Aurora 사용 설명서

Amazon Aurora

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Aurora: Aurora 사용 설명서

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 해당 상표의 소유자의 자산이며, 해당 상표의 소유자가 Amazon의 계열사이거나 Amazon과 제휴 관계에 있거나 Amazon의 후원을 받는 업체일 수 있습니다.

Table of Contents

Amazon Aurora이란 무엇인가요?

Amazon Aurora(Aurora)는 MySQL 및 PostgreSQL과 호환되는 완전 관리형 관계형 데이터베이스 엔진입니다. MySQL 및 PostgreSQL이 이 고급 상용 데이터베이스의 속도와 안정성을 오픈 소스 데이터베이스의 단순성 및 비용 효율성과 어떻게 결합하는지 이미 알고 계실 것입니다. 오늘날 기존 MySQL 및 PostgreSQL 데이터베이스에 사용되는 코드, 도구 및 애플리케이션 모두 Aurora에서도 사용할 수 있습니다. 일부 워크로드의 경우 Aurora은 기존 애플리케이션을 거의 변경하지 않고도 MySQL의 처리량을 최대 5배, PostgreSQL의 처리량을 최대 3배 제공할 수 있습니다.

Aurora에는 고성능 스토리지 하위시스템이 포함됩니다. MySQL 및 PostgreSQL과 호환되는 데이터베이스 엔진은 빠른 분산형 스토리지를 활용하도록 사용자 지정됩니다. 기본 스토리지는 필요에 따라 자동으로 커집니다. Aurora 클러스터 볼륨 크기는 최대 128 tebibytes (TiB)까지 증가할 수 있습니다. Aurora는 또한 데이터베이스 구성 및 관리의 가장 어려운 측면 중 하나인 데이터베이스 클러스터링 및 복제를 자동화하고 표준화합니다.

Aurora는 관리형 데이터베이스 서비스인 Amazon Realtional Database Service(Amazon RDS)의 일부입니다. Amazon RDS는 클라우드에서 관계형 데이터베이스의 설치, 운영 및 크기 조정을 용이하게 해줍니다. Amazon RDS에 익숙하지 않은 경우 Amazon Relational Database Service 사용 설명서를 참조하세요. Amazon Web Services Services에서 사용할 수 있는 다양한 데이터베이스 옵션에 대해 자세히 알아보려면 Choosing the right database for your organization on AWS(AWS 에서 조직에 적합한 데이터베이스 선택)를 참조하세요.

주제

- Amazon Aurora의 고가용성
- Amazon Aurora를 사용한 복제
- Aurora에 대한 DB 인스턴스 결제

Amazon RDS 공동 책임 모델

Amazon RDS는 DB 인스턴스 및 DB 클러스터의 소프트웨어 구성 요소와 인프라 호스팅을 담당합니다. 사용자는 성능을 개선하기 위해 SQL 쿼리를 조정하는 프로세스인 쿼리 튜닝을 담당합니다. 쿼리 성능은 데이터베이스 디자인, 데이터 크기, 데이터 배포, 애플리케이션 워크로드 및 쿼리 패턴에 따라 크게 달라질 수 있습니다. 모니터링 및 튜닝은 RDS 데이터베이스에 대해 사용자가 소유하는 매우 개별화된 프로세스입니다. Amazon RDS 성능 개선 도우미를 비롯한 도구를 사용하여 문제가 있는 쿼리를 식별할 수 있습니다.

Amazon Aurora를 Amazon RDS와 함께 사용하는 방법

다음 사항은 Amazon Aurora가 Amazon RDS에서 사용 가능한 표준 MySQL 및 PostgreSQL 엔진과 어떻게 관련되는지를 보여줍니다.

- Amazon RDS를 통해 새 데이터베이스 서버를 설정할 때 Aurora MySQL 또는 Aurora PostgreSQL을 DB 엔진 옵션으로 선택합니다.
- Aurora는 관리를 위해 익숙한 Amazon Relational Database Service(Amazon RDS) 기능을 활용합니다. Aurora는 Amazon RDS AWS Management Console 인터페이스, AWS CLI 명령 및 API 작업을 사용하여 프로비저닝, 패치 적용, 백업, 복구, 장애 감지 및 복구와 같은 일상적인 데이터베이스 태스크를 처리합니다.
- Aurora 관리 작업에는 일반적으로 개별 데이터베이스 인스턴스 대신 복제를 통해 동기화되는 전체 데이터베이스 서버 클러스터가 포함됩니다. 자동 클러스터링, 복제 및 스토리지 할당을 통해 최대 MySQL 및 PostgreSQL 배포판에 대한 설정, 작동 및 확장 작업이 간편하고 비용 효율적입니다.
- 스냅 샷을 생성 및 복원하거나 단방향 복제를 설정하여 Amazon RDS for MySQL 및 Amazon RDS for PostgreSQL의 데이터를 Aurora로 가져올 수 있습니다. 기존 RDS for MySQL 및 RDS for PostgreSQL 애플리케이션을 Aurora로 전환할 수 있는 푸시 버튼식 마이그레이션 도구를 사용할 수 있습니다.

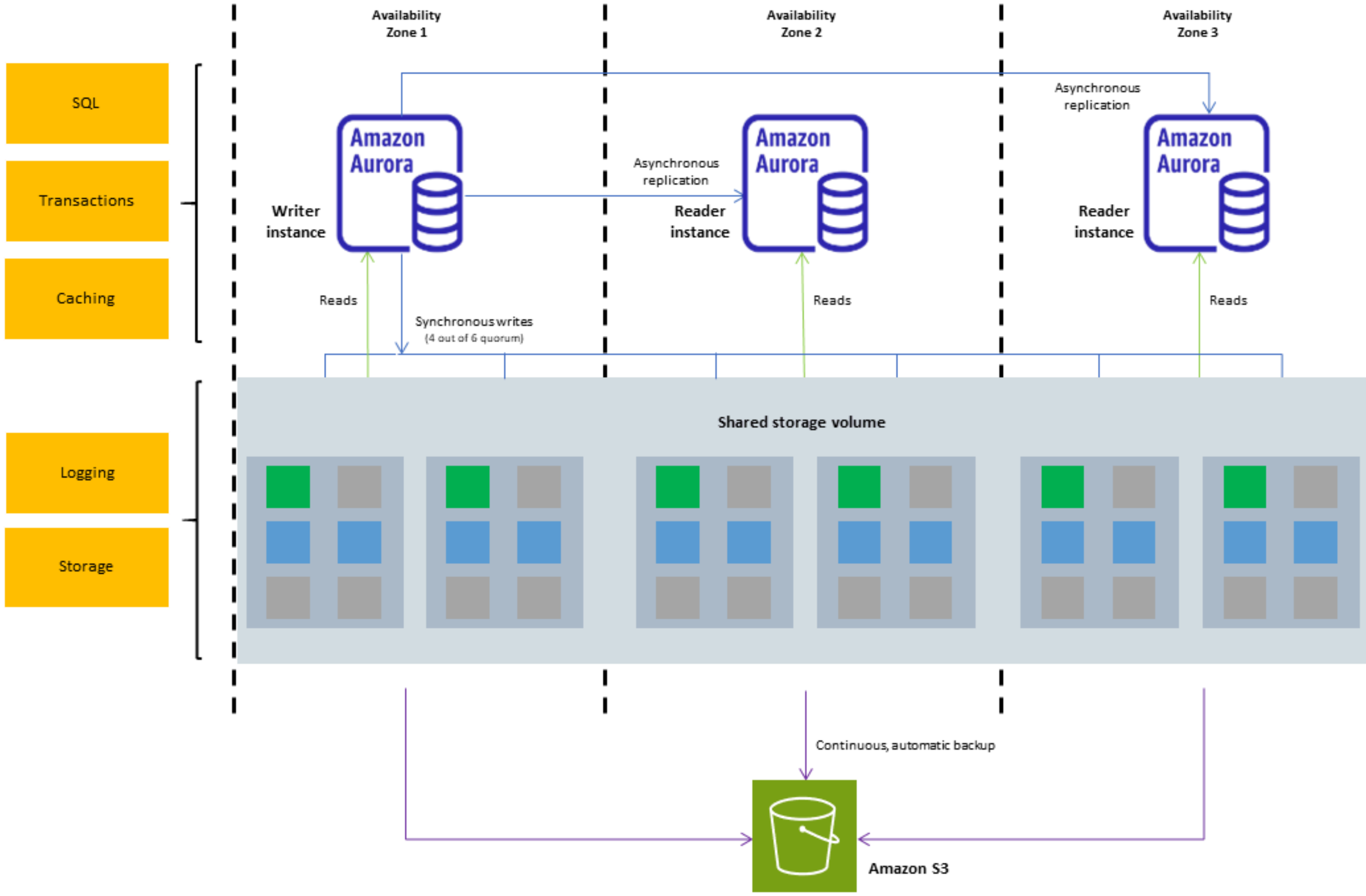
Amazon Aurora를 사용하기 전에 먼저 Amazon Aurora 환경 설정에서 설명하는 단계를 마친 후에 Amazon Aurora DB 클러스터에서 Aurora에 대한 개념과 기능을 검토합니다.

Amazon Aurora DB 클러스터

Amazon Aurora DB cluster는 하나 이상의 DB 인스턴스와 이 DB 인스턴스의 데이터를 관리하는 클러스터 볼륨으로 구성됩니다. Aurora 클러스터 볼륨은 다중 가용 영역을 아우르는 가상 데이터베이스 스토리지 볼륨으로서, 각 가용 영역에는 DB 클러스터 데이터의 사본이 있습니다. Aurora DB 클러스터는 다음과 같이 두 가지 유형의 DB 인스턴스로 구성됩니다.

- 기본(라이터) DB 인스턴스 – 읽기 및 쓰기 작업을 지원하고, 클러스터 볼륨의 모든 데이터 수정을 실행합니다. Aurora DB 클러스터마다 기본 DB 인스턴스가 하나씩 있습니다.
- Aurora 복제본(리더 DB 인스턴스) – 기본 DB 인스턴스와 동일한 스토리지 볼륨에 연결되지만, 읽기 작업만 지원합니다. 각 Aurora DB 클러스터는 기본 DB 인스턴스에 더해 최대 15개까지 Aurora 복제본을 구성할 수 있습니다. Aurora 복제본을 별도의 가용 영역에 배치하여 고가용성을 유지합니다. Aurora는 기본 DB 인스턴스를 사용할 수 없는 경우 자동으로 Aurora 복제본으로 장애 조치합니다. Aurora 복제본에 대해 장애 조치 우선 순위를 지정할 수 있습니다. 또한 Aurora 복제본은 기본 DB 인스턴스에서 읽기 워크로드를 오프로드할 수 있습니다.

다음 다이어그램은 Aurora DB 클러스터에서 클러스터 볼륨, 라이터 DB 인스턴스 및 리더 DB 인스턴스 간의 관계를 보여줍니다.



> **Note**
>
> 위의 정보는 프로비저닝된 병렬 쿼리, Aurora Global Database, Aurora Serverless, Aurora MySQL 호환, Aurora PostgreSQL 호환 등 모든 Aurora DB 클러스터에 적용됩니다.

Aurora DB 클러스터를 보면 컴퓨팅 용량과 스토리지가 분리되어 있습니다. 예를 들어 DB 인스턴스가 1개뿐인 Aurora 구성은 계속해서 클러스터입니다. 기본 스토리지 볼륨에는 여러 가용 영역(AZ)으로 분산된 다수의 스토리지 노드가 포함되어 있기 때문입니다.

Aurora DB 클러스터의 입출력(I/O) 작업은 라이터 DB 인스턴스인지 리더 DB 인스턴스인지에 관계없이 동일한 방식으로 계산됩니다. 자세한 내용은 [Amazon Aurora DB 클러스터의 스토리지 구성](#) 섹션을 참조하세요.

Amazon Aurora

Amazon Aurora를 사용하면 기본 엔진과의 호환성을 유지하면서 애플리케이션 요구 사항에 가장 적합한 지원되는 관계형 데이터베이스 엔진을 선택할 수 있습니다. Aurora는 지원되는 엔진의 데이터베이스 엔진 코드를 재사용하므로 해당 엔진에 대한 기존 스킬, 도구 및 라이브러리를 활용할 수 있습니다. 클러스터를 만들 때 사용할 Amazon Aurora 데이터베이스 엔진 버전을 지정합니다. 선택한 버전에 따라 호환성과 사용 가능한 기능이 결정됩니다.

이 설명서에서는 Amazon Aurora와 해당 데이터베이스 엔진 간의 공통점과 차이점을 설명합니다. 이 정보를 사용하여 선택해야 하는 소프트웨어 버전과 각 버전에서 사용 가능한 기능 및 버그 수정을 확인하는 방법을 알 수 있습니다. 이 참조를 사용하여 적절한 업그레이드 주기를 결정하고 업그레이드를 계획할 수도 있습니다.

Amazon Aurora 데이터베이스 클러스터에 지원되는 데이터베이스 엔진

Amazon Aurora에서 사용할 수 있는 관계형 데이터베이스는 다음과 같습니다. Aurora는 코드를 재사용하고 기본 DB 엔진과의 호환성을 유지합니다. 그러나 Aurora에는 자체 버전 번호, 릴리스 주기, 버전 사용 중단 일정이 있습니다. 각각의 새로운 Aurora 버전에는 각 버전에 적용되는 새로운 기능, 수정 사항, 기타 변경 사항 및 향상된 기능 등을 나열하는 릴리스 정보가 함께 제공됩니다.

Aurora 데이터베이스	사용 설명서	사용 가능한 버전	릴리스 정보
Amazon Aurora MySQL 호환 버전	Amazon Aurora MySQL 작업	Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트	Aurora MySQL 릴리스 정보
Amazon Aurora PostgreSQL 호환 에디션	Amazon Aurora PostgreSQL 작업	Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트	Aurora PostgreSQL 릴리스 정보

데이터베이스 클러스터에 대한 Amazon Aurora 데이터베이스 버전 지정

AWS Management Console에서 또는 AWS CLI에서 데이터베이스 만들기 작업을 사용하거나 `CreateDBCluster` API 작업을 사용하여 새 DB 클러스터를 만들 때는 현재 사용 가능한 모든 Aurora 데이터베이스 버전(메이저 및 마이너) 중에서 지정할 수 있습니다.

Aurora 클러스터를 만드는 방법을 알아보려면 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요. 기존 Aurora 클러스터의 버전을 변경하는 방법을 알아보려면 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

> **Note**
>
> 일부 Aurora 데이터베이스 버전은 사용할 수 없는 AWS 리전도 있습니다. 리전 및 각 AWS 리전에서 사용 가능한 버전에 대한 자세한 내용은 리전 및 가용 영역 및 Aurora Global Database를 지원하는 리전 및 DB 엔진 섹션을 참조하세요.

Amazon Aurora 버전 관리

Amazon Aurora 버전은 호환되는 업스트림 커뮤니티 데이터베이스와 다릅니다. 애플리케이션 호환성을 유지하고 최신 DB 엔진 기능을 활용하는 데 도움이 되도록 다음 섹션에서는 Aurora 버전 관리 규칙과 Aurora 버전이 각 커뮤니티 데이터베이스에 매핑되는 방법을 설명합니다.

Amazon Aurora에서 사용할 수 있는 관계형 데이터베이스의 목록은 Amazon Aurora 데이터베이스 클러스터에 지원되는 데이터베이스 엔진 섹션을 참조하세요.

커뮤니티 데이터베이스와 Aurora 간의 버전 번호의 차이점

각 Amazon Aurora 버전은 해당 커뮤니티 데이터베이스의 특정 버전과 호환됩니다. `version` 함수를 사용하여 데이터베이스의 커뮤니티 버전을 찾고 `aurora_version` 함수를 사용하여 Aurora 버전을 찾을 수 있습니다.

다음 예시에서는 Aurora MySQL 및 Aurora PostgreSQL용 데이터베이스의 커뮤니티 버전을 찾는 방법을 보여줍니다.

Aurora MySQL

`version` 함수는 Aurora MySQL용 데이터베이스의 커뮤니티 버전을 반환합니다.

```
mysql> select version();
```

출력 예제:

```
+------------------+
|   version()      |
+------------------+
```

```
|  8.0.32            |
+------------------+
```

`aurora_version` 함수는 Aurora 버전을 반환합니다.

```
mysql> select aurora_version(), @@aurora_version;
```

출력 예제:

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Aurora PostgreSQL

`version` 함수는 Aurora PostgreSQL용 데이터베이스의 커뮤니티 버전을 반환합니다.

```
postgres=> select version();
```

출력 예제:

```
-----------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)
```

`aurora_version` 함수는 Aurora 버전을 반환합니다.

```
postgres=> select aurora_version();
```

출력 예제:

```
aurora_version
----------------
3.2.2
```

자세한 내용은 SQL을 사용하여 Aurora MySQL 버전 확인 및 Amazon Aurora PostgreSQL 버전 식별(을)를 참조하세요.

기본 Amazon Aurora 버전

기본 버전은 대상 엔진 버전을 수동으로 지정하지 않을 때 Aurora가 데이터베이스 생성 또는 업그레이드를 위해 자동으로 선택하는 버전입니다. 예를 들어 다음 명령은 Aurora PostgreSQL의 기본 엔진 버전(샘플 출력 포함)을 보여줍니다.

```
aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text

16.4
```

모든 메이저 버전에는 해당하는 기본 마이너 버전이 포함되어 있습니다. 따라서 Aurora PostgreSQL 16의 기본 마이너 버전은 16.n이며, Aurora가 새 기본 마이너 버전을 릴리스하면 버전 번호 n이 변경됩니다. 일반적으로 Aurora는 매년 각 메이저 버전에 대해 두 개의 기본 마이너 버전을 릴리스합니다. 다음 bash shell 스크립트는 Aurora PostgreSQL 메이저 버전 세트(샘플 출력 포함)의 기본 마이너 버전을 보여줍니다.

```
for major in 16 15 14 13 12 11; do
  echo -n "Default for Aurora PostgreSQL major version $major: "
  aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --engine-version "$major" \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text
done

Default for Aurora PostgreSQL major version 16: 16.4
Default for Aurora PostgreSQL major version 15: 15.8
Default for Aurora PostgreSQL major version 14: 14.13
Default for Aurora PostgreSQL major version 13: 13.16
Default for Aurora PostgreSQL major version 12: 12.20
Default for Aurora PostgreSQL major version 11: 11.21
```

Aurora DB 클러스터에 대해 마이너 버전 자동 업그레이드를 사용 설정하면 Aurora는 지정된 메이저 버전에 대해 기본 마이너 버전 또는 최신 마이너 버전을 사용합니다. 예를 들어 Aurora PostgreSQL 15의 기본 마이너 버전이 15.8이고 최신 버전 15.10도 사용할 수 있는 경우 Aurora는 15.8 또는 15.10으로 자동 업그레이드할 수 있습니다.

Amazon Aurora 메이저 버전

Aurora 버전에서는 *major.minor.patch* 스키마를 사용합니다. Aurora 메이저 버전은 Aurora와 호환되는 MySQL 또는 PostgreSQL 커뮤니티 메이저 버전을 나타냅니다. Aurora MySQL 및 Aurora PostgreSQL 메이저 버전은 적어도 해당 커뮤니티 버전에 대한 커뮤니티 사용 주기가 끝날 때까지 표준 지원에 따라 사용할 수 있습니다. Aurora 표준 지원이 종료된 이후에도 유료로 메이저 버전을 계속 실행할 수 있습니다. 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 및 Amazon Aurora 요금 단원을 참조하세요.

