AWS PrivateLink com o Direct Connect ou emparelhamento da Amazon VPC. Inclui orientação para conexão sem DNS privado usando a opção de conexão `amzn-cluster-id` . Para ter mais informações, consulte Gerenciar e conectar-se com clusters do Amazon Aurora DSQL usando o AWS PrivateLink.	11 de dezembro de 2025
Conteúdo atualizado: “Ciclo de vida do cluster do Aurora DSQL	Documentação atualizada para gerenciamento do ciclo de vida de clusters do Aurora DSQL. Oferece explicações sobre as definições de status de cluster, as transições de estado e as operações disponíveis durante os estados ocioso e inativo. Para ter mais informações, consulte Ciclo de vida do cluster do Aurora DSQL.	4 de dezembro de 2025

Novo conteúdo: “Conectores do Aurora DSQL para Python e Node.js	Documentação adicionada para conectores do Aurora DSQL para Python (psycopg, psycopg2 e asyncpg) e Node.js (node-postgres e Postgres.js). Esses conectores integram a autenticação do IAM para conectar aplicações aos clusters do Aurora DSQL. Para ter mais informações, consulte Conectores do Aurora DSQL.	21 de novembro de 2025
Novo conteúdo: “Usar o JupyterLab com o Aurora DSQL	Foi adicionado um guia detalhado para conectar e consultar o Aurora DSQL usando o JupyterLab com Python. Inclui instruções para instalações locais do JupyterLab e ambientes de do Amazon SageMaker AI. Para ter mais informações, consulte Usar o JupyterLab com o Aurora DSQL.	20 de novembro de 2025
Conteúdo atualizado: “Cotas para o Aurora DSQL	A cota máxima de armazenamento em cluster foi atualizada de 128 TiB para 256 TiB. Para ter mais informações, consulte Quotas para o Aurora DSQL.	19 de novembro de 2025

Novo conteúdo: “Comece a usar o Editor de Consultas do Aurora DSQL”	Documentação adicionada para usar o Editor de Consultas do Aurora DSQL no Console de Gerenciamento da AWS. Inclui pré-requisitos, configuração de conexão e instruções de execução de consultas. Para ter mais informações, consulte Comece a usar o Editor de Consultas do Aurora DSQL.	18 de novembro de 2025
Suporte a políticas baseadas em recursos para o Amazon Aurora DSQL	Adição de suporte à política baseada em recursos (RBP) com novas permissões: `PutClusterPolicy` , `GetClusterPolicy` e `DeleteClusterPolicy` . Essas permissões possibilitam o gerenciamento de políticas em linha anexadas clusters do Aurora DSQL para controle de acesso refinado. Atualização das políticas gerenciadas AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess e AmazonAuroraDSQLConsoleFullAccess para incluir recursos de RBP. Para acessar mais informações, consulte Políticas gerenciadas pela AWS para o Amazon Aurora DSQL.	15 de outubro de 2025

Conector JDBC do Aurora DSQL	Documentação adicionada para o conector JDBC do Aurora DSQL, um conector PgJDBC que integra a autenticação do IAM para conectar aplicações Java aos clusters do Amazon Aurora DSQL. Para obter mais informações, consulte Conectar-se a clusters do Aurora DSQL com um conector JDBC.	2 de setembro de 2025
Atualizações de políticas gerenciadas pela AWS para integração com o AWS FIS	Atualização das políticas `AmazonAuroraDSQLFullAccess` e `AmazonAuroraDSQLConsoleFullAccess` para permitir a integração do AWS Fault Injection Service com o Aurora DSQL. Isso permite que você injete falhas em clusters do Aurora DSQL de região única e multirregionais para testar a tolerância a falhas de suas aplicações. Para saber mais sobre essas políticas, consulte as atualizações de políticas gerenciadas pela AWS.	19 de agosto de 2025

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 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 Usar perfis vinculados ao serviço no 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 Usar perfis vinculados ao serviço no 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`. 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