Aurora MySQL 및 Aurora PostgreSQL의 메이저 버전 및 릴리스 일정에 대한 자세한 내용은 각 릴리스 정보에서 다음 페이지를 참조하세요.

- Aurora MySQL 메이저 버전 릴리스 일정
- RDS for PostgreSQL 메이저 버전 릴리스 일정

describe-db-major-engine-versions AWS CLI 명령을 실행하거나 DescribeDBMajorEngineVersions RDS API 작업을 사용하여 메이저 엔진 버전의 지원 날짜에 대한 정보를 볼 수도 있습니다.

> **Note**
>
> Aurora MySQL 버전 2에 대한 Amazon RDS 추가 지원은 2024년 11월 1일에 시작되지만, 2024년 12월 1일까지는 요금이 청구되지 않습니다. 2024년 11월 1일부터 11월 30일 사이에 해당하는 모든 Aurora MySQL 버전 2 DB 클러스터는 Amazon RDS 추가 지원 범위에 포함됩니다. 자세한 내용은 Amazon RDS 추가 지원(일부 Aurora 버전) 섹션을 참조하세요.

Amazon Aurora 메이저 버전을 사용할 수 있는 기간

Amazon Aurora 메이저 버전은 해당 커뮤니티 버전에 대한 커뮤니티 수명이 끝날 때까지 사용할 수 있습니다. Aurora 표준 지원 종료일을 사용하여 테스트 및 업그레이드 주기를 계획할 수 있습니다. 이 날짜는 최신 버전 업그레이드가 필요할 수 있는 가장 빠른 날짜를 나타냅니다. 날짜에 대한 자세한 내용은 Amazon Aurora 메이저 버전 단원을 참조하세요.

Aurora는 최신 메이저 버전으로 업그레이드하고 사용자의 계획을 지원하기 전, 최소 12개월 전에 미리 알려줍니다. 알림은 업그레이드 프로세스에 대해 다음을 전달합니다.

- 특정 마일스톤의 타이밍
- DB 클러스터에 미치는 영향

- 권장 조치

새 메이저 버전으로 클러스터를 업그레이드하기 전에 항상 새 데이터베이스 버전으로 애플리케이션을 철저히 테스트하는 것이 좋습니다.

메이저 버전이 Aurora의 표준 지원 종료 시점에 도달하면 예정된 유지 관리 기간에 이전 버전을 실행 중인 모든 DB 클러스터가 추가 지원 버전으로 자동 업그레이드됩니다. 추가 지원 요금이 적용될 수 있습니다. Amazon RDS 추가 지원에 대한 자세한 내용은 Amazon RDS 추가 지원 사용을 참조하세요.

Amazon Aurora 마이너 버전

Aurora 버전에서는 *major*.*minor*.*patch* 스키마를 사용합니다. Aurora 마이너 버전은 새 기능과 수정을 위해 증분의 커뮤니티와 Aurora에 특정된 개선을 제공합니다.

Aurora MySQL 및 Aurora PostgreSQL의 마이너 버전 및 릴리스 일정에 대한 자세한 내용은 각 릴리스 정보에서 다음 페이지를 참조하세요.

- Release calendar for Aurora MySQL minor versions
- Release calendar for Aurora PostgreSQL minor versions

다음 섹션에서는 Aurora 마이너 버전에 대해 예상할 수 있는 주기와 수명에 대한 세부 정보를 설명합니다.

주제

- Amazon Aurora 마이너 버전이 출시되는 빈도
- Amazon Aurora 마이너 버전을 사용할 수 있는 기간

Amazon Aurora 마이너 버전이 출시되는 빈도

일반적으로 Amazon Aurora 마이너 버전은 분기별로 출시됩니다. 릴리스 일정은 추가 기능 또는 수정 사항에 따라 다를 수 있습니다.

Amazon Aurora 마이너 버전을 사용할 수 있는 기간

일반적으로 Amazon Aurora는 특정 메이저 버전의 각 마이너 버전을 최소 12개월 동안 사용할 수 있도록 구성합니다. 이 기간이 끝나면 Aurora는 기본 마이너 버전이나 후속 버전으로 데이터베이스를 자동으로 업그레이드할 수 있습니다. Aurora는 이전 마이너 버전을 실행 중인 모든 DB 클러스터에 대해 예약된 유지 관리 기간에 업그레이드를 시작합니다.

경우에 따라 Aurora는 일반적인 12개월보다 빨리 특정 메이저 버전의 마이너 버전을 교체할 수 있습니다. 이유에는 중요한 보안 문제 또는 메이저 버전의 지원 종료일이 포함될 수 있습니다.

수명이 종료된 마이너 버전의 자동 업그레이드를 시작하기 전에 일반적으로 Aurora는 3개월 전에 미리 알림을 제공합니다. Aurora는 업그레이드 프로세스에 대해 다음을 자세히 설명합니다.

- 특정 마일스톤의 타이밍
- DB 클러스터에 미치는 영향
- 권장 조치

알림 기간이 3개월 미만인 알림은 보안 문제와 같이 더 빠른 조치가 필요한 중요한 문제가 있을 때 제공됩니다.

마이너 버전 자동 업그레이드를 사용 설정한 경우 알림이 표시되지만 RDS 이벤트 알림은 표시되지 않습니다. Aurora는 필수 업그레이드 기한이 지난 후 유지 관리 기간 내에 데이터베이스를 업그레이드합니다.

마이너 버전 자동 업그레이드를 사용 설정한 경우 범주가 `maintenance`이고 ID가 `RDS-EVENT-0156`인 Amazon RDS DB 클러스터 이벤트와 알림 메시지가 표시됩니다. Aurora는 다음 유지 관리 기간 내에 데이터베이스를 업그레이드합니다.

마이너 버전이 Aurora 표준 지원 종료에 도달한 후에는 해당 마이너 버전에 대한 추가 패치 버전이 릴리스되지 않습니다. 중요한 버그 수정 또는 CVE를 받으려면 표준 지원을 통해 마이너 버전으로 업그레이드해야 합니다.

마이너 버전 자동 업그레이드에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하시기 바랍니다.

Amazon Aurora 패치 버전

Aurora 버전에서는 *major*.*minor*.*patch* 스키마를 사용합니다. Aurora 패치 버전에는 초기 릴리스 이후 마이너 버전에 추가된 중요한 수정 사항이 포함되어 있습니다(예: Aurora MySQL 3.04.0, 3.04.1, ..., 3.04.3). 새로운 각 마이너 버전은 새로운 Aurora 기능을 제공하지만 특정 마이너 버전 내의 새 패치 버전은 주로 중요한 문제를 해결하는 데 사용됩니다.

패치에 대한 자세한 정보는 Amazon Aurora DB 클러스터 유지 관리 섹션을 참조하세요.

Amazon Aurora DB 클러스터 업그레이드

Amazon Aurora를 사용하면 DB 클러스터에 대한 업그레이드를 제어하고 테스트할 수 있습니다. Amazon Aurora는 마이너 버전 자동 업그레이드, 수동 업그레이드 제어, 필수 업그레이드 및 업그레이드 사전 테스트 옵션을 제공합니다. 중요하지 않은 업그레이드를 연기하고, 심각한 문제에 대한 업그레이드를 강제로 진행하고, 비프로덕션 환경에서 업그레이드 동작을 검증하면서 최신 마이너 버전으로 클러스터를 최신 상태로 유지할 수 있습니다. 다음 섹션에서는 이러한 기능을 사용하여 Aurora DB 클러스터 업그레이드를 관리하고 테스트하는 방법을 자세히 설명합니다.

Aurora의 마이너 버전 자동 업그레이드

마이너 버전 자동 업그레이드는 데이터베이스를 최신 데이터베이스 엔진 버전으로 주기적으로 업데이트합니다. 그러나 업그레이드에 항상 최신 데이터베이스 엔진 버전이 포함되는 것은 아닙니다. 특정 시간에 데이터베이스를 특정 버전으로 유지해야 하는 경우 필요한 일정에 따라 필요한 데이터베이스 버전으로 수동으로 업그레이드하는 것이 좋습니다. 중요한 보안 문제가 발생하거나 버전이 지원 종료에 도달하면 마이너 버전 자동 업그레이드 옵션을 활성화하지 않은 경우에도 Amazon Aurora에서 마이너 버전 업그레이드가 적용될 수 있습니다. 자세한 내용은 특정 데이터베이스 엔진에 대한 업그레이드 설명서를 참조하세요.

Aurora MySQL DB 클러스터의 부 버전 또는 패치 수준 업그레이드 및 마이너 버전 업그레이드 수행 섹션을 참조하세요.

Aurora 클러스터의 모든 DB 인스턴스에 대해 [자동 마이너 버전 업그레이드(Auto minor version upgrade)]를 활성화하여 Aurora 마이너 버전을 최신 상태로 유지합니다. Aurora는 클러스터의 모든 DB 인스턴스에 이 설정이 사용 설정된 경우에만 자동 업그레이드를 수행합니다.

DB 클러스터에 대해 마이너 버전 자동 업그레이드가 예인 경우 Aurora는 자동으로 기본 마이너 버전 또는 최신 마이너 버전으로 업그레이드합니다. 예를 들어 Aurora PostgreSQL 15의 기본 마이너 버전이 15.8이고 버전 15.10이 있는 경우 자동 업그레이드의 대상은 15.8 또는 15.10일 수 있습니다.

Aurora는 일반적으로 자동 마이너 버전 업그레이드가 사용 설정된 DB 클러스터의 경우 1년에 두 번 자동 업그레이드를 예약합니다. 이러한 업그레이드는 클러스터에 대해 지정한 유지 관리 기간 동안 발생합니다. 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하세요.

자동 마이너 버전 업그레이드는 카테고리가 `maintenance`이며 ID가 `RDS-EVENT-0156`인 Amazon RDS DB 클러스터 이벤트를 통해 미리 전달됩니다. 자세한 내용은 Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지 섹션을 참조하세요.

DB 클러스터를 새 버전으로 수동으로 제어하는 업그레이드

마이너 버전 자동 업그레이드 설정을 사용 설정한 경우 Aurora는 DB 클러스터를 기본 마이너 버전 또는 최신 마이너 버전으로 자동 업그레이드합니다. Aurora는 일반적으로 마이너 버전 자동 업그레이드가 사용 설정된 DB 클러스터의 경우 1년에 두 번 자동 업그레이드를 예약합니다. 이러한 업그레이드는 고객이 지정한 유지 관리 기간 중에 시작됩니다.

자동 마이너 버전 업그레이드 기능을 끄려면 Aurora 클러스터 내의 모든 DB 인스턴스에서 마이너 버전 자동 업그레이드를 사용 해제합니다. Aurora는 클러스터의 모든 DB 인스턴스에 설정이 켜져 있는 경우에만 자동 마이너 버전 업그레이드를 수행합니다.

> **Note**
>
> 마이너 버전 수명 종료와 같은 필수 업그레이드의 경우 마이너 버전 자동 업그레이드 설정이 사용 해제되어 있더라도 Aurora는 DB 클러스터를 업그레이드합니다. 알림은 표시되지만 RDS 이벤트 알림은 표시되지 않습니다. Aurora는 필수 업그레이드 기한이 지난 후 유지 관리 기간 내에 클러스터를 업그레이드합니다.

메이저 버전 업그레이드에는 약간의 호환성 위험이 있을 수 있으므로 업그레이드가 자동으로 이루어지지 않습니다. 메이저 버전이 사용 중단되는 경우를 제외하고 이러한 패치를 시작해야 합니다. 메이저 버전으로 클러스터를 업그레이드하기 전에 항상 새 데이터베이스 버전으로 애플리케이션을 철저히 테스트하는 것이 좋습니다.

DB 클러스터를 새로운 Aurora 메이저 버전으로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora MySQL DB 클러스터 업그레이드 및 Amazon Aurora PostgreSQL DB 클러스터 업그레이드 섹션을 참조하세요.

필수 Amazon Aurora 업그레이드

일부 중요 수정 사항의 경우 Aurora는 동일한 마이너 버전 내에서 최신 패치 수준으로 관리형 업그레이드를 수행할 수 있습니다. 이 경우 Aurora는 마이너 버전 자동 업그레이드 기능이 꺼져 있더라도 클러스터를 업그레이드합니다. 이 작업을 수행하기 전에 Aurora에서 자세한 업그레이드 프로세스를 알려줍니다. 세부 정보에는 특정 일정의 타이밍, DB 클러스터에 미치는 영향, 권장 조치 등이 포함됩니다. 이러한 관리형 업그레이드는 클러스터 유지 관리 기간 내에 자동으로 진행됩니다.

업그레이드하기 전에 새로운 Aurora 버전으로 DB 클러스터 테스트

업그레이드 프로세스와 새 버전이 애플리케이션 및 워크로드와 함께 작동하는 방식을 테스트할 수 있습니다. 다음 방법 중 한 가지를 선택하십시오.

- Amazon Aurora의 빠른 데이터베이스 복제 기능을 사용하여 클러스터를 복제합니다. 새 클러스터에서 업그레이드 및 사후 업그레이드 테스트를 수행합니다.
- 클러스터 스냅샷에서 복원하여 새 Aurora 클러스터를 생성합니다. 기존 Aurora 클러스터에서 직접 클러스터 스냅샷을 생성할 수 있습니다. 또한 Aurora는 각 클러스터에 대해 주기적 스냅샷을 자동으로 생성합니다. 그런 다음 새 클러스터에 대한 버전 업그레이드를 시작할 수 있습니다. 원본 클러스터를 업그레이드할지 여부를 결정하기 전에 클러스터의 업그레이드된 복사본을 시험해 볼 수 있습니다.

테스트용 새 클러스터를 만드는 방법에 대한 자세한 내용은 [Aurora DB 클러스터에 대한 볼륨 복제](#) 및 [DB 클러스터 스냅샷 생성](#) 섹션을 참조하세요.

Amazon Aurora 버전 지원

Amazon Aurora DB에 특정 데이터베이스 엔진 동작에 대한 복잡한 종속성이 있는 경우 최신 데이터베이스 엔진 버전으로 업그레이드하기 전에 광범위한 테스트를 수행하는 것이 좋습니다. 일부 Aurora 엔진 버전이 최신 버전으로 대체된 후에도 DB 클러스터를 유지 관리할 수 있는 장기 지원 옵션이 있습니다. 다음 섹션에서는 Aurora DB 클러스터에 대한 장기 지원 옵션을 설명합니다.

선택한 Amazon Aurora 마이너 버전에 대한 장기 지원

각 Aurora 메이저 버전에 대해 특정 마이너 버전은 장기 지원(LTS) 버전으로 지정되며 최소 3년 동안 사용할 수 있습니다. 즉, 메이저 버전당 최소 하나의 마이너 버전을 일반적인 12개월보다 오래 사용할 수 있습니다. 일반적으로 Aurora는 이 기간이 끝나기 6개월 전에 사용자에게 알림 메시지를 전달합니다. Aurora는 업그레이드 프로세스에 대해 다음을 전달합니다.

- 특정 마일스톤의 타이밍
- DB 클러스터에 미치는 영향
- 권장 조치

알림 기간이 6개월 미만인 알림의 경우 보안 문제와 같이 더 빠른 조치가 필요한 중요한 문제가 있을 때 알려줍니다.

LTS 마이너 버전에는 패치 버전을 통한 중요한 수정만 포함됩니다. LTS 버전에는 출시 이후 릴리스된 새로운 기능이 포함되지 않습니다. 일 년에 한 번 LTS 마이너 버전에서 실행되는 DB 클러스터는 LTS 릴리스의 최신 패치 버전으로 패치됩니다. Aurora는 클러스터를 패치하므로 누적 보안 및 안정성 수정의 이점을 누릴 수 있습니다. Aurora는 보안과 같이 적용해야 하는 중요한 수정 사항이 있는 경우 LTS 마이너 버전을 더 자주 패치할 수 있습니다.

> **Note**
>
> 수명 주기 동안 LTS 마이너 버전을 계속 사용하려면 해당 DB 인스턴스에 대해 자동 마이너 버전 업그레이드를 비활성화합니다. LTS 마이너 버전에서 DB 클러스터를 자동으로 업그레이드하지 않으려면 Aurora 클러스터의 모든 DB 인스턴스에서 자동 마이너 버전 업그레이드 활성화 확인란의 선택을 취소합니다.

모든 Aurora LTS 버전의 버전 번호는 Aurora MySQL LTS(장기 지원) 릴리스 및 Aurora PostgreSQL 장기 지원(LTS) 릴리스 사용를 참조하세요.

Amazon RDS 추가 지원(일부 Aurora 버전)

Amazon RDS 추가 지원을 사용하면 Aurora 표준 지원 종료일이 지난 후 메이저 엔진 버전에서 추가 비용을 지불하고 데이터베이스를 계속 실행할 수 있습니다. RDS 추가 지원 기간에 Amazon RDS는 국가 취약성 데이터베이스(NVD) CVSS 심각도 등급에 정의된 대로 중요 및 상위 CVE에 대한 패치를 제공합니다. 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 섹션을 참조하세요.

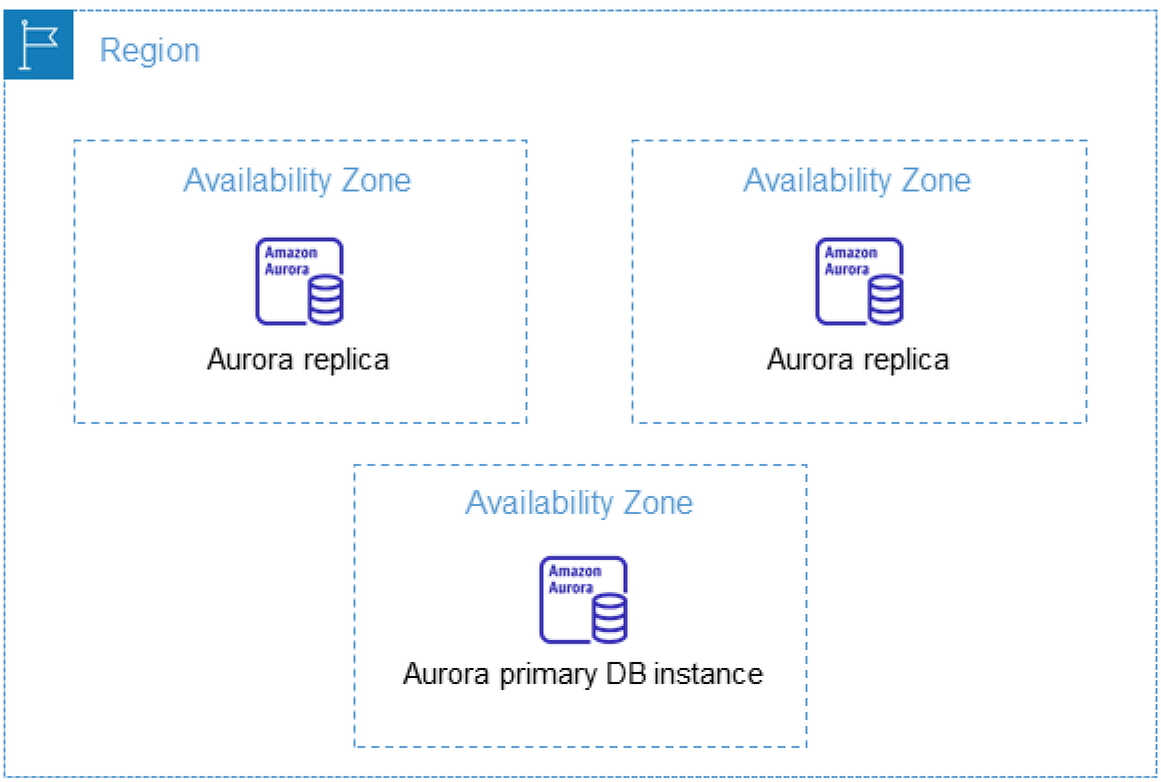
RDS 추가 지원은 특정 Aurora 버전에서만 사용할 수 있습니다. 자세한 내용은 Amazon Aurora 메이저 버전 섹션을 참조하세요.

리전 및 가용 영역

Amazon 클라우드 컴퓨팅 리소스는 세계 각지의 여러 곳에서 호스팅됩니다. 이 위치들은 AWS 리전과 가용 영역으로 구성됩니다. 각 AWS 리전은 개별 지리 영역입니다. 각 AWS 리전은 가용 영역이라고 알려진 격리된 위치를 여러 개 가지고 있습니다.

> **Note**
>
> AWS 리전의 가용 영역을 찾는 방법에 대한 자세한 내용은 Amazon EC2 설명서의 가용 영역 설명을 참조하세요.

Amazon은 최신 기술을 탑재한 고가용성 데이터 센터를 운영하고 있습니다. 드물기는 하지만 동일한 위치에 있는 DB 인스턴스의 가용성에 영향을 미치는 장애가 발생할 수도 있습니다. 그런 장애의 영향을 받는 하나의 위치에서 모든 DB 인스턴스를 호스팅하는 경우에는 모든 DB 인스턴스가 사용이 불가능해질 수 있습니다.



각 AWS 리전이 서로 완전히 독립적이라는 점에 유의하세요. 사용자의 모든 Amazon RDS 활동(예: 데이터베이스 인스턴스 또는 사용할 수 있는 데이터베이스 인스턴스 목록 생성 등)은 현재 기본 AWS 리전에서만 실행됩니다. 기본 AWS 리전은 콘솔에서 변경하거나 `AWS_DEFAULT_REGION` 환경 변수를

설정하여 변경할 수 있습니다. 또는 AWS Command Line Interface(AWS CLI)에서 `--region` 파라미터를 사용하여 재정의할 수 있습니다. 자세한 내용은 AWS Command Line Interface 구성, 특히 환경 변수 및 명령줄 옵션에 대한 섹션을 참조하십시오.

Amazon RDS는 AWS라는 특수 AWS GovCloud (US) 리전을 지원합니다. 이러한 리전은 미국 정부 기관 및 고객이 더욱 민감한 워크로드를 클라우드로 이전하도록 설계되었습니다. AWS GovCloud (US) 리전은 미국 정부의 규제 및 규정 준수 요구 사항을 충족합니다. 자세한 내용은 AWS GovCloud (US)란 무엇입니까?를 참조하십시오.

특정 AWS 리전에서 Amazon RDS DB 인스턴스를 생성하거나 사용하려면 해당 리전 서비스 엔드포인트를 사용하세요.

> **Note**
>
> Aurora는 Local Zones를 지원하지 않습니다.

AWS 리전

각 AWS 리전은 다른 AWS 리전에서 격리되도록 설계되었습니다. 이를 통해 가장 강력한 내결함성 및 안정성을 달성할 수 있습니다.

리소스를 볼 때 지정한 AWS 리전에 연결된 리소스만 표시됩니다. AWS 리전이 서로 격리되어 있고 여러 AWS 리전에 리소스가 자동으로 복제되지 않기 때문입니다.

리전 가용성

명령줄 인터페이스 또는 API 작업을 사용하여 Aurora DB 클러스터로 작업하는 경우 리전 엔드포인트를 지정해야 합니다.

주제

- Aurora MySQL 리전 가용성
- Aurora PostgreSQL 리전 가용성

Aurora MySQL 리전 가용성

다음 표에는 현재 Aurora MySQL을 사용할 수 있는 AWS 리전 및 각 리전의 엔드포인트가 나와 있습니다.

리전 이름	지역	엔드포인트	프로토콜
미국 동부(오하이오)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS
미국 동부(버지니아 북부)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
미국 서부(캘리포니아 북부)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
미국 서부(오레곤)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
아프리카(케이프타운)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
아시아 태평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
아시아 태평양(하이데라바드)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
아시아 태평양(자카르타)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
아시아 태평양(멜버른)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
아시아 태평양(뭄바이)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
아시아 태평양(오사카)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
아시아 태평양(서울)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
아시아 태평양(싱가포르)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
아시아 태평양(시드니)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
아시아 태평양(도쿄)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
캐나다(중부)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
캐나다 서부(캘거리)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
유럽(프랑크푸르트)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
유럽(아일랜드)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
유럽(런던)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
유럽(밀라노)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
유럽(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
유럽(스페인)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
유럽(취리히)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
이스라엘(텔아비브)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
중동(바레인)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
중동(UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
남아메리카(상파울루)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud(미국 동부)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
AWS GovCloud(미국 서부)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS

Aurora PostgreSQL 리전 가용성

다음 표에는 현재 Aurora PostgreSQL을 사용할 수 있는 AWS 리전 및 각 리전의 엔드포인트가 나와 있습니다.

리전 이름	지역	엔드포인트	프로토콜
미국 동부 (오하이오)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS
미국 동부 (버지니아 북부)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
미국 서부 (캘리포니아 북부)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
미국 서부 (오레곤)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
아프리카 (케이프타운)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
아시아 태평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
아시아 태평양(하이데라바드)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
아시아 태평양(자카르타)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
아시아 태평양(멜버른)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS
아시아 태평양(뭄바이)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
아시아 태평양(오사카)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
아시아 태평양(서울)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
아시아 태평양(싱가포르)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
아시아 태평양(시드니)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
아시아 태평양(도쿄)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
캐나다(중부)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
캐나다 서부(캘거리)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
유럽(프랑크푸르트)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
유럽(아일랜드)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
유럽(런던)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
유럽(밀라노)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
유럽(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
유럽(스페인)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
유럽(취리히)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
이스라엘(텔아비브)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
중동(바레인)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
중동 (UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
남아메리카(상파울루)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud(미국 동부)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud(미국 서부)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS

가용 영역

가용 영역은 지정된 AWS 리전에 격리된 위치입니다. 각 리전에는 리전에 대한 고가용성을 제공하도록 설계된 여러 가용 영역(AZ)이 있습니다. AZ는 식별 문자(예: `us-east-1a`)와 같이 AWS 리전 코드로 식별됩니다. 기본 VPC를 사용하지 않고 VPC 및 서브넷을 생성하는 경우 특정 AZ에서 각 서브넷을 정의합니다. Aurora DB 클러스터를 생성하면 Aurora는 VPC의 DB 서브넷 그룹에 있는 서브넷 중 하나에 기본 인스턴스를 생성합니다. 따라서 해당 인스턴스를 Aurora에서 선택한 특정 AZ와 연결합니다.

각 Aurora DB 클러스터는 DB 서브넷 그룹의 AZ 중에서 Aurora가 자동으로 선택하는 세 개의 AZ에 스토리지의 복사본을 각각 호스팅합니다. 클러스터의 모든 DB 인스턴스는 이 3개의 AZ 중 하나에 있어야 합니다.

클러스터에 DB 인스턴스를 생성할 때 AZ를 지정하지 않으면 Aurora에서 해당 인스턴스에 적절한 AZ를 자동으로 선택합니다.

다음과 같이 describe-availability-zones Amazon EC2 명령을 사용하여 계정에 대해 활성화된 지정된 리전 내의 가용 영역을 설명합니다.

```
aws ec2 describe-availability-zones --region region-name
```

예를 들어 계정에 활성화된 미국 동부(버지니아 북부) 리전(us-east-1) 내의 가용 영역을 설명하려면 다음 명령을 실행합니다.

```
aws ec2 describe-availability-zones --region us-east-1
```

클러스터를 생성하거나 클러스터에 인스턴스를 추가할 때 AZ를 지정하는 방법에 대한 자세한 내용은 DB 클러스터의 네트워크 구성 섹션을 참조하세요.

Amazon Aurora DB 클러스터의 현지 시간대

기본적으로 Amazon Aurora DB 클러스터의 시간대는 협정 세계시(UTC)입니다. 대신 DB 클러스터의 인스턴스 시간대를 애플리케이션의 현지 시간대로 설정할 수 있습니다.

DB 클러스터의 현지 시간대를 설정하려면 시간대 파라미터를 지원되는 값 중 하나로 설정합니다. 이 파라미터 값은 DB 클러스터의 파라미터 그룹에서 설정합니다.

- Aurora MySQL의 경우, 이 파라미터의 이름은 `time_zone`입니다. `time_zone` 파라미터 설정 모범 사례에 대한 자세한 내용은 타임스탬프 작업 최적화 섹션을 참조하세요.
- Aurora PostgreSQL의 경우, 이 파라미터의 이름은 `timezone`입니다.

DB 클러스터에 대한 시간대 파라미터를 설정하면 DB 클러스터의 모든 인스턴스가 새로운 현지 시간대를 사용하도록 변경됩니다. 경우에 따라 다른 Aurora DB 클러스터가 동일한 클러스터 파라미터 그룹을 사용하고 있을 수 있습니다. 이 경우 해당 DB 클러스터의 모든 인스턴스도 새로운 현지 시간대를 사용하도록 변경됩니다. 클러스터 수준 파라미터에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 단원을 참조하십시오.

현지 시간대를 설정하면 데이터베이스에 대한 모든 새 연결에 변경 사항이 반영됩니다. 현지 시간대를 변경할 때 데이터베이스에 대해 열린 연결이 있는 경우도 있습니다. 그렇다면 연결을 닫고 새 연결을 열어야 현지 시간대 업데이트가 표시됩니다.

AWS 리전 간에 복제하는 경우 복제 소스 DB 클러스터와 복제본이 서로 다른 파라미터 그룹을 사용합니다. 파라미터 그룹은 AWS 지역에 고유합니다. 각 인스턴스에 대해 동일한 현지 시간대를 사용하려면 복제 소스와 복제본 모두의 파라미터 그룹에서 시간대 파라미터를 설정해야 합니다.

DB 클러스터 스냅샷에서 DB 클러스터를 복원할 경우 현지 시간대가 UTC로 설정됩니다. 복원이 완료된 후 시간대를 현지 시간대로 업데이트할 수 있습니다. DB 클러스터를 특정 시점으로 복원하기도 합니다. 이 경우 복원된 DB 클러스터의 현지 시간대는 복원된 DB 클러스터의 파라미터 그룹에서 설정한 시간대입니다.

다음 테이블에는 현지 시간대를 설정할 수 있는 값의 일부가 나열되어 있습니다. 사용 가능한 모든 시간대를 나열하려면 다음 SQL 쿼리를 사용할 수 있습니다.

- Aurora MySQL: `select * from mysql.time_zone_name;`
- Aurora PostgreSQL: `select * from pg_timezone_names;`

> **Note**
> 일부 시간대의 경우 표에 설명된 대로 특정 날짜 범위 시간 값이 잘못 보고될 수 있습니다. 호주 시간대의 경우 표에 설명된 대로 반환된 시간대 약어가 만료된 값입니다.

시간대	참고
`Africa/Harare`	시간대 설정이 1903년 2월 28일 21:49:40 GMT에서 1903년 2월 28일 21:55:48 GMT까지 잘못된 값을 반환할 수 있습니다.
`Africa/Monrovia`	
`Africa/Nairobi`	시간대 설정이 1939년 12월 31일 21:30:00 GMT에서 1959년 12월 31일 21:15:15 GMT까지 잘못된 값을 반환할 수 있습니다.
`Africa/Windhoek`	
`America/Bogota`	시간대 설정이 1914년 11월 23일 04:56:16 GMT에서 1914년 11월 23일 04:56:20 GMT까지 잘못된 값을 반환할 수 있습니다.
`America/Caracas`	
`America/Chihuahua`	
`America/Cuiaba`	
`America/Denver`	

시간대	참고
`America/Fortaleza`	남아메리카(상파울루) 리전에 있는 DB 클러스터의 경우 최근에 변경된 브라질 시간대의 시간이 제대로 표시되지 않을 수도 있습니다. 그럴 경우 DB 클러스터의 시간대 파라미터를 `America/Fortaleza` 로 재설정하세요.
`America/Guatemala`	
`America/Halifax`	시간대 설정이 1918년 10월 27일 05:00:00 GMT에서 1918년 10월 31일 05:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
`America/Manaus`	DB 클러스터가 남아메리카(쿠이아바) 시간대에 있고 최근에 변경된 브라질 시간대의 예상 시간이 제대로 표시되지 않으면 DB 클러스터의 시간대 파라미터를 `America/Manaus` 로 재설정합니다.
`America/Matamoros`	
`America/Monterrey`	
`America/Montevideo`	
`America/Phoenix`	
`America/Tijuana`	
`Asia/Ashgabat`	
`Asia/Baghdad`	
`Asia/Baku`	
`Asia/Bangkok`	
`Asia/Beirut`	
`Asia/Calcutta`	

시간대	참고
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	시간대 설정이 1919년 12월 31일 20:05:36 GMT에서 1919년 12월 31일 20:05:40 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Riyadh	시간대 설정이 1947년 3월 13일 20:53:08 GMT에서 1949년 12월 31일 20:53:08 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Seoul	시간대 설정이 1904년 11월 30일 15:30:00 GMT에서 1945년 9월 7일 15:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Shanghai	시간대 설정이 1927년 12월 31일 15:54:08 GMT에서 1940년 6월 2일 16:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Singapore	
Asia/Taipei	시간대 설정이 1937년 9월 30일 16:00:00 GMT에서 1979년 9월 29일 15:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Tehran	
Asia/Tokyo	시간대 설정이 1937년 9월 30일 15:00:00 GMT에서 1937년 12월 31일 15:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Asia/Ulaanbaatar	
Atlantic/Azores	시간대 설정이 1911년 5월 24일 01:54:32 GMT에서 1912년 1월 1일 01:54:32 GMT까지 잘못된 값을 반환할 수 있습니다.
Australia/Adelaide	이 시간대의 약어는 ACDT/ACST가 아닌 CST로 반환됩니다.

시간대	참고
Australia/Brisbane	이 시간대의 약어는 AEDT/AEST가 아닌 EST로 반환됩니다.
Australia/Darwin	이 시간대의 약어는 ACDT/ACST가 아닌 CST로 반환됩니다.
Australia/Hobart	이 시간대의 약어는 AEDT/AEST가 아닌 EST로 반환됩니다.
Australia/Perth	이 시간대의 약어는 AWDT/AWST가 아닌 WST로 반환됩니다.
Australia/Sydney	이 시간대의 약어는 AEDT/AEST가 아닌 EST로 반환됩니다.
Brazil/East	
Canada/Saskatchewan	시간대 설정이 1918년 10월 27일 08:00:00 GMT에서 1918년 10월 31일 08:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	시간대 설정이 1921년 4월 30일 22:20:08 GMT에서 1921년 4월 30일 22:20:11 GMT까지 잘못된 값을 반환할 수 있습니다.
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	

시간대	참고
Pacific/Guam	
Pacific/Honolulu	시간대 설정이 1933년 5월 21일 11:30:00 GMT에서 1945년 9월 30일 11:30:00 GMT까지 잘못된 값을 반환할 수 있습니다.
Pacific/Samoa	시간대 설정이 1911년 1월 1일 11:22:48 GMT에서 1950년 1월 1일 11:30:00 GMT까지 잘못된 값을 반환할 수 있습니다.
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

Amazon Aurora에서 AWS 리전 및 Aurora DB 엔진별 지원 기능

Aurora MySQL 및 PostgreSQL 호환 데이터베이스 엔진은 여러 Amazon Aurora 및 Amazon RDS 기능과 옵션을 지원합니다. 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전마다 다릅니다. 주어진 AWS 리전 내에서 Aurora 데이터베이스 엔진 버전 지원 및 가용성을 확인하려면 다음 섹션을 참조하세요.

이러한 기능 중 일부는 Aurora 전용 기능입니다. 예를 들어 Aurora Serverless, Aurora Global Database 및 AWS 기계 학습 서비스와의 통합 지원은 Amazon RDS에서 지원되지 않습니다. Amazon RDS Proxy와 같은 다른 기능은 Amazon Aurora 및 Amazon RDS에서 모두 지원됩니다.

지원되는 리전 및 DB 엔진

- 테이블 규칙
- 블루/그린 배포를 지원하는 리전 및 Aurora DB 엔진
- 클러스터 스토리지 구성을 지원하는 리전 및 Aurora DB 엔진
- 데이터베이스 활동 스트림을 지원하는 리전 및 Aurora DB 엔진
- Amazon S3로 클러스터 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진
- Amazon S3로 스냅샷 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진
- Aurora Global Database를 지원하는 리전 및 DB 엔진
- IAM 데이터베이스 인증을 지원하는 리전 및 Aurora DB 엔진
- Kerberos 인증을 지원하는 리전 및 Aurora DB 엔진
- Aurora 기계 학습을 지원하는 리전 및 DB 엔진
- 성능 개선 도우미를 지원하는 리전 및 Aurora DB 엔진
- Amazon Redshift와 제로 ETL 통합을 지원하는 리전 및 Aurora DB 엔진
- Amazon RDS 프록시를 지원하는 리전 및 Aurora DB 엔진
- Secrets Manager 통합을 지원하는 리전 및 Aurora DB 엔진
- Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진
- Aurora Serverless v1
- RDS Data API를 지원하는 리전 및 Aurora DB 엔진
- 제로 가동 중지 패치 적용(ZDP)을 지원하는 리전 및 Aurora DB 엔진
- Aurora 엔진 네이티브 기능을 지원하는 리전 및 DB 엔진

테이블 규칙

이 기능 섹션의 테이블은 다음 패턴을 사용하여 버전 번호 및 지원 수준을 지정합니다:

- 버전 x.y – 특정 버전을 단독으로 지원합니다.
- 버전 x.y 이상 – 지정된 버전 및 해당 버전의 메이저 버전에 속하며 지정된 버전보다 높은 모든 마이너 버전도 지원합니다. 예를 들어 "버전 10.11 이상"은 버전 10.11, 10.11.1은 물론 10.12도 지원됨을 의미합니다.

블루/그린 배포를 지원하는 리전 및 Aurora DB 엔진

블루/그린 배포는 프로덕션 데이터베이스 환경을 별도의 동기화된 스테이징 환경에 복사합니다. Amazon RDS 블루/그린 배포를 사용하면 프로덕션 환경에 영향을 주지 않고 스테이징 환경에서 데이터베이스를 변경할 수 있습니다. 예를 들어 메이저 또는 마이너 DB 엔진 버전을 업그레이드하거나, 데이터베이스 파라미터를 변경하거나, 스테이징 환경 내 스키마를 변경할 수 있습니다. 준비가 되면 스테이징 환경을 새 프로덕션 데이터베이스 환경으로 승격할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 단원을 참조하십시오.

Aurora MySQL을 사용한 블루/그린 배포

블루/그린 배포 기능은 모든 AWS 리전의 전체 Aurora MySQL 버전에서 사용할 수 있습니다.

Aurora PostgreSQL을 사용한 블루/그린 배포

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 블루/그린 배포에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
모든 AWS 리전	버전 16.1 이상	버전 15.4 이상	버전 14.9 이상	버전 13.12 이상	버전 12.16 이상	버전 11.21 이상

클러스터 스토리지 구성을 지원하는 리전 및 Aurora DB 엔진

Amazon Aurora에는 Aurora I/O-Optimized 및 Aurora Standard의 두 가지 DB 클러스터 스토리지 구성이 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 단원을 참조하십시오.

Aurora I/O-Optimized

Aurora I/O-Optimized는 다음과 같은 Amazon Aurora 버전에 대한 모든 AWS 리전에서 사용할 수 있습니다.

- Aurora MySQL 버전 3.03.1 이상
- Aurora PostgreSQL 버전 16.1 이상, 15.2 이상, 14.7 이상, 13.10 이상

Aurora Standard

Aurora Standard는 모든 Aurora MySQL 및 Aurora PostgreSQL 버전의 모든 AWS 리전에서 사용할 수 있습니다.

데이터베이스 활동 스트림을 지원하는 리전 및 Aurora DB 엔진

Aurora에서 데이터베이스 활동 스트림을 사용하면 Aurora 데이터베이스에서 감사 활동에 대한 경보를 모니터링하고 설정할 수 있습니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 단원을 참조하십시오.

다음 기능에서는 데이터베이스 활동 스트림이 지원되지 않습니다.

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish for Aurora PostgreSQL

주제

- Amazon MySQL을 사용하는 데이터베이스 활동 스트림
- Amazon PostgreSQL을 사용하는 데이터베이스 활동 스트림

Amazon MySQL을 사용하는 데이터베이스 활동 스트림

다음 리전 및 엔진 버전을 Aurora MySQL을 사용하는 데이터베이스 활동 스트림에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	사용 가능한 모든 버전	Aurora 버전 2.11 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(버지니아 북부)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
미국 서부(캘리포니아 북부)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
미국 서부(오레곤)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아프리카(케이프타운)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(홍콩)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(하이데라바드)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(자카르타)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(뭄바이)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(오사카)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(서울)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(싱가포르)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(시드니)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
아시아 태평양(도쿄)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
캐나다(중부)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음
중국(베이징)	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	사용할 수 없음
유럽(프랑크푸르트)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(아일랜드)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(런던)	사용 가능한 모든 버전	Aurora 버전 2.11 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
유럽(밀라노)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(파리)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(스페인)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(스톡홀름)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
유럽(취리히)	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음
중동(바레인)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
중동(UAE)	사용 가능한 모든 버전	Aurora 버전 2.11 이상
남아메리카(상파울루)	사용 가능한 모든 버전	Aurora 버전 2.11 이상

Amazon PostgreSQL을 사용하는 데이터베이스 활동 스트림

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용하는 데이터베이스 활동 스트림에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 동부(오하이오)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
미국 동부(버지니아 북부)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 서부(캘리포니아 북부)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
미국 서부(오레곤)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아프리카(케이프타운)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(홍콩)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(하이데라바드)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(자카르타)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(뭄바이)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(오사카)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(서울)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(싱가포르)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(시드니)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
아시아 태평양(도쿄)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
캐나다(중부)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중국(베이징)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(프랑크푸르트)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
유럽(아일랜드)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(런던)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(밀라노)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(파리)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(스페인)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(스톡홀름)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
유럽(취리히)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(바레인)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
중동(UAE)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상
남아메리카(상파울루)	버전 16.1 이상	버전 15.2 이상	사용 가능한 모든 버전	사용 가능한 모든 버전	사용 가능한 모든 버전	버전 11.9 및 버전 11.13 이상

Amazon S3로 클러스터 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진

Aurora DB 클러스터 데이터를 Amazon S3 버킷으로 내보낼 수 있습니다. 데이터를 내보낸 후에는 Amazon Athena 또는 Amazon Redshift Spectrum 같은 도구를 통해 직접 내보낸 데이터를 분석할 수 있습니다. 자세한 내용은 Amazon S3로 DB 클러스터 데이터 내보내기 단원을 참조하십시오.

클러스터 데이터를 S3로 내보내는 기능은 다음 AWS 리전에서 사용할 수 있습니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오레곤)
- 아시아 태평양(홍콩)
- 아시아 태평양(하이데라바드)
- 아시아 태평양(자카르타)
- 아시아 태평양(멜버른)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)

- 아시아 태평양(도쿄)
- 캐나다(중부)
- 캐나다 서부(캘거리)
- 중국(닝샤)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스페인)
- 유럽(스톡홀름)
- 유럽(취리히)
- 이스라엘(텔아비브)
- 중동(UAE)
- 남아메리카(상파울루)

주제

- Aurora MySQL을 이용해 클러스터 데이터를 S3로 내보내기
- Aurora PostgreSQL을 이용해 클러스터 데이터를 S3로 내보내기

Aurora MySQL을 이용해 클러스터 데이터를 S3로 내보내기

현재 사용 가능한 모든 Aurora MySQL 엔진 버전은 기존 DB 클러스터 데이터를 Amazon S3로 내보내는 작업을 지원합니다. 버전에 대한 자세한 내용은 Aurora MySQL 릴리스 정보를 참조하세요.

Aurora PostgreSQL을 이용해 클러스터 데이터를 S3로 내보내기

현재 사용 가능한 모든 Aurora PostgreSQL 엔진 버전은 기존 DB 클러스터 데이터를 Amazon S3로 내보내는 작업을 지원합니다. 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

Amazon S3로 스냅샷 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진

Aurora DB 클러스터 스냅샷 데이터를 Amazon S3 버킷으로 내보낼 수 있습니다. 수동 스냅샷 및 자동화된 시스템 스냅샷을 내보낼 수 있습니다. 데이터를 내보낸 후에는 Amazon Athena 또는 Amazon

Redshift Spectrum 같은 도구를 통해 직접 내보낸 데이터를 분석할 수 있습니다. 자세한 내용은 Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기 단원을 참조하십시오.

S3로 스냅샷 내보내기는 다음을 제외한 모든 AWS 리전에서 사용할 수 있습니다.

- 아시아 태평양(말레이시아)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

주제

- Aurora MySQL을 이용해 스냅샷 데이터를 S3로 내보내기
- Aurora PostgreSQL을 이용해 스냅샷 데이터를 S3로 내보내기

Aurora MySQL을 이용해 스냅샷 데이터를 S3로 내보내기

현재 사용 가능한 모든 Aurora MySQL 엔진 버전은 기존 DB 클러스터 스냅샷 데이터를 Amazon S3로 내보내는 작업을 지원합니다. 버전에 대한 자세한 내용은 Aurora MySQL 릴리스 정보를 참조하세요.

Aurora PostgreSQL을 이용해 스냅샷 데이터를 S3로 내보내기

현재 사용 가능한 모든 Aurora PostgreSQL 엔진 버전은 DB 클러스터 스냅샷 데이터를 Amazon S3로 내보내는 작업을 지원합니다. 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

Aurora Global Database를 지원하는 리전 및 DB 엔진

Aurora Global Database는 다중 AWS 리전으로 확장할 수 있는 단일 데이터베이스로, 짧은 대기 시간의 글로벌 읽기를 지원하며 리전 규모의 가동 중단 발생 시 재해 복구를 제공합니다. DB 인스턴스는 단일 AWS 리전이 아니라 여러 리전 및 가용 영역에 의존하기 때문에, 배포한 내용에 자체 결함 용인을 제공합니다. 자세한 내용은 Amazon Aurora Global Database 사용 섹션을 참조하세요.

주제

- Aurora MySQL을 사용하는 Aurora 전역 데이터베이스
- Aurora PostgreSQL을 사용하는 Aurora 전역 데이터베이스

Aurora MySQL을 사용하는 Aurora 전역 데이터베이스

다음 리전 및 엔진 버전을 Aurora MySQL을 사용하는 Aurora Global Database에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(버지니아 북부)	버전 3.01.0 이상	버전 2.07.0 이상
미국 동부(오하이오)	버전 3.01.0 이상	버전 2.07.0 이상
미국 서부(캘리포니아 북부)	버전 3.01.0 이상	버전 2.07.0 이상
미국 서부(오리건)	버전 3.01.0 이상	버전 2.07.0 이상
아프리카(케이프타운)	버전 3.01.0 이상	버전 2.07.1 이상
아시아 태평양(홍콩)	버전 3.01.0 이상	버전 2.07.1 이상
아시아 태평양(하이데라바드)	버전 3.02.0 이상	버전 2.11.2 이상
아시아 태평양(자카르타)	버전 3.01.0 이상	버전 2.07.6 이상
아시아 태평양(말레이시아)	버전 3.04.0 이상	버전 2.07.6 이상
아시아 태평양(멜버른)	버전 3.03.0 이상	버전 2.07.6 이상
아시아 태평양(뭄바이)	버전 3.01.0 이상	버전 2.07.0 이상
아시아 태평양(오사카)	버전 3.01.0 이상	버전 2.07.3 이상
아시아 태평양(서울)	버전 3.01.0 이상	버전 2.07.0 이상
아시아 태평양(싱가포르)	버전 3.01.0 이상	버전 2.07.0 이상
아시아 태평양(시드니)	버전 3.01.0 이상	버전 2.07.0 이상
아시아 태평양(도쿄)	버전 3.01.0 이상	버전 2.07.0 이상
캐나다(중부)	버전 3.01.0 이상	버전 2.07.0 이상
캐나다 서부(캘거리)	버전 3.01.0 이상	버전 2.07.0 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
중국(베이징)	버전 3.01.0 이상	버전 2.07.2 이상
중국(닝샤)	버전 3.01.0 이상	버전 2.07.2 이상
유럽(프랑크푸르트)	버전 3.01.0 이상	버전 2.07.0 이상
유럽(아일랜드)	버전 3.01.0 이상	버전 2.07.0 이상
유럽(런던)	버전 3.01.0 이상	버전 2.07.0 이상
유럽(밀라노)	버전 3.01.0 이상	버전 2.07.1 이상
유럽(파리)	버전 3.01.0 이상	버전 2.07.0 이상
유럽(스페인)	버전 3.02.0 이상	버전 2.07.6 이상
유럽(스톡홀름)	버전 3.01.0 이상	버전 2.07.0 이상
유럽(취리히)	버전 3.02.0 이상	버전 2.07.6 이상
이스라엘(텔아비브)	버전 3.02.0 이상	버전 2.07.6 이상
중동(바레인)	버전 3.01.0 이상	버전 2.07.1 이상
중동(UAE)	버전 3.02.0 이상	버전 2.07.6 이상
남아메리카(상파울루)	버전 3.01.0 이상	버전 2.07.1 이상
AWS GovCloud(미국 동부)	버전 3.01.0 이상	버전 2.07.0 이상
AWS GovCloud(미국 서부)	버전 3.01.0 이상	버전 2.07.0 이상

Aurora PostgreSQL을 사용하는 Aurora 전역 데이터베이스

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용하는 Aurora Global Database에 사용할 수 있습니다.

리전	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 동부 (버지니아 북부)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 동부 (오하이오)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 서부 (캘리포니아 북부)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 서부 (오리건)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아프리카 (케이프타운)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(홍콩)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(하이데라바드)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(자카르타)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(말레이시아)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(멜버른)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(뭄바이)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(오사카)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(서울)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(싱가포르)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(시드니)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(도쿄)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
캐나다(중부)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
캐나다 서부(캘거리)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중국(베이징)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중국(닝샤)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(프랑크푸르트)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
유럽(아일랜드)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(런던)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(밀라노)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(파리)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(스페인)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(스톡홀름)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(취리히)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(바레인)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중동(UAE)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
남아메리카(상파울루)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
AWS GovCloud(미국 동부)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
AWS GovCloud(미국 서부)	버전 17.4 이상	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

IAM 데이터베이스 인증을 지원하는 리전 및 Aurora DB 엔진

Aurora에서 IAM 데이터베이스 인증을 사용하면 AWS Identity and Access Management(IAM) 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다. 이러한 인증 방식은 DB 클러스터에 연결

할 때 암호를 사용할 필요 없습니다. 대신에 인증 토큰을 사용합니다. 자세한 내용은 IAM 데이터베이스 인증 단원을 참조하십시오.

주제

- Aurora MySQL을 사용하는 IAM 데이터베이스 인증
- Aurora PostgreSQL을 사용하는 IAM 데이터베이스 인증

Aurora MySQL을 사용하는 IAM 데이터베이스 인증

Aurora MySQL을 사용하는 IAM 데이터베이스 인증은 다음 버전의 경우 모든 리전에서 사용할 수 있습니다.

- Aurora MySQL 3 – 사용 가능한 모든 버전
- Aurora MySQL 2 – 사용 가능한 모든 버전

Aurora PostgreSQL을 사용하는 IAM 데이터베이스 인증

Aurora PostgreSQL을 사용하는 IAM 데이터베이스 인증은 다음 엔진 버전의 경우 모든 리전에서 사용할 수 있습니다.

- Aurora PostgreSQL 16 – 사용 가능한 모든 버전
- Aurora PostgreSQL 15 – 사용 가능한 모든 버전
- Aurora PostgreSQL 14 – 사용 가능한 모든 버전
- Aurora PostgreSQL 13 – 사용 가능한 모든 버전
- Aurora PostgreSQL 12 – 사용 가능한 모든 버전
- Aurora PostgreSQL 11 – 사용 가능한 모든 버전

Kerberos 인증을 지원하는 리전 및 Aurora DB 엔진

Aurora를 통해 Kerberos 인증을 사용하면 Kerberos 및 Microsoft Active Directory를 통해 데이터베이스 사용자의 외부 인증을 사용할 수 있습니다. Kerberos 및 Active Directory는 데이터베이스 사용자에게 SSO(Single Sign-On) 및 중앙 집중식 인증의 이점을 제공합니다. Kerberos 및 Active Directory는 AWS Directory Service의 기능인 AWS Directory Service for Microsoft Active Directory와 함께 사용할 수 있습니다. 자세한 내용은 Kerberos 인증 단원을 참조하십시오.

주제

- [Aurora MySQL을 사용하는 Kerberos 인증]
- [Aurora PostgreSQL을 사용하는 Kerberos 인증]

Aurora MySQL을 사용하는 Kerberos 인증

다음 리전 및 엔진 버전을 Aurora MySQL을 사용한 Kerberos 인증에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3
미국 동부(오하이오)	버전 3.03.0 이상
미국 동부(버지니아 북부)	버전 3.03.0 이상
미국 서부(캘리포니아 북부)	버전 3.03.0 이상
미국 서부(오레곤)	버전 3.03.0 이상
아프리카(케이프타운)	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음
아시아 태평양(뭄바이)	버전 3.03.0 이상
아시아 태평양(오사카)	사용할 수 없음
아시아 태평양(서울)	버전 3.03.0 이상
아시아 태평양(싱가포르)	버전 3.03.0 이상
아시아 태평양(시드니)	버전 3.03.0 이상
아시아 태평양(도쿄)	버전 3.03.0 이상
캐나다(중부)	버전 3.03.0 이상
캐나다 서부(캘거리)	사용할 수 없음
중국(베이징)	버전 3.03.0 이상

리전	Aurora MySQL 버전 3
중국(닝샤)	버전 3.03.0 이상
유럽(프랑크푸르트)	버전 3.03.0 이상
유럽(아일랜드)	버전 3.03.0 이상
유럽(런던)	버전 3.03.0 이상
유럽(밀라노)	사용할 수 없음
유럽(파리)	버전 3.03.0 이상
유럽(스페인)	사용할 수 없음
유럽(스톡홀름)	버전 3.03.0 이상
유럽(취리히)	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음
중동(바레인)	사용할 수 없음
중동(UAE)	사용할 수 없음
남아메리카(상파울루)	버전 3.03.0 이상
AWS GovCloud(미국 동부)	버전 3.03.0 이상
AWS GovCloud(미국 서부)	버전 3.03.0 이상

Aurora PostgreSQL을 사용하는 Kerberos 인증

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 Kerberos 인증에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 동부(오하이오)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 동부(버지니아 북부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 서부(캘리포니아 북부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 서부(오레곤)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(뭄바이)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(오사카)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(서울)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(싱가포르)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(시드니)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(도쿄)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
캐나다(중부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중국(베이징)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
중국(닝샤)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(프랑크푸르트)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
유럽(아일랜드)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(런던)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(밀라노)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(파리)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(스페인)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(스톡홀름)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(취리히)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(바레인)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(UAE)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
남아메리카(상파울루)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
AWS GovCloud(미국 동부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud(미국 서부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

Aurora 기계 학습을 지원하는 리전 및 DB 엔진

Amazon Aurora 기계 학습을 사용하면 필요에 따라 Aurora DB 클러스터를 다음 AWS 기계 학습 서비스 중 하나와 통합할 수 있습니다. 각 서비스마다 특정 기계 학습 사용 사례를 지원합니다.

Amazon Bedrock은 생성형 AI 애플리케이션을 구축하고 확장하는 데 도움이 되는 개발자 도구와 함께 API를 통해 AI 회사의 주요 기반 모델을 사용할 수 있게 해주는 완전 관리형 서비스입니다.

Amazon Comprehend는 문서에서 인사이트를 추출하는 데 사용되는 자연어 처리(NLP) 서비스입니다. Amazon Comprehend와 함께 Aurora 기계 학습을 사용하면 데이터베이스 테이블의 텍스트 감정을 판단할 수 있습니다.

SageMaker는 종합적인 기능을 갖춘 기계 학습 서비스입니다. 데이터 과학자는 Amazon SageMaker를 사용하여 사기 탐지와 같은 다양한 추론 작업을 위한 기계 학습 모델을 구축하고 훈련하고 테스트합니다. 데이터베이스 개발자는 SageMaker와 함께 Aurora 기계 학습을 사용하여 SQL 코드에서 SageMaker 기능을 호출할 수 있습니다.

모든 AWS 리전에서 모든 기계 학습 서비스를 지원하는 것은 아닙니다. 일부 AWS 리전에서만 Aurora 기계 학습을 지원하여 Aurora DB 클러스터에서 이러한 서비스에 대한 액세스를 제공합니다. Aurora 기계 학습의 통합 프로세스도 데이터베이스 엔진에 따라 다릅니다. 자세한 내용은 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.

주제

- Aurora MySQL을 사용하는 Aurora 기계 학습
- Aurora PostgreSQL을 사용하는 Aurora 기계 학습

Aurora MySQL을 사용하는 Aurora 기계 학습

Amazon Bedrock은 Aurora MySQL 버전 3.06 이상에서만 지원됩니다. Amazon Bedrock의 리전 가용성에 대한 자세한 내용은 Amazon Bedrock 사용 설명서에서 AWS 리전에서 지원하는 모델을 참조하세요.

표에 나열된 AWS 리전에서 Aurora MySQL은 Amazon Comprehend 및 Amazon SageMaker를 사용한 Aurora 기계 학습을 지원합니다. 사용 중인 Aurora MySQL 버전을 사용 가능하게 하는 것 외에 AWS 리전 또한 사용하려는 서비스를 지원해야 합니다. Amazon SageMaker를 사용할 수 있는 AWS 리전 목록은 Amazon Web Services 일반 참조의 Amazon SageMaker 엔드포인트 및 할당량을 참조하세요. Amazon Comprehend를 사용할 수 있는 AWS 리전 목록은 Amazon Web Services 일반 참조의 Amazon Comprehend 엔드포인트 및 할당량을 참조하세요.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	버전 3.01.0 이상	버전 2.07 이상
미국 동부(버지니아 북부)	버전 3.01.0 이상	버전 2.07 이상
미국 서부(캘리포니아 북부)	버전 3.01.0 이상	버전 2.07 이상
미국 서부(오레곤)	버전 3.01.0 이상	버전 2.07 이상
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(하이데라바드)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(자카르타)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(멜버른)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(뭄바이)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(오사카)	버전 3.01.0 이상	버전 2.07.3 이상
아시아 태평양(서울)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(싱가포르)	버전 3.01.0 이상	버전 2.07 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(시드니)	버전 3.01.0 이상	버전 2.07 이상
아시아 태평양(도쿄)	버전 3.01.0 이상	버전 2.07 이상
캐나다(중부)	버전 3.01.0 이상	버전 2.07 이상
캐나다 서부(캘거리)	버전 3.01.0 이상	버전 2.07 이상
중국(베이징)	버전 3.01.0 이상	버전 2.07 이상
중국(닝샤)	버전 3.01.0 이상	버전 2.07 이상
유럽(프랑크푸르트)	버전 3.01.0 이상	버전 2.07 이상
유럽(아일랜드)	버전 3.01.0 이상	버전 2.07 이상
유럽(런던)	버전 3.01.0 이상	버전 2.07 이상
유럽(밀라노)	사용할 수 없음	사용할 수 없음
유럽(파리)	버전 3.01.0 이상	버전 2.07 이상
유럽(스페인)	버전 3.01.0 이상	버전 2.07 이상
유럽(스톡홀름)	버전 3.01.0 이상	버전 2.07 이상
유럽(취리히)	버전 3.01.0 이상	버전 2.07 이상
이스라엘(텔아비브)	버전 3.01.0 이상	버전 2.07 이상
중동(바레인)	버전 3.01.0 이상	버전 2.07 이상
중동(UAE)	버전 3.01.0 이상	버전 2.07 이상
남아메리카(상파울루)	버전 3.01.0 이상	버전 2.07 이상
AWS GovCloud(미국 동부)	버전 3.01.0 이상	버전 2.07 이상
AWS GovCloud(미국 서부)	버전 3.01.0 이상	버전 2.07 이상

Aurora PostgreSQL을 사용하는 Aurora 기계 학습

Aurora PostgreSQL 기반 Amazon Bedrock의 버전 지원에 대한 자세한 내용은 Aurora PostgreSQL을 Amazon Bedrock의 지식 기반으로 사용 섹션을 참조하세요.

표에 나열된 AWS 리전에서 Aurora PostgreSQL은 Amazon Comprehend 및 Amazon SageMaker를 사용한 Aurora 기계 학습을 지원합니다. 사용 중인 Aurora PostgreSQL 버전을 사용 가능하게 하는 것 외에 AWS 리전 또한 사용하려는 서비스를 지원해야 합니다. Amazon SageMaker를 사용할 수 있는 AWS 리전 목록은 Amazon Web Services 일반 참조의 Amazon SageMaker 엔드포인트 및 할당량을 참조하세요. Amazon Comprehend를 사용할 수 있는 AWS 리전 목록은 Amazon Web Services 일반 참조의 Amazon Comprehend 엔드포인트 및 할당량을 참조하세요.

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 Aurora 기계 학습에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 동부(오하이오)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
미국 동부(버지니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
미국 서부(캘리포니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
미국 서부(오레곤)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(하이데라바드)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(자카르타)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(멜버른)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(뭄바이)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(오사카)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(서울)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(싱가포르)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(시드니)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
아시아 태평양(도쿄)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
캐나다(중부)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
캐나다 서부(캘거리)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
중국(베이징)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
중국(닝샤)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(프랑크푸르트)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(아일랜드)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(런던)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(밀라노)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(파리)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(스페인)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(스톡홀름)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
유럽(취리히)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
이스라엘(텔아비브)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
중동(바레인)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
중동(UAE)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
남아메리카(상파울루)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
AWS GovCloud(미국 동부)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상
AWS GovCloud(미국 서부)	버전 16.1 이상	버전 15.2 이상	버전 14.3	버전 13.3 이상	버전 12.4 이상	버전 11.9, 11.12 이상

성능 개선 도우미를 지원하는 리전 및 Aurora DB 엔진

성능 개선 도우미는 기존 Amazon RDS 모니터링 기능을 확장하여 데이터베이스 성능을 설명하고 분석하는 데 도움을 줍니다. 성능 개선 도우미 대시보드를 사용하면 Amazon RDS DB 인스턴스 로드를 시각화하고 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 로드를 필터링할 수 있습니다. 자세한 내용은 Amazon Aurora의 성능 개선 도우미 개요 단원을 참조하십시오.

성능 개선 도우미에 대한 리전, DB 엔진 및 인스턴스 클래스 지원 정보는 성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원 섹션을 참조하세요.

주제

- Aurora MySQL을 사용하는 성능 개선 도우미
- Aurora PostgreSQL을 사용하는 성능 개선 도우미
- Aurora Serverless를 사용하는 성능 개선 도우미

Aurora MySQL을 사용하는 성능 개선 도우미

> **Note**
>
> 병렬 쿼리가 활성화되어 있는 경우 Aurora MySQL을 사용하는 성능 개선 도우미에 대한 엔진 버전 지원이 다릅니다. 병렬 쿼리에 대한 자세한 내용은 [Amazon Aurora MySQL용 병렬 쿼리](#) 섹션을 참조하세요.

주제

- [Aurora MySQL을 사용하며 병렬 쿼리가 비활성화되어 있는 경우 성능 개선 도우미](#)
- [Aurora MySQL을 사용하며 병렬 쿼리가 활성화되어 있는 경우 성능 개선 도우미](#)

Aurora MySQL을 사용하며 병렬 쿼리가 비활성화되어 있는 경우 성능 개선 도우미

성능 개선 도우미에 Aurora MySQL을 사용하고 병렬 쿼리를 비활성화한 상태에서 다음 리전 및 엔진 버전을 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	모든 버전	모든 버전
미국 동부(버지니아 북부)	모든 버전	모든 버전
미국 서부(캘리포니아 북부)	모든 버전	모든 버전
미국 서부(오레곤)	모든 버전	모든 버전
아프리카(케이프타운)	모든 버전	모든 버전
아시아 태평양(홍콩)	모든 버전	모든 버전
아시아 태평양(하이데라바드)	모든 버전	모든 버전
아시아 태평양(자카르타)	모든 버전	모든 버전
아시아 태평양(멜버른)	모든 버전	모든 버전
아시아 태평양(뭄바이)	모든 버전	모든 버전

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(오사카)	모든 버전	모든 버전
아시아 태평양(서울)	모든 버전	모든 버전
아시아 태평양(싱가포르)	모든 버전	모든 버전
아시아 태평양(시드니)	모든 버전	모든 버전
아시아 태평양(도쿄)	모든 버전	모든 버전
캐나다(중부)	모든 버전	모든 버전
캐나다 서부(캘거리)	모든 버전	모든 버전
중국(베이징)	모든 버전	모든 버전
중국(닝샤)	모든 버전	모든 버전
유럽(프랑크푸르트)	모든 버전	모든 버전
유럽(아일랜드)	모든 버전	모든 버전
유럽(런던)	모든 버전	모든 버전
유럽(밀라노)	모든 버전	모든 버전
유럽(파리)	모든 버전	모든 버전
유럽(스페인)	모든 버전	모든 버전
유럽(스톡홀름)	모든 버전	모든 버전
유럽(취리히)	모든 버전	모든 버전
이스라엘(텔아비브)	모든 버전	모든 버전
중동(바레인)	모든 버전	모든 버전
중동(UAE)	모든 버전	모든 버전

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
남아메리카(상파울루)	모든 버전	모든 버전
AWS GovCloud(미국 동부)	모든 버전	모든 버전
AWS GovCloud(미국 서부)	모든 버전	모든 버전

Aurora MySQL을 사용하며 병렬 쿼리가 활성화되어 있는 경우 성능 개선 도우미

성능 개선 도우미에 Aurora MySQL을 사용하고 병렬 쿼리를 활성화한 상태에서 다음 리전 및 엔진 버전을 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	사용할 수 없음	버전 2.09.0 이상
미국 동부(버지니아 북부)	사용할 수 없음	버전 2.09.0 이상
미국 서부(캘리포니아 북부)	사용할 수 없음	버전 2.09.0 이상
미국 서부(오레곤)	사용할 수 없음	버전 2.09.0 이상
아프리카(케이프타운)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(홍콩)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(하이데라바드)	사용할 수 없음	모든 버전
아시아 태평양(자카르타)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(멜버른)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(뭄바이)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(오사카)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(서울)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(싱가포르)	사용할 수 없음	버전 2.09.0 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(시드니)	사용할 수 없음	버전 2.09.0 이상
아시아 태평양(도쿄)	사용할 수 없음	버전 2.09.0 이상
캐나다(중부)	사용할 수 없음	버전 2.09.0 이상
캐나다 서부(캘거리)	사용할 수 없음	버전 2.09.0 이상
중국(베이징)	사용할 수 없음	버전 2.09.0 이상
중국(닝샤)	사용할 수 없음	버전 2.09.0 이상
유럽(프랑크푸르트)	사용할 수 없음	버전 2.09.0 이상
유럽(아일랜드)	사용할 수 없음	버전 2.09.0 이상
유럽(런던)	사용할 수 없음	버전 2.09.0 이상
유럽(밀라노)	사용할 수 없음	버전 2.09.0 이상
유럽(파리)	사용할 수 없음	버전 2.09.0 이상
유럽(스페인)	사용할 수 없음	버전 2.09.0 이상
유럽(스톡홀름)	사용할 수 없음	버전 2.09.0 이상
유럽(취리히)	사용할 수 없음	버전 2.09.0 이상
이스라엘(텔아비브)	사용할 수 없음	버전 2.09.0 이상
중동(바레인)	사용할 수 없음	버전 2.09.0 이상
중동(UAE)	사용할 수 없음	버전 2.09.0 이상
남아메리카(상파울루)	사용할 수 없음	버전 2.09.0 이상
AWS GovCloud(미국 동부)	사용할 수 없음	버전 2.09.0 이상
AWS GovCloud(미국 서부)	사용할 수 없음	버전 2.09.0 이상

Aurora PostgreSQL을 사용하는 성능 개선 도우미

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 성능 개선 도우미에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
미국 동부 (오하이오)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 동부 (버지니아 북부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 서부 (캘리포니아 북부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
미국 서부 (오레곤)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아프리카 (케이프타운)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(홍콩)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(하이데라바드)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(자카르타)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
아시아 태평양(멜버른)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(뭄바이)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(오사카)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(서울)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(싱가포르)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(시드니)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
아시아 태평양(도쿄)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
캐나다(중부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
캐나다 서부(캘거리)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
중국(베이징)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
중국(닝샤)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(프랑크푸르트)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(아일랜드)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(런던)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(밀라노)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(파리)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(스페인)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(스톡홀름)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
유럽(취리히)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
이스라엘(텔아비브)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
중동(바레인)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
중동 (UAE)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
남아메리카(상파울루)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
AWS GovCloud(미국 동부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전
AWS GovCloud(미국 서부)	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전	모든 버전

Aurora Serverless를 사용하는 성능 개선 도우미

Aurora Serverless v2는 모든 MySQL 호환 및 PostgreSQL 호환 버전에 대한 성능 개선 도우미를 지원합니다. 최소 용량을 최소 2개의 Aurora 용량 단위(ACU)로 설정하는 것이 좋습니다.

Aurora Serverless v1은 Performance Insights를 지원하지 않습니다.

Amazon Redshift와 제로 ETL 통합을 지원하는 리전 및 Aurora DB 엔진

Amazon Redshift가 구성된 Amazon Aurora 제로 ETL 통합은 트랜잭션 데이터가 Aurora 클러스터에 기록된 후 Amazon Redshift에서 사용할 수 있도록 하기 위한 완전관리형 솔루션입니다. 자세한 내용은 제로 ETL 통합 단원을 참조하십시오.

주제

- [Aurora MySQL 제로 ETL 통합]
- [Aurora PostgreSQL 제로 ETL 통합]

Aurora MySQL 제로 ETL 통합

Amazon Redshift가 구성된 Aurora MySQL 제로 ETL 통합에 사용할 수 있는 리전 및 엔진 버전은 다음과 같습니다.

리전	Aurora MySQL 버전 3
미국 동부(오하이오)	버전 3.05.2 이상
미국 동부(버지니아 북부)	버전 3.05.2 이상
미국 서부(캘리포니아 북부)	버전 3.05.2 이상
미국 서부(오레곤)	버전 3.05.2 이상
아프리카(케이프타운)	버전 3.05.2 이상
아시아 태평양(홍콩)	버전 3.05.2 이상
아시아 태평양(하이데라바드)	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음
아시아 태평양(뭄바이)	버전 3.05.2 이상
아시아 태평양(오사카)	버전 3.05.2 이상
아시아 태평양(서울)	버전 3.05.2 이상
아시아 태평양(싱가포르)	버전 3.05.2 이상
아시아 태평양(시드니)	버전 3.05.2 이상
아시아 태평양(도쿄)	버전 3.05.2 이상
캐나다(중부)	버전 3.05.2 이상

리전	Aurora MySQL 버전 3
캐나다 서부(캘거리)	사용할 수 없음
중국(베이징)	버전 3.05.2 이상
중국(닝샤)	버전 3.05.2 이상
유럽(프랑크푸르트)	버전 3.05.2 이상
유럽(아일랜드)	버전 3.05.2 이상
유럽(런던)	버전 3.05.2 이상
유럽(밀라노)	버전 3.05.2 이상
유럽(파리)	버전 3.05.2 이상
유럽(스페인)	사용할 수 없음
유럽(스톡홀름)	버전 3.05.2 이상
유럽(취리히)	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음
중동(바레인)	버전 3.05.2 이상
중동(UAE)	사용할 수 없음
남아메리카(상파울루)	버전 3.05.2 이상
AWS GovCloud(미국 동부)	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음

Aurora PostgreSQL 제로 ETL 통합

Amazon Redshift가 구성된 Aurora PostgreSQL 제로 ETL 통합의 평가판 릴리스를 사용하려면 미국 동부(오하이오)(us-east-2) AWS 리전에 있는 Amazon RDS 데이터베이스 미리 보기 환경 내에서 통합을

생성해야 합니다. 미리 보기 환경에서는 PostgreSQL 데이터베이스 엔진 소프트웨어의 베타, 릴리스 후보 및 초기 프로덕션 버전을 테스트할 수 있습니다.

소스 DB 클러스터는 Aurora PostgreSQL(PostgreSQL 15.4 및 제로 ETL 지원과 호환)을 실행해야 합니다.

Amazon RDS 프록시를 지원하는 리전 및 Aurora DB 엔진

Amazon RDS 프록시는 설정된 데이터베이스 연결을 풀링하고 공유하여 응용 프로그램의 확장성을 높여주는 완전히 관리되는 고가용성 데이터베이스 프록시입니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

주제

- Aurora MySQL과 Amazon RDS 프록시 사용
- Aurora PostgreSQL과 Amazon RDS 프록시

Aurora MySQL과 Amazon RDS 프록시 사용

다음 리전 및 엔진 버전을 Aurora MySQL을 사용한 RDS 프록시에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
미국 동부(버지니아 북부)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
미국 서부(캘리포니아 북부)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
미국 서부(오레곤)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아프리카(케이프타운)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(홍콩)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(하이데라바드)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(자카르타)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(멜버른)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(뭄바이)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(오사카)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(서울)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(싱가포르)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(시드니)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
아시아 태평양(도쿄)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
캐나다(중부)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
캐나다 서부(캘거리)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
중국(베이징)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
중국(닝샤)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(프랑크푸르트)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(아일랜드)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(런던)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(밀라노)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(파리)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(스페인)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(스톡홀름)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
유럽(취리히)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
이스라엘(텔아비브)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
중동(바레인)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
중동(UAE)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
남아메리카(상파울루)	버전 3.01.0 이상	버전 2.07 및 버전 2.11 이상
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음

Aurora PostgreSQL과 Amazon RDS 프록시

Aurora PostgresSQL을 사용하는 RDS 프록시에 지원되는 엔진 및 리전 가용성은 다음과 같습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
미국 동부(오하이오)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 동부(버지니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 서부(캘리포니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
미국 서부(오레곤)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아프리카(케이프타운)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(홍콩)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(하이데라바드)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(자카르타)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(멜버른)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(뭄바이)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(오사카)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(서울)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(싱가포르)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
아시아 태평양(시드니)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
아시아 태평양(도쿄)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
캐나다(중부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
캐나다 서부(캘거리)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중국(베이징)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중국(닝샤)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(프랑크푸르트)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(아일랜드)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(런던)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(밀라노)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
유럽(파리)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(스페인)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(스톡홀름)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
유럽(취리히)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
이스라엘 (텔아비브)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중동(바레인)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
중동(UAE)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
남아메리카 (상파울루)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.4 이상	버전 12.8 이상	버전 11.9 및 버전 11.13 이상
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음

Secrets Manager 통합을 지원하는 리전 및 Aurora DB 엔진

AWS Secrets Manager을 이용하면 데이터베이스 암호를 포함한 하드 코딩된 보안 인증 정보를 Secrets Manager에서 프로그래밍 방식으로 보안 암호를 검색하게 하는 API 호출로 바꿀 수 있습니다. Secrets Manager 사용에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서를 참조하세요.

Amazon Aurora가 Aurora DB 클러스터의 Secrets Manager에서 마스터 사용자 암호를 관리하도록 지정할 수 있습니다. Aurora는 암호를 생성하여 Secrets Manager에 저장한 다음 정기적으로 교체합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 단원을 참조하십시오.

Secrets Manager 통합은 모든 AWS 리전에서 사용할 수 있습니다.

Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진

Aurora Serverless v2은 Amazon Aurora에서 간헐적이거나 예측할 수 없는 워크로드를 실행하는 비용 효율적인 접근 방식으로 설계된 주문형 자동 확장 기능입니다. 애플리케이션 요구 사항을 기반으로 자동으로 용량을 확장 또는 축소합니다. Aurora Serverless v1을 사용하면 스케일링이 보다 빠르고 상세합니다. Aurora Serverless v2를 사용하면 각 클러스터에는 라이터 DB 인스턴스와 여러 리더 DB 인스턴스가 포함될 수 있습니다. Aurora Serverless v2와 기존 프로비저닝된 DB 인스턴스를 동일한 클러스터 내에 결합할 수 있습니다. 자세한 내용은 Aurora Serverless v2 사용하기 섹션을 참조하세요.

주제

- Aurora MySQL을 사용하는 Aurora Serverless v2
- Aurora PostgreSQL을 사용하는 Aurora Serverless v2

Aurora MySQL을 사용하는 Aurora Serverless v2

다음 리전 및 엔진 버전을 Aurora MySQL을 사용한 Aurora Serverless v2에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3
미국 동부(버지니아 북부)	버전 3.02.0 이상
미국 동부(오하이오)	버전 3.02.0 이상
미국 서부(캘리포니아 북부)	버전 3.02.0 이상
미국 서부(오리건)	버전 3.02.0 이상
아프리카(케이프타운)	버전 3.02.0 이상
아시아 태평양(홍콩)	버전 3.02.0 이상
아시아 태평양(하이데라바드)	버전 3.02.3 이상
아시아 태평양(자카르타)	버전 3.02.0 이상
아시아 태평양(말레이시아)	버전 3.04.3, 3.05.2, 3.06.1, 3.07.1 이상
아시아 태평양(멜버른)	버전 3.02.3 이상
아시아 태평양(뭄바이)	버전 3.02.0 이상
아시아 태평양(오사카)	버전 3.02.0 이상
아시아 태평양(서울)	버전 3.02.0 이상
아시아 태평양(싱가포르)	버전 3.02.0 이상
아시아 태평양(시드니)	버전 3.02.0 이상
아시아 태평양(도쿄)	버전 3.02.0 이상
캐나다(중부)	버전 3.02.0 이상
캐나다 서부(캘거리)	버전 3.04.0 이상
중국(베이징)	버전 3.02.2 이상
중국(닝샤)	버전 3.02.2 이상

리전	Aurora MySQL 버전 3
유럽(프랑크푸르트)	버전 3.02.0 이상
유럽(아일랜드)	버전 3.02.0 이상
유럽(런던)	버전 3.02.0 이상
유럽(밀라노)	버전 3.02.0 이상
유럽(파리)	버전 3.02.0 이상
유럽(스페인)	버전 3.02.3 이상
유럽(스톡홀름)	버전 3.02.0 이상
유럽(취리히)	버전 3.02.3 이상
이스라엘(텔아비브)	버전 3.02.3 이상, 3.03.1 이상
중동(바레인)	버전 3.02.0 이상
중동(UAE)	버전 3.02.3 이상
남아메리카(상파울루)	버전 3.02.0 이상
AWS GovCloud(미국 동부)	버전 3.02.2 이상
AWS GovCloud(미국 서부)	버전 3.02.2 이상

Aurora Serverless v2 용량 상한 및 하한 ACU 한도는 엔진 버전에 따라 다를 수 있습니다. 자세한 내용은 Aurora Serverless v2 용량을 참조하세요.

Aurora PostgreSQL을 사용하는 Aurora Serverless v2

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 Aurora Serverless v2에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
미국 동부(버지니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
미국 동부(오하이오)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
미국 서부(캘리포니아 북부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
미국 서부(오리건)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아프리카(케이프타운)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(홍콩)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(하이데라바드)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상
아시아 태평양(자카르타)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(말레이시아)	버전 16.1 이상	버전 15.4 이상	버전 14.6, 14.9 이상	버전 13.9, 13.12 이상
아시아 태평양(멜버른)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상
아시아 태평양(뭄바이)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(오사카)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
아시아 태평양(서울)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(싱가포르)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(시드니)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
아시아 태평양(도쿄)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
캐나다(중부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
캐나다 서부(캘거리)	버전 16.1 이상	버전 15.3 이상	버전 14.6, 14.8 이상	버전 13.9, 13.11 이상
중국(베이징)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
중국(닝샤)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(프랑크푸르트)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(아일랜드)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(런던)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(밀라노)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(파리)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(스페인)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상
유럽(스톡홀름)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
유럽(취리히)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
이스라엘(텔아비브)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상
중동(바레인)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
중동(UAE)	버전 16.1 이상	버전 15.2 이상	버전 14.6 이상	버전 13.9 이상
남아메리카(상파울루)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
AWS GovCloud(미국 동부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상
AWS GovCloud(미국 서부)	버전 16.1 이상	버전 15.2 이상	버전 14.3 이상	버전 13.6 이상

Aurora Serverless v2 용량 상한 및 하한 ACU 한도는 엔진 버전에 따라 다를 수 있습니다. 자세한 내용은 Aurora Serverless v2 용량 섹션을 참조하십시오.

Aurora Serverless v1

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 이 날짜 전에 Aurora Serverless v1 DB 클러스터를 Aurora Serverless v2로 업그레이드하는 것이 좋습니다. 업그레이드로 인해 데이터베이스 엔진의 메이저 버전 번호가 변경될 수 있습니다. 따라서 수명 종료일 전에 이 전환을 계획, 테스트 및 구현하는 것이 중요합니다. 2025년 1월 8일부터는 더 이상 AWS Management Console 또는 CLI를 사용하여 새 Aurora Serverless v1 클러스터 또는 인스턴스를 만들 수 없습니다. 마이그레이션 프로세스에 대한 자세한 내용은 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드 섹션을 참조하세요.
> Aurora Serverless v2는 더 빠르고 세분화된 방식으로 크기를 조정합니다. 또한 Aurora Serverless v2는 리더 DB 인스턴스와 같은 다른 Aurora 기능과의 호환성이 더 뛰어납니다.
> Aurora Serverless v2 사용하기의 Aurora Serverless v2에서 알아볼 수 있습니다.

Aurora Serverless v1은 Amazon Aurora에서 간헐적이거나 예측할 수 없는 워크로드를 실행하는 비용 효율적인 접근 방식으로 설계된 주문형 자동 확장 기능입니다. 각 클러스터에서 하나의 DB 인스턴스를 사용하여 애플리케이션 요구 사항을 기반으로 자동으로 시작 및 종료하고 용량을 확장 또는 축소합니다. 자세한 내용은 Amazon Aurora Serverless v1 사용 섹션을 참조하세요.

주제

- Aurora MySQL을 사용하는 Aurora Serverless v1
- Aurora PostgreSQL을 사용하는 Aurora Serverless v1

Aurora MySQL을 사용하는 Aurora Serverless v1

다음 리전 및 엔진 버전을 Aurora MySQL을 사용한 Aurora Serverless v1에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(버지니아 북부)	사용할 수 없음	버전 2.11.4
미국 동부(오하이오)	사용할 수 없음	버전 2.11.4
미국 서부(캘리포니아 북부)	사용할 수 없음	버전 2.11.4
미국 서부(오리건)	사용할 수 없음	버전 2.11.4
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음	사용할 수 없음
아시아 태평양(말레이시아)	사용할 수 없음	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음
아시아 태평양(뭄바이)	사용할 수 없음	버전 2.11.4
아시아 태평양(오사카)	사용할 수 없음	사용할 수 없음
아시아 태평양(서울)	사용할 수 없음	버전 2.11.4

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(싱가포르)	사용할 수 없음	버전 2.11.4
아시아 태평양(시드니)	사용할 수 없음	버전 2.11.4
아시아 태평양(도쿄)	사용할 수 없음	버전 2.11.4
캐나다(중부)	사용할 수 없음	버전 2.11.4
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음
중국(베이징)	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	버전 2.11.4
유럽(프랑크푸르트)	사용할 수 없음	버전 2.11.4
유럽(아일랜드)	사용할 수 없음	버전 2.11.4
유럽(런던)	사용할 수 없음	버전 2.11.4
유럽(밀라노)	사용할 수 없음	사용할 수 없음
유럽(파리)	사용할 수 없음	버전 2.11.4
유럽(스페인)	사용할 수 없음	사용할 수 없음
유럽(스톡홀름)	사용할 수 없음	사용할 수 없음
유럽(취리히)	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음
중동(바레인)	사용할 수 없음	사용할 수 없음
중동(UAE)	사용할 수 없음	사용할 수 없음
남아메리카(상파울루)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음

Aurora PostgreSQL을 사용하는 Aurora Serverless v1

다음 리전 및 엔진 버전을 Aurora PostgreSQL을 사용한 Aurora Serverless v1에 사용할 수 있습니다.

리전	Aurora PostgreSQL 13
미국 동부(버지니아 북부)	버전 13.12
미국 동부(오하이오)	버전 13.12
미국 서부(캘리포니아 북부)	버전 13.12
미국 서부(오리건)	버전 13.12
아프리카(케이프타운)	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음
아시아 태평양(말레이시아)	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음
아시아 태평양(뭄바이)	버전 13.12
아시아 태평양(오사카)	사용할 수 없음
아시아 태평양(서울)	버전 13.12
아시아 태평양(싱가포르)	버전 13.12
아시아 태평양(시드니)	버전 13.12

리전	Aurora PostgreSQL 13
아시아 태평양(도쿄)	버전 13.12
캐나다(중부)	버전 13.12
캐나다 서부(캘거리)	사용할 수 없음
중국(베이징)	사용할 수 없음
중국(닝샤)	사용할 수 없음
유럽(프랑크푸르트)	버전 13.12
유럽(아일랜드)	버전 13.12
유럽(런던)	버전 13.12
유럽(밀라노)	사용할 수 없음
유럽(파리)	버전 13.12
유럽(스페인)	사용할 수 없음
유럽(스톡홀름)	사용할 수 없음
유럽(취리히)	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음
중동(바레인)	사용할 수 없음
중동(UAE)	사용할 수 없음
남아메리카(상파울루)	사용할 수 없음
AWS GovCloud(미국 동부)	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음

RDS Data API를 지원하는 리전 및 Aurora DB 엔진

RDS Data API(데이터 API)는 Amazon Aurora DB 클러스터에 대한 웹 서비스 인터페이스를 제공합니다. 클라이언트 애플리케이션에서 데이터베이스 연결을 관리하는 대신 HTTPS 엔드포인트에 대해 SQL 명령을 실행할 수 있습니다. 자세한 내용은 Amazon RDS Data API 사용 단원을 참조하십시오.

Aurora MySQL의 경우 프로비저닝된 DB 클러스터에는 Aurora Serverless v2에 대한 Data API가 지원되지 않습니다.

주제

Aurora MySQL Serverless v1을 사용하는 Data API

다음 리전 및 엔진 버전을 Aurora MySQL Serverless v1을 사용하는 Data API에 사용할 수 있습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(오하이오)	사용할 수 없음	버전 2.11.3
미국 동부(버지니아 북부)	사용할 수 없음	버전 2.11.3
미국 서부(캘리포니아 북부)	사용할 수 없음	버전 2.11.3
미국 서부(오레곤)	사용할 수 없음	버전 2.11.3
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음
아시아 태평양(뭄바이)	사용할 수 없음	버전 2.11.3

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(오사카)	사용할 수 없음	사용할 수 없음
아시아 태평양(서울)	사용할 수 없음	버전 2.11.3
아시아 태평양(싱가포르)	사용할 수 없음	버전 2.11.3
아시아 태평양(시드니)	사용할 수 없음	버전 2.11.3
아시아 태평양(도쿄)	사용할 수 없음	버전 2.11.3
캐나다(중부)	사용할 수 없음	버전 2.11.3
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음
중국(베이징)	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	버전 2.11.3
유럽(프랑크푸르트)	사용할 수 없음	버전 2.11.3
유럽(아일랜드)	사용할 수 없음	버전 2.11.3
유럽(런던)	사용할 수 없음	버전 2.11.3
유럽(밀라노)	사용할 수 없음	사용할 수 없음
유럽(파리)	사용할 수 없음	버전 2.11.3
유럽(스페인)	사용할 수 없음	사용할 수 없음
유럽(스톡홀름)	사용할 수 없음	사용할 수 없음
유럽(취리히)	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음
중동(바레인)	사용할 수 없음	사용할 수 없음
중동(UAE)	사용할 수 없음	사용할 수 없음

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
남아메리카(상파울루)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음

Aurora PostgreSQL Serverless v2를 사용하고 프로비저닝된 Data API

다음 리전 및 엔진 버전을 Aurora PostgreSQL Serverless v2를 사용하고 프로비저닝된 Data API에 사용할 수 있습니다.

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
미국 동부(오하이오)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
미국 동부(버지니아 북부)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
미국 서부(캘리포니아 북부)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
미국 서부(오레곤)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(뭄바이)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
아시아 태평양(오사카)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
아시아 태평양(서울)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
아시아 태평양(싱가포르)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
아시아 태평양(시드니)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
아시아 태평양(도쿄)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
캐나다(중부)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
캐나다 서부(캘거리)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중국(베이징)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(프랑크푸르트)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
유럽(아일랜드)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
유럽(런던)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상

리전	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
유럽(밀라노)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(파리)	버전 16.1 이상	버전 15.3 이상	버전 14.8 이상	버전 13.11 이상
유럽(스페인)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(스톡홀름)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
유럽(취리히)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(바레인)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
중동(UAE)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
남아메리카(상파울루)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음	사용할 수 없음	사용할 수 없음

Aurora PostgreSQL Serverless v1을 사용하는 Data API

다음 리전 및 엔진 버전을 Aurora PostgreSQL Serverless v1을 사용하는 Data API에 사용할 수 있습니다.

리전	Aurora PostgreSQL 13	Aurora PostgreSQL 11
미국 동부(오하이오)	버전 13.9	버전 11.18
미국 동부(버지니아 북부)	버전 13.9	버전 11.18

리전	Aurora PostgreSQL 13	Aurora PostgreSQL 11
미국 서부(캘리포니아 북부)	버전 13.9	버전 11.18
미국 서부(오레곤)	버전 13.9	버전 11.18
아프리카(케이프타운)	사용할 수 없음	사용할 수 없음
아시아 태평양(홍콩)	사용할 수 없음	사용할 수 없음
아시아 태평양(하이데라바드)	사용할 수 없음	사용할 수 없음
아시아 태평양(자카르타)	사용할 수 없음	사용할 수 없음
아시아 태평양(멜버른)	사용할 수 없음	사용할 수 없음
아시아 태평양(뭄바이)	버전 13.9	버전 11.18
아시아 태평양(오사카)	사용할 수 없음	사용할 수 없음
아시아 태평양(서울)	버전 13.9	버전 11.18
아시아 태평양(싱가포르)	버전 13.9	버전 11.18
아시아 태평양(시드니)	버전 13.9	버전 11.18
아시아 태평양(도쿄)	버전 13.9	버전 11.18
캐나다(중부)	버전 13.9	버전 11.18
중국(베이징)	사용할 수 없음	사용할 수 없음
중국(닝샤)	사용할 수 없음	사용할 수 없음
유럽(프랑크푸르트)	버전 13.9	버전 11.18
유럽(아일랜드)	버전 13.9	버전 11.18
유럽(런던)	버전 13.9	버전 11.18
유럽(밀라노)	사용할 수 없음	사용할 수 없음

리전	Aurora PostgreSQL 13	Aurora PostgreSQL 11
유럽(파리)	버전 13.9	버전 11.18
유럽(스페인)	사용할 수 없음	사용할 수 없음
유럽(스톡홀름)	사용할 수 없음	사용할 수 없음
유럽(취리히)	사용할 수 없음	사용할 수 없음
이스라엘(텔아비브)	사용할 수 없음	사용할 수 없음
중동(바레인)	사용할 수 없음	사용할 수 없음
중동(UAE)	사용할 수 없음	사용할 수 없음
남아메리카(상파울루)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 동부)	사용할 수 없음	사용할 수 없음
AWS GovCloud(미국 서부)	사용할 수 없음	사용할 수 없음

제로 가동 중지 패치 적용(ZDP)을 지원하는 리전 및 Aurora DB 엔진

Aurora DB 클러스터에 대한 업그레이드를 수행할 때는 데이터베이스가 종료되고 업그레이드되는 동안 중단될 가능성이 있습니다. 기본적으로 데이터베이스가 사용 중인 동안 업그레이드를 시작하면 DB 클러스터에서 처리하는 모든 연결 및 트랜잭션이 손실됩니다. 업그레이드를 수행하기 위해 데이터베이스가 유휴 상태가 될 때까지 기다리려면 오랜 시간을 기다려야 할 수 있습니다.

제로 가동 중지 패치 적용(ZDP) 기능은 최선의 노력을 기반으로 Aurora 업그레이드 중에 클라이언트 연결을 유지하려고 시도합니다. ZDP가 성공적으로 완료되면 업그레이드 진행 중에 애플리케이션 세션이 유지되고 데이터베이스 엔진이 다시 시작됩니다. 데이터베이스 엔진이 다시 시작되면 몇 초에서 약 1분간 처리량이 저하될 수 있습니다.

Aurora MySQL 업그레이드에 ZDP를 사용할 수 있는 조건 및 엔진 버전에 대한 자세한 내용은 제로 가동 중지 패치 적용 기능 사용 섹션을 참조하세요.

Aurora PostgreSQL 업그레이드에 ZDP를 사용할 수 있는 조건 및 엔진 버전에 대한 자세한 내용은 마이너 릴리스 업그레이드 및 제로 가동 중지 패치 적용 섹션을 참조하세요.

Aurora 엔진 네이티브 기능을 지원하는 리전 및 DB 엔진

Aurora 데이터베이스 엔진은 Aurora만을 위한 추가 기능도 지원합니다. 일부 엔진 네이티브 기능에는 특정 Aurora DB 엔진, 버전 또는 지역에 대한 제한된 지원 또는 제한된 권한이 있을 수 있습니다.

주제

Aurora MySQL의 엔진 네이티브 기능

다음은 Aurora MySQL의 엔진 네이티브 기능입니다.

- 고급 감사
- 역추적
- 오류 삽입 쿼리
- 클러스터 내 쓰기 전달
- 병렬 쿼리

Aurora PostgreSQL의 엔진 네이티브 기능

다음은 Aurora PostgreSQL의 엔진 네이티브 기능입니다.

- Babelfish
- 오류 삽입 쿼리
- 쿼리 계획 관리

Amazon Aurora 엔드포인트 연결

Amazon Aurora는 일반적으로 단일 인스턴스 대신에 DB 인스턴스 클러스터와 관련됩니다. 각 연결은 특정 DB 인스턴스에서 처리합니다. Aurora 클러스터에 연결하면 지정한 호스트 이름과 포트가 엔드포인트라는 중간 핸들러를 가리킵니다. Aurora는 엔드포인트 메커니즘을 사용하여 이러한 연결을 추상화합니다. 따라서 일부 DB 인스턴스를 사용할 수 없을 때 모든 호스트 이름을 하드코딩하거나, 연결을 다시 라우팅하고 밸런싱하기 위해 자체 로직을 작성할 필요가 없습니다.

특정 Aurora 작업의 경우 다른 인스턴스 또는 인스턴스 그룹이 다른 역할을 수행합니다. 예를 들어 기본 인스턴스는 모든 데이터 정의 언어(DDL) 및 데이터 조작 언어(DML) 문을 처리합니다. 최대 15개의 Aurora 복제본이 읽기 전용 쿼리 트래픽을 처리합니다.

주제

- Aurora 엔드포인트 유형
- Aurora 클러스터의 엔드포인트 보기
- Aurora 엔드포인트가 고가용성으로 작동하는 방법
- Amazon Aurora용 클러스터 엔드포인트
- Amazon Aurora용 리더 엔드포인트
- Amazon Aurora용 인스턴스 엔드포인트
- Amazon Aurora용 사용자 지정 엔드포인트

Aurora 엔드포인트 유형

엔드포인트를 사용하여 사용 사례에 따라 각 연결을 해당 인스턴스 또는 인스턴스 그룹에 매핑할 수 있습니다. 예를 들어 DDL 문을 수행하려면 기본 인스턴스인 어떤 인스턴스에나 연결하면 됩니다. 쿼리를 수행하려면 리더 엔드포인트에 연결하면 되며, Aurora가 모든 Aurora 복제본 간에 연결 밸런싱을 자동으로 수행합니다. 다른 용량 또는 구성의 DB 인스턴스가 있는 클러스터의 경우, DB 인스턴스의 다른 하위 집합과 연결된 사용자 지정 엔드포인트에 연결할 수 있습니다. 진단 또는 튜닝의 경우 특정 인스턴스 엔드포인트에 연결하여 특정 DB 인스턴스에 대한 세부 정보를 검토할 수 있습니다.

엔드포인트는 호스트 주소와 포트를 포함하는 Aurora별 URL로 표시됩니다. Aurora DB 클러스터에서 제공하는 엔드포인트 유형은 다음과 같습니다.

클러스터 엔드포인트

> 클러스터의 기본 인스턴스에 연결하여 애플리케이션을 개발 및 테스트하고 `INSERT` 문, DDL, DML 및 ETL 작업과 같은 변환을 수행합니다. Aurora 클러스터의 엔드포인트 보기에 설명된 대로 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 클러스터 엔드포인트 위치를 찾습니다.
>
> 클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 클러스터 엔드포인트 섹션을 참조하세요.

리더 엔드포인트

쿼리를 수행합니다. Aurora는 모든 Aurora 복제본 간에 연결 밸런싱을 자동으로 수행합니다. Aurora 클러스터의 엔드포인트 보기에 설명된 대로 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 리더 엔드포인트 위치를 찾습니다.

리더 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 리더 엔드포인트 섹션을 참조하세요.

인스턴스 엔드포인트

진단 또는 조정을 위해 특정 DB 인스턴스에 대한 세부 정보를 검토하세요. AWS Management Console의 각 인스턴스에 대한 인스턴스 엔드포인트 위치는 인스턴스의 인스턴스 세부 정보 페이지에서만 찾아볼 수 있습니다.

인스턴스 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 인스턴스 엔드포인트 섹션을 참조하세요.

사용자 지정 엔드포인트

DB 클러스터에 있는 DB 인스턴스의 여러 하위 집합에 연결합니다. 이는 DB 클러스터 내에서 인스턴스 용량과 구성이 서로 다른 경우에 유용합니다. Aurora 클러스터의 엔드포인트 보기에 설명된 대로 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 사용자 지정 엔드포인트 위치를 찾습니다.

사용자 지정 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 사용자 지정 엔드포인트 섹션을 참조하세요.

Aurora Global Database 라이터 엔드포인트

Aurora Global Database에는 독립 실행형 Aurora 클러스터의 클러스터 엔드포인트와 동일한 목적을 수행하는 특별한 종류의 엔드포인트가 있습니다. 쓰기 및 읽기 요청을 모두 처리합니다. 전환 또는 장애 조치로 인해 보조 클러스터가 새 기본 클러스터가 되면 Aurora는 다른 AWS 리전에서 이 엔드포인트를 새 프라이머리 클러스터의 클러스터 엔드포인트를 가리키도록 자동으로 전환합니다. 이렇게 하면 AWS 리전을 애플리케이션의 연결 문자열로 인코딩할 필요가 없으며 글로벌 데이터베이스의 레이아웃이 변경될 때 연결 문자열을 변경할 필요가 없습니다. Aurora는 예를 들어 AWS Management Console에서 Aurora 클러스터에 대해 리전 추가를 선택하여 Aurora Global Database를 설정할 때 이 엔드포인트를 생성합니다.

Aurora Global Database에서 이러한 유형의 엔드포인트를 사용하는 방법에 대한 자세한 내용은 Amazon Aurora Global Database에 연결 섹션을 참조하세요.

Aurora 클러스터의 엔드포인트 보기

AWS Management Console의 인스턴스 세부 정보 페이지에서만 인스턴스 엔드포인트 위치를 찾아 볼 수 있지만, 클러스터, 리더 및 사용자 지정 엔드포인트의 위치는 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 확인할 수 있습니다.

Console

AWS Management Console에서 클러스터의 인스턴스 세부 정보 페이지에서 클러스터 엔드포인트, 리더 엔드포인트 및 사용자 지정 엔드포인트를 찾습니다. 각 인스턴스의 세부 정보 페이지에서 인스턴스 엔드포인트를 봅니다. 연결할 때 이 세부 정보 페이지에 표시되는 엔드포인트 이름에 콜론과 연결된 포트 번호를 추가합니다.

AWS CLI

AWS CLI을 사용하여 describe-db-clusters 명령의 출력에서 라이터, 리더 및 사용자 지정 엔드포인트를 찾습니다. 예를 들어, 다음 명령은 현재 AWS 리전의 모든 클러스터에 대한 엔드포인트 속성을 보여줍니다.

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API

Amazon RDS API로 DescribeDBClusterEndpoints 작업을 직접 호출하여 엔드포인트를 검색합니다.

Aurora 엔드포인트가 고가용성으로 작동하는 방법

고가용성이 중요한 클러스터의 경우 클러스터 엔드포인트를 읽기/쓰기 또는 범용 연결에 사용하고 리더 엔드포인트를 읽기 전용 연결에 사용합니다. 라이터 및 리더 엔드포인트는 인스턴스 엔드포인트보다 DB 인스턴스 장애 조치를 더 잘 관리합니다. 인스턴스 엔드포인트와 달리, 라이터 및 리더 엔드포인트는 클러스터의 DB 인스턴스를 사용할 수 없게 되면 연결된 DB 인스턴스를 자동으로 변경합니다. 클러스터 및 리더 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 클러스터 엔드포인트 및 Amazon Aurora용 리더 엔드포인트 섹션을 참조하세요.

DB 클러스터의 기본 DB 인스턴스에 장애가 발생하면 Aurora가 자동으로 새로운 기본 DB 인스턴스로 장애 조치합니다. 이때는 기존 Aurora 복제본을 새로운 기본 DB 인스턴스로 승격시키거나, 기본 DB 인스턴스를 새롭게 생성하는 방법이 있습니다. 장애 조치가 발생하면 클러스터 엔드포인트를 사용하여 새롭게 승격 또는 생성된 기본 DB 인스턴스에 다시 연결하거나, 리더 엔드포인트를 사용하여 DB 클러

스터에서 Aurora 복제본 중 하나에 다시 연결할 수 있습니다. 장애 조치 중 리더 엔드포인트는 Aurora 복제본이 기본 DB 인스턴스로 새롭게 승격된 후 짧은 시간 동안 DB 클러스터의 새로운 기본 DB 인스턴스에 직접 연결할 수 있습니다.

인스턴스 엔드포인트 연결을 관리하는 자체 애플리케이션 로직을 설계하는 경우, DB 클러스터에서 사용 가능한 DB 인스턴스의 결과 집합을 수동으로 또는 프로그래밍 방식으로 가져올 수 있습니다. describe-db-clusters AWS CLI 명령 또는 DescribeDBClusters RDS API 작업을 사용하여 DB 클러스터 및 리더 엔드포인트, DB 인스턴스, DB 인스턴스가 리더인지 여부, 승격 계층을 찾습니다. 그런 다음 장애 조치 이후 인스턴스 클래스를 확인하고 해당 인스턴스 엔드포인트에 연결하면 됩니다.

장애 조치에 대한 자세한 내용은 Aurora DB 클러스터의 내결함성 단원을 참조하십시오.

Amazon Aurora의 고가용성에 대한 자세한 내용은 Amazon Aurora의 고가용성 섹션을 참조하세요.

Amazon Aurora용 클러스터 엔드포인트

Aurora DB 클러스터의 클러스터 엔드포인트(또는 라이터 엔드포인트)는 해당 DB 클러스터의 현재 기본 DB 인스턴스에 연결됩니다. 이 엔드포인트는 DDL 문 등의 쓰기 작업을 수행할 수 있는 유일한 엔드포인트입니다. 이 때문에 클러스터 엔드포인트는 클러스터를 처음 설정하거나 클러스터에 단일 DB 인스턴스만 포함된 경우에 연결하는 엔드포인트입니다.

각 Aurora DB 클러스터에는 클러스터 엔드포인트 하나와 기본 DB 인스턴스 하나가 있습니다.

삽입, 업데이트, 삭제 및 DDL 변경을 비롯하여 DB 클러스터의 모든 쓰기 작업에 대해 클러스터 엔드포인트를 사용합니다. 또한 쿼리와 같은 읽기 작업에도 클러스터 엔드포인트를 사용할 수 있습니다.

이러한 클러스터 엔드포인트는 DB 클러스터에 대한 읽기/쓰기 연결 시 장애 조치를 지원합니다. DB 클러스터의 현재 기본 DB 인스턴스에 장애가 발생하면 Aurora가 자동으로 새로운 기본 DB 인스턴스로 장애 조치합니다. 장애 조치가 이루어지는 동안에도 DB 클러스터가 새로운 기본 DB 인스턴스의 클러스터 엔드포인트 연결 요청을 처리하여 서비스 중단 시간을 최소화합니다.

다음은 Aurora MySQL DB 클러스터의 클러스터 엔드포인트를 나타낸 예제입니다.

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

각 Aurora 클러스터에는 기본 제공되는 단일 클러스터 엔드포인트가 있으며, Aurora에서 이 엔드포인트의 이름과 기타 속성을 관리합니다. 이 종류의 엔드포인트는 생성, 삭제 또는 수정할 수 없습니다.

클러스터를 관리하거나, 추출, 변환, 로드(ETL) 작업을 수행하거나, 애플리케이션을 개발 및 테스트할 때 클러스터 엔드포인트를 사용합니다. 클러스터 엔드포인트는 클러스터의 기본 인스턴스에 연결됩니

다. 기본 인스턴스는 테이블과 인덱스를 만들고, INSERT 문을 실행하며, 기타 DDL 및 DML 작업을 수행할 수 있는 유일한 DB 인스턴스입니다.

장애 조치 메커니즘에서 새 DB 인스턴스가 클러스트의 읽기/쓰기 기본 인스턴스가 되도록 승격하면 클러스터 엔드포인트가 가리키는 물리적 IP 주소가 변경됩니다. 어떤 형식의 연결 풀링이나 기타 멀티플렉싱을 사용하는 경우, 캐싱된 DNS 정보의 TTL(Time to Live)을 플러시하거나 줄이도록 준비합니다. 이렇게 하면 사용할 수 없게 되었거나 장애 조치 후 이제 읽기 전용인 DB 인스턴스에 읽기/쓰기 연결 설정을 시도하지 않아도 됩니다.

Amazon Aurora용 리더 엔드포인트

Aurora DB 클러스터의 리더 엔드포인트는 DB 클러스터에 대한 읽기 전용 연결 시 연결 밸런싱을 지원합니다. 쿼리와 같은 읽기 작업에 리더 엔드포인트를 사용합니다. 이 엔드포인트는 읽기 전용 Aurora 복제본에서 이러한 문을 처리하여 기본 인스턴스에 대한 오버헤드를 줄입니다. 또한 클러스터가 클러스터의 Aurora 복제본 수에 비례하여 동시에 SELECT 쿼리를 처리할 수 있도록 용량을 확장할 수 있습니다. 각 Aurora DB 클러스터에는 리더 엔드포인트가 1개씩 있습니다.

클러스터에 하나 이상의 Aurora 복제본이 포함된 경우 리더 엔드포인트는 Aurora 복제본 사이의 각 연결 요청을 밸런싱합니다. 이 경우 해당 세션의 SELECT과 같은 읽기 전용 문만 실행할 수 있습니다. 클러스터에 기본 인스턴스만 있고 Aurora 복제본이 없는 경우 리더 엔드포인트는 기본 인스턴스에 연결합니다. 이 경우 엔드포인트를 통해 쓰기 작업을 수행할 수 있습니다.

다음은 Aurora MySQL DB 클러스터의 리더 엔드포인트를 나타낸 예제입니다.

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Aurora 클러스터의 읽기 전용 연결에 리더 엔드포인트를 사용합니다. 이 엔드포인트는 클러스터가 쿼리 집약적인 워크로드를 처리할 수 있도록 돕는 연결 밸런싱 메커니즘을 사용합니다. 리더 엔드포인트는 클러스터에서 보고 또는 기타 읽기 전용 작업을 수행하는 애플리케이션에 제공하는 엔드포인트입니다.

리더 엔드포인트는 Aurora DB 클러스터에서 사용 가능한 Aurora 복제본 연결에 밸런싱을 적용합니다. 개별 쿼리는 밸런싱하지 않습니다. 각 쿼리를 밸런싱하여 DB 클러스터의 읽기 워크로드를 분산하려면, 각 쿼리의 리더 엔드포인트에 대한 새 연결을 엽니다.

각 Aurora 클러스터에는 기본 제공되는 단일 리더 엔드포인트가 있으며, Aurora에서 이 엔드포인트의 이름과 기타 속성을 관리합니다. 이 종류의 엔드포인트는 생성, 삭제 또는 수정할 수 없습니다.

클러스터에 프라이머리 대상(인스턴스 또는 DB 샤드 그룹)만 있고 Aurora 복제본이 없는 경우 리더 엔드포인트는 기본 인스턴스에 연결합니다. 이 경우 이 엔드포인트를 통해 쓰기 작업을 수행할 수 있습니다.

> **Tip**
>
> RDS 프록시를 통해 Aurora클러스터용으로 추가 읽기 전용 엔드포인트를 생성할 수 있습니다. 이러한 엔드포인트는 Aurora 리더 엔드포인트와 동일한 종류의 밸런싱을 수행합니다. 리더 인스턴스를 사용할 수 없게 될 경우, Aurora 리더 엔드포인트보다 애플리케이션이 프록시 엔드포인트에 더 빠르게 다시 연결할 수 있습니다. 프록시 엔드포인트는 멀티플렉싱과 같은 다른 프록시 기능을 활용할 수도 있습니다. 자세한 내용은 Aurora 클러스터에 리더 엔드포인트 사용 섹션을 참조하세요.

Amazon Aurora용 인스턴스 엔드포인트

인스턴스 엔드포인트는 Aurora 클러스터에 있는 특정 DB 인스턴스에 연결됩니다. DB 클러스터의 DB 인스턴스에는 각각 고유한 인스턴스 엔드포인트가 있습니다. 그러므로 DB 클러스터의 현재 기본 DB 인스턴스에 대해 인스턴스 엔드포인트 하나가 있고, DB 클러스터의 각 Aurora 복제본마다 인스턴스 엔드포인트 하나가 있습니다.

클러스터 엔드포인트 또는 리더 엔드포인트의 사용이 부적합한 시나리오에서는 인스턴스 엔드포인트가 DB 클러스터에 대한 연결을 직접 제어합니다. 예를 들어 클라이언트 애플리케이션에서 워크로드 유형에 따라 더욱 세분화된 연결 밸런싱이 필요할 수 있습니다. 이 경우에는 여러 클라이언트를 구성하여 DB 클러스터에 속한 각기 다른 Aurora 복제본에 연결한 후 읽기 워크로드를 분산시킬 수 있습니다. Aurora PostgreSQL에 대한 장애 조치 후 인스턴스 엔드포인트를 사용하여 연결 속도를 높이는 예제는 Amazon Aurora PostgreSQL를 사용한 빠른 장애 조치 단원을 참조하세요. Aurora MySQL에 대한 장애 조치 후 인스턴스 엔드포인트를 사용하여 연결 속도를 높이는 예제는 MariaDB 커넥터/J 장애 조치 지원- Amazon Aurora 사례를 참조하세요.

다음은 Aurora MySQL DB 클러스터의 DB 인스턴스 엔드포인트를 나타낸 예제입니다.

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Aurora 클러스터의 각 DB 인스턴스에는 기본 제공되는 자체 인스턴스 엔드포인트가 있으며, Aurora에서 이 엔드포인트의 이름과 기타 속성을 관리합니다. 이 종류의 엔드포인트는 생성, 삭제 또는 수정할 수 없습니다. Amazon RDS를 사용하는 경우 인스턴스 엔드포인트에 익숙해질 수 있습니다. 그러나

Aurora에서는 일반적으로 라이터 및 리더 엔드포인트를 인스턴스 엔드포인트보다 더 자주 사용합니다.

일상적인 Aurora 작업에서 인스턴스 엔드포인트를 사용하는 주요 방법은 Aurora 클러스터의 특정 인스턴스 하나에 영향을 주는 용량 또는 성능 문제를 진단하는 것입니다. 특정 인스턴스에 연결되어 있는 동안 해당 인스턴스의 상태 변수, 지표 등을 검토할 수 있습니다. 이렇게 하면 클러스터의 다른 인스턴스에 일어난 일과 별개로 해당 인스턴스에 일어난 일을 확인하는 데 도움이 됩니다.

고급 사용 사례에서는 일부 DB 인스턴스를 다른 인스턴스와 다르게 구성할 수 있습니다. 이 경우 인스턴스 엔드포인트를 사용하여 더 작거나, 더 크거나, 다른 인스턴스와 특성이 다른 인스턴스에 직접 연결합니다. 또한 이 특별한 DB 인스턴스가 기본 인스턴스로 대체되는 마지막 인스턴스가 되도록 장애 조치 우선 순위를 설정합니다. 그러한 경우 인스턴스 엔드포인트 대신에 사용자 지정 엔드포인트를 사용하는 것이 좋습니다. 그렇게 하면 클러스터에 DB 인스턴스를 더 추가할 때 연결 관리와 고가용성이 간소화됩니다.

Amazon Aurora용 사용자 지정 엔드포인트

Aurora 클러스터의 사용자 지정 엔드포인트는 선택한 DB 인스턴스 집합을 나타냅니다. 엔드포인트에 연결하면 Aurora가 연결 밸런싱을 수행하고 그룹에서 연결을 처리할 인스턴스 중 하나를 선택합니다. 이 엔드포인트가 참조하는 인스턴스를 정의하고, 이 엔드포인트가 어떤 목적으로 사용되는지 결정합니다.

사용자 지정 엔드포인트를 만들기 전까지 Aurora DB 클러스터에는 사용자 지정 엔드포인트가 없습니다. 프로비저닝된 각 Aurora 클러스터 또는 Aurora Serverless v2 클러스터에 대해 최대 다섯 개의 사용자 지정 엔드포인트를 만들 수 있습니다. Aurora Serverless v1 클러스터에는 사용자 지정 엔드포인트를 사용할 수 없습니다.

사용자 지정 엔드포인트는 DB 인스턴스의 읽기 전용 기능 또는 읽기/쓰기 기능 이외의 다른 조건을 기반으로 밸런싱된 데이터베이스 연결을 제공합니다. 예를 들어 특정 AWS 인스턴스 클래스 또는 특정 DB 파라미터 그룹을 사용하는 인스턴스에 연결할 사용자 지정 엔드포인트를 정의할 수 있습니다. 그런 다음 특정 사용자 그룹에 이 사용자 지정 엔드포인트에 대해 알릴 수 있습니다. 예를 들어 보고 생성 또는 임시(일회) 쿼리를 위해 저용량 인스턴스로 내부 사용자를 보내고, 고용량 인스턴스로 프로덕션 트래픽을 보낼 수 있습니다.

사용자 지정 엔드포인트와 연결된 모든 DB 인스턴스로 연결이 이동할 수 있기 때문에, 해당 그룹 내의 모든 DB 인스턴스가 일부 유사한 특성을 공유하는지 확인하는 것이 좋습니다. 그렇게 하면 성능, 메모리 용량 등이 해당 엔드포인트에 연결하는 모든 사람에게 일관되도록 보장할 수 있습니다.

이 기능은 모든 Aurora 복제본을 동일한 클러스터에 유지하는 것이 적절치 않은 특수한 종류의 워크로드가 있는 고급 사용자를 위한 것입니다. 사용자 지정 엔드포인트를 통해 각 연결에 사용되는 DB 인스

턴스 용량을 예측할 수 있습니다. 사용자 지정 엔드포인트를 사용할 경우 일반적으로 해당 클러스터에 리더 엔드포인트를 사용하지 않습니다.

다음은 Aurora MySQL DB 클러스터에 있는 DB 인스턴스의 사용자 지정 엔드포인트를 나타낸 예제입니다.

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

클러스터에 용량 및 구성 설정이 서로 다른 DB 인스턴스가 포함된 경우 사용자 지정 엔드포인트를 사용하여 연결 관리를 간소화합니다.

이전에 자체 도메인에서 DNS(Domain Name Service) 별칭을 설정하는 CNAME 메커니즘을 사용하여 비슷한 결과를 달성했을 수도 있습니다. 사용자 지정 엔드포인트를 사용하면 클러스터가 커지거나 줄어들 때 CNAME 레코드를 업데이트하지 않아도 됩니다. 또한 사용자 지정 엔드포인트는 암호화된 전송 계층 보안/Secure Sockets Layer(TLS/SSL) 연결을 사용할 수 있음을 뜻합니다.

각각의 특수 목적에 DB 인스턴스를 하나씩 사용하고 인스턴스 엔드포인트에 연결하는 대신에, 특수 DB 인스턴스 그룹을 여러 개 사용할 수 있습니다. 이 경우 각 그룹에 자체 사용자 지정 엔드포인트가 있습니다. 이러한 방식으로 Aurora는 프로덕션 또는 내부 쿼리 보고나 처리 등의 작업 전용 인스턴스 간에 연결 밸런싱을 수행할 수 있습니다. 사용자 지정 엔드포인트는 DNS를 사용하여 인스턴스 중 하나의 IP 주소를 무작위로 반환하는 방식으로 인스턴스 간에 연결을 수동적으로 배포합니다. 그룹 내의 DB 인스턴스 중 하나를 사용할 수 없게 되면, Aurora는 동일한 엔드포인트와 연결된 다른 DB 인스턴스 중 하나로 후속 사용자 지정 엔드포인트 연결을 보냅니다.

주제

- Amazon Aurora의 사용자 지정 엔드포인트에 대한 고려 사항
- 사용자 지정 엔드포인트 만들기
- 사용자 지정 엔드포인트 보기
- 사용자 지정 엔드포인트 편집
- 사용자 지정 엔드포인트 삭제
- Amazon Aurora용 사용자 지정 엔드포인트의 AWS CLI 예제

Amazon Aurora의 사용자 지정 엔드포인트에 대한 고려 사항

다음 섹션을 사용하여 속성을 관리 및 지정하고, 사용자 지정 엔드포인트에 대한 멤버십 규칙을 사용합니다.

주제

- 사용자 지정 엔드포인트 관리
- 사용자 지정 엔드포인트의 속성 지정
- 사용자 지정 엔드포인트의 멤버십 규칙

사용자 지정 엔드포인트 관리

새로 생성된 Aurora 클러스터에는 사용자 지정 엔드포인트가 없기 때문에, 이러한 객체를 직접 만들고 관리해야 합니다. AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 관리할 수 있습니다.

> **Note**
>
> 또한 스냅샷에서 복원된 Aurora 클러스터의 사용자 지정 엔드포인트도 만들고 관리해야 합니다. 사용자 지정 엔드포인트는 스냅샷에 포함되지 않습니다. 복원 후 사용자 지정 엔드포인트를 다시 만들고, 복원된 클러스터가 원래 리전과 동일한 리전에 있는 경우 새 엔드포인트 이름을 선택합니다.

AWS Management Console에서 사용자 지정 엔드포인트를 작업하려면 Aurora 클러스터의 세부 정보 페이지로 이동하고 사용자 지정 엔드포인트 섹션 아래의 컨트롤을 사용합니다.

AWS CLI에서 사용자 지정 엔드포인트를 작업하려면 다음 작업을 사용하면 됩니다.

- create-db-cluster-endpoint
- describe-db-cluster-endpoints
- modify-db-cluster-endpoint
- delete-db-cluster-endpoint

Amazon RDS API를 통해 사용자 지정 엔드포인트를 작업하려면 다음 함수를 사용하면 됩니다.

- CreateDBClusterEndpoint
- DescribeDBClusterEndpoints
- ModifyDBClusterEndpoint
- DeleteDBClusterEndpoint

사용자 지정 엔드포인트의 속성 지정

사용자 지정 엔드포인트 이름의 최대 길이는 63자입니다. 이름은 다음 형식을 따릅니다.

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

동일한 AWS 리전에 있는 둘 이상의 클러스터에 동일한 사용자 지정 엔드포인트 이름을 재사용할 수 없습니다. 사용자 지정 DNS 식별자는 특정 AWS 리전 내에서 AWS 계정과 관련된 고유한 식별자입니다.

각 사용자 지정 엔드포인트의 연결 유형은 해당 엔드포인트와 연결할 수 있는 DB 인스턴스를 결정합니다. 현재 이 유형은 `READER` 또는 `ANY`일 수 있습니다. 사용자 지정 엔드포인트 유형에는 다음 고려 사항이 적용됩니다.

- AWS Management Console에서는 사용자 지정 엔드포인트는 선택할 수 없습니다. AWS Management Console을 통해 만드는 모든 사용자 지정 엔드포인트는 `ANY` 유형입니다.

  AWS CLI 또는 Amazon RDS API를 사용하여 사용자 지정 엔드포인트 유형을 설정하고 수정할 수 있습니다.
- 리더 DB 인스턴스만 `READER` 사용자 지정 엔드포인트의 일부일 수 있습니다.
- 리더 및 라이터 DB 인스턴스가 `ANY` 사용자 지정 엔드포인트의 일부일 수 있습니다. Aurora는 `ANY` 유형이 있는 클러스터 엔드포인트에 동일한 확률을 가진 연관된 DB 인스턴스로 연결합니다. `ANY` 유형은 복제 토폴로지를 사용하는 클러스터에 적용됩니다.
- 클러스터의 복제 구성을 기반으로 적합하지 않은 유형의 사용자 지정 엔드포인트를 만들려고 하면 Aurora가 오류를 반환합니다.

사용자 지정 엔드포인트의 멤버십 규칙

사용자 지정 엔드포인트에 DB 인스턴스를 추가하거나 사용자 지정 엔드포인트에서 제거해도, 해당 DB 인스턴스로의 기존 연결은 활성 상태로 유지됩니다.

사용자 지정 엔드포인트에 포함시키거나 제외시킬 DB 인스턴스 목록을 정의할 수 있습니다. 이러한 목록을 각각 정적 및 제외 목록이라고 합니다. 포함/제외 메커니즘을 사용하여 DB 인스턴스 그룹을 더 세분화하고, 사용자 지정 엔드포인트 집합이 클러스터의 모든 DB 인스턴스를 포함하도록 할 수 있습니다. 각 사용자 지정 엔드포인트는 이러한 목록 유형 중 하나만 포함할 수 있습니다.

AWS Management Console:

- Attach future instances added to this cluster(이 클러스터에 추가된 향후 인스턴스 첨부) 확인란에 선택이 표시됩니다. 이 확인란을 선택하지 않으면 사용자 지정 엔드포인트가 페이지에 지정된 DB 인스턴스만 포함하는 정적 목록을 사용합니다. 이 확인란을 선택하면 사용자 지정 엔드포인트가 제외 목록을 사용합니다. 이 경우 사용자 지정 엔드포인트는 페이지에서 선택되지 않은 인스턴스를 제외한 클러스터의 모든 DB 인스턴스를 나타냅니다(향후 추가하는 모든 인스턴스 포함).
- 콘솔에서는 엔드포인트 유형을 지정할 수 없습니다. 콘솔을 사용하여 만든 모든 사용자 지정 엔드포인트는 ANY 유형입니다.

  따라서 장애 조치 또는 승격으로 인해 DB 인스턴스가 라이터와 리더 간에 역할을 변경할 때 Aurora는 사용자 지정 엔드포인트의 구성원 자격을 변경하지 않습니다.

AWS CLI 및 Amazon RDS API:

- 엔드포인트 유형을 지정할 수 있습니다. 따라서 엔드포인트 유형을 READER로 설정하면 장애 조치 및 승격 중에 엔드포인트 멤버십이 자동으로 조정됩니다.

  예를 들어 유형이 READER인 사용자 지정 엔드포인트에는 라이터 DB 인스턴스로 승격되는 Aurora 복제본이 포함됩니다. 새 라이터 인스턴스는 더 이상 사용자 지정 엔드포인트의 일부가 아닙니다.
- 역할을 변경한 후 개별 구성원을 목록에 추가하거나 목록에서 제거할 수 있습니다. modify-db-cluster-endpoint AWS CLI 명령 또는 ModifyDBClusterEndpoint API 작업을 사용하세요.

DB 인스턴스 하나를 둘 이상의 사용자 지정 엔드포인트와 연결할 수 있습니다. 예를 들어 클러스터에 새 DB 인스턴스를 추가하거나 Aurora가 AutoScaling 메커니즘을 통해 DB 인스턴스를 자동으로 추가한다고 가정하겠습니다. 이러한 경우 적합한 모든 사용자 지정 엔드포인트에 DB 인스턴스가 추가됩니다. DB 인스턴스가 추가되는 엔드포인트는 사용자 지정 엔드포인트 유형(READER 또는 ANY)과 각 엔드포인트에 대해 정의한 정적 또는 제외 목록에 따라 다릅니다. 예를 들어 엔드포인트에 DB 인스턴스의 정적 목록이 포함된 경우, 해당 엔드포인트에는 새로 추가된 Aurora 복제본이 추가되지 않습니다. 반대로 엔드포인트에 제외 목록이 있는 경우, 새로 추가된 Aurora 복제본이 제외 목록에서 이름이 지정되지 않았고 역할이 사용자 지정 엔드포인트 유형과 일치하면 엔드포인트에 추가됩니다.

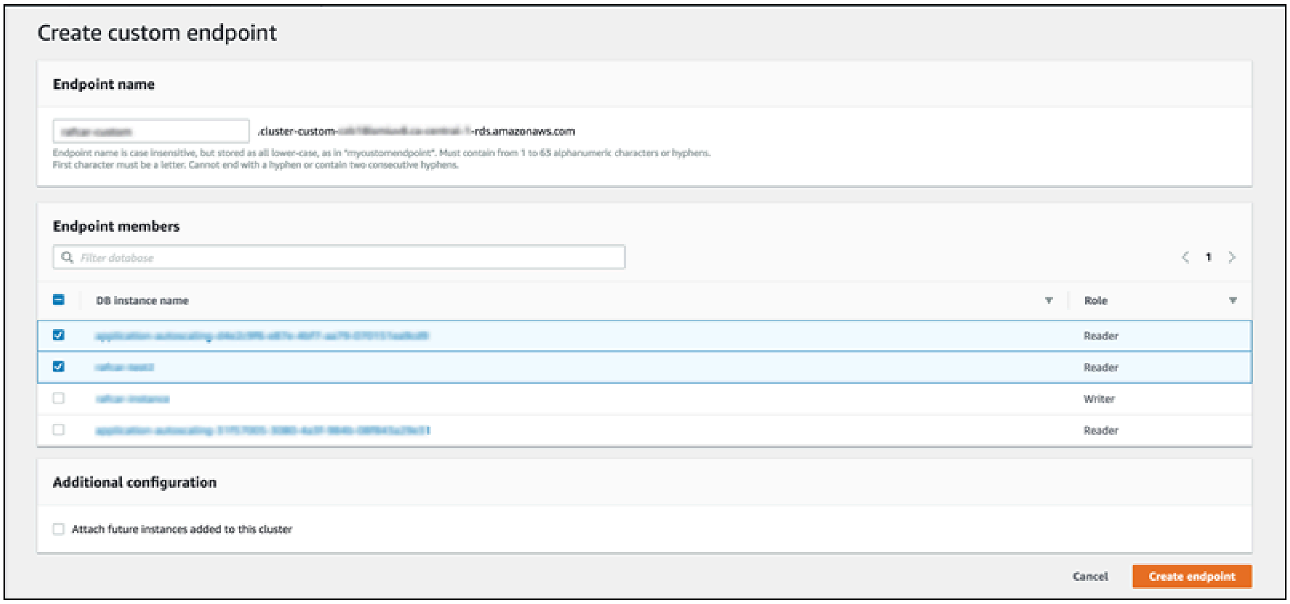
Aurora 복제본을 사용할 수 없게 된 경우, 사용자 지정 엔드포인트와 연결된 상태로 유지됩니다. 예를 들어 이상이 있거나, 중지되었거나, 재부팅되더라도 사용자 지정 엔드포인트의 일부로 유지됩니다. 그러나 다시 사용할 수 있게 될 때까지 그러한 엔드포인트를 통해 연결할 수 없습니다.

사용자 지정 엔드포인트 만들기

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 사용자 지정 엔드포인트를 생성합니다.

콘솔

AWS Management Console을 사용하여 사용자 지정 엔드포인트를 만들려면 클러스터 세부 정보 페이지로 이동하고 엔드포인트 섹션에서 `Create custom endpoint` 작업을 선택합니다. 사용자 ID와 리전에 고유한 사용자 지정 엔드포인트의 이름을 선택합니다. 클러스터가 확장되더라도 동일하게 유지되는 DB 인스턴스의 목록을 선택하려면 Attach future instances added to this cluster(이 클러스터에 추가된 향후 인스턴스 첨부) 확인란을 선택하지 않습니다. 이 확인란을 선택한 경우 클러스터에 새 인스턴스를 추가하면 사용자 지정 엔드포인트가 이러한 새 인스턴스를 동적으로 추가합니다.



AWS Management Console에서는 사용자 지정 엔드포인트는 선택할 수 없습니다. AWS Management Console을 통해 만드는 모든 사용자 지정 엔드포인트는 ANY 유형입니다.

AWS CLI

AWS CLI를 사용하여 사용자 지정 엔드포인트를 만들려면 create-db-cluster-endpoint 명령을 실행합니다.

다음 명령은 특정 클러스터에 연결된 사용자 지정 엔드포인트를 만듭니다. 처음에 엔드포인트는 클러스터의 모든 Aurora 복제본 인스턴스와 연결되어 있습니다. 후속 명령은 엔드포인트를 클러스터의 특정 DB 인스턴스 집합과 연결합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --endpoint-type reader \
```

```
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

Windows의 경우:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

RDS API

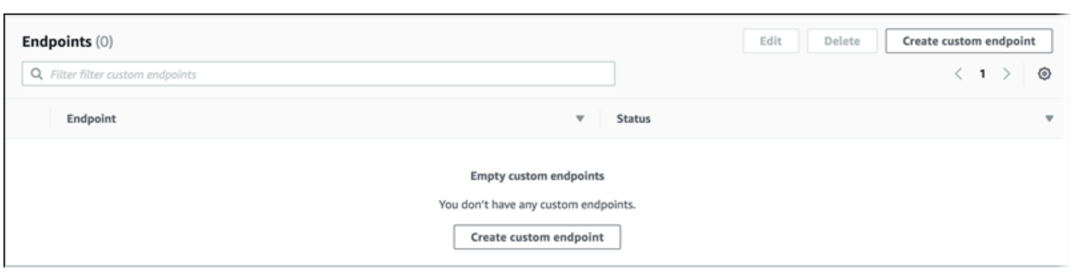
RDS API를 사용하여 사용자 지정 엔드포인트를 만들려면 CreateDBClusterEndpoint 작업을 실행합니다.

사용자 지정 엔드포인트 보기

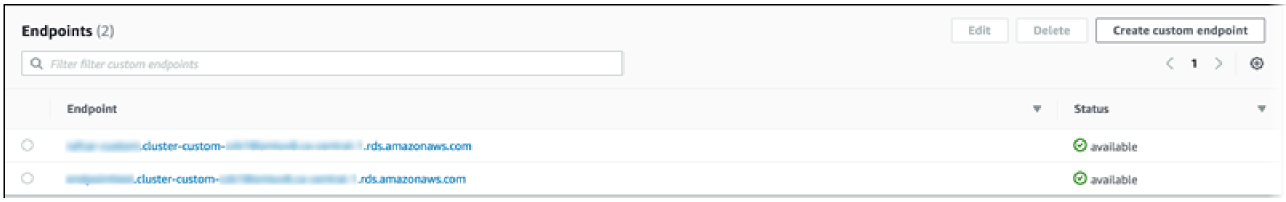
콘솔

AWS Management Console을 사용하여 사용자 지정 엔드포인트를 보려면 해당 클러스터의 클러스터 세부 정보 페이지로 이동하고 엔드포인트 섹션 아래를 봅니다. 이 섹션에는 사용자 지정 엔드포인트에 대한 정보만 포함되어 있습니다. 기본 제공 엔드포인트의 세부 정보는 기본 세부 정보 섹션에 나와 있습니다. 특정 사용자 지정 엔드포인트의 세부 정보를 보려면 해당 이름을 선택하여 해당 엔드포인트의 세부 정보 페이지를 불러옵니다.

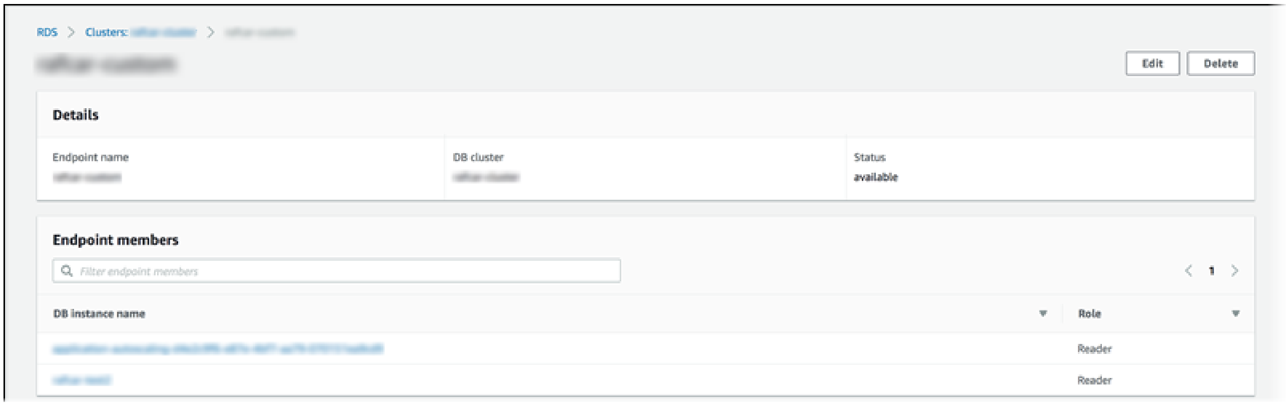
다음 스크린샷은 Aurora 클러스터의 사용자 지정 엔드포인트 목록이 처음에 비어 있음을 보여 줍니다.



해당 클러스터의 사용자 지정 엔드포인트를 만들면 엔드포인트 섹션 아래에 표시됩니다.



세부 정보 페이지까지 클릭해 가면 엔드포인트가 현재 연결된 DB 인스턴스가 표시됩니다.



클러스터에 추가된 새 DB 인스턴스가 엔드포인트에도 자동으로 추가되었는지 여부에 대한 추가 정보를 보려면 엔드포인트의 편집(Edit) 페이지를 엽니다.

AWS CLI

AWS CLI를 사용하여 사용자 지정 엔드포인트를 보려면 describe-db-cluster-endpoints 명령을 실행합니다.

다음 명령은 지정된 리전의 지정된 클러스터와 연결된 사용자 지정 엔드포인트를 보여 줍니다. 출력에는 기본 제공 엔드포인트와 사용자 지정 엔드포인트가 모두 포함됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

Windows의 경우:

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

다음은 `describe-db-cluster-endpoints` 명령의 출력 샘플을 보여 줍니다. `EndpointType` 또는 `WRITER`의 `READER`는 클러스터의 기본 제공 읽기/쓰기 및 읽기 전용 엔드포인트를 나타냅니다. `EndpointType`의 `CUSTOM`는 연결된 DB 인스턴스를 생성하고 선택한 엔드포인트를 나타냅니다.

엔드포인트 중 하나의 비어 있지 않은 `StaticMembers` 필드는 정확한 DB 인스턴스 집합과 연결되었음을 나타냅니다. 다른 엔드포인트의 비어 있지 않은 `ExcludedMembers` 필드는 엔드포인트가 `ExcludedMembers` 아래에 나열된 인스턴스 이외의 모든 DB 인스턴스와 연결되어 있음을 나타냅니다. 이 두 번째 종류의 사용자 지정 엔드포인트는 클러스터에 새 인스턴스를 추가할 때 이 새 인스턴스를 포함하도록 확장됩니다.

```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
```

```
    "CustomEndpointType": "ANY",
    "DBClusterEndpointIdentifier": "eight-and-higher",
    "ExcludedMembers": [
      "custom-endpoint-demo-04",
      "custom-endpoint-demo-02",
      "custom-endpoint-demo-07",
      "custom-endpoint-demo-05",
      "custom-endpoint-demo-03",
      "custom-endpoint-demo-06",
      "custom-endpoint-demo-01"
    ],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "available",
    "EndpointType": "CUSTOM",
    "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

RDS API

RDS API를 사용하여 사용자 지정 엔드포인트를 보려면 DescribeDBClusterEndpoints.html 작업을 실행합니다.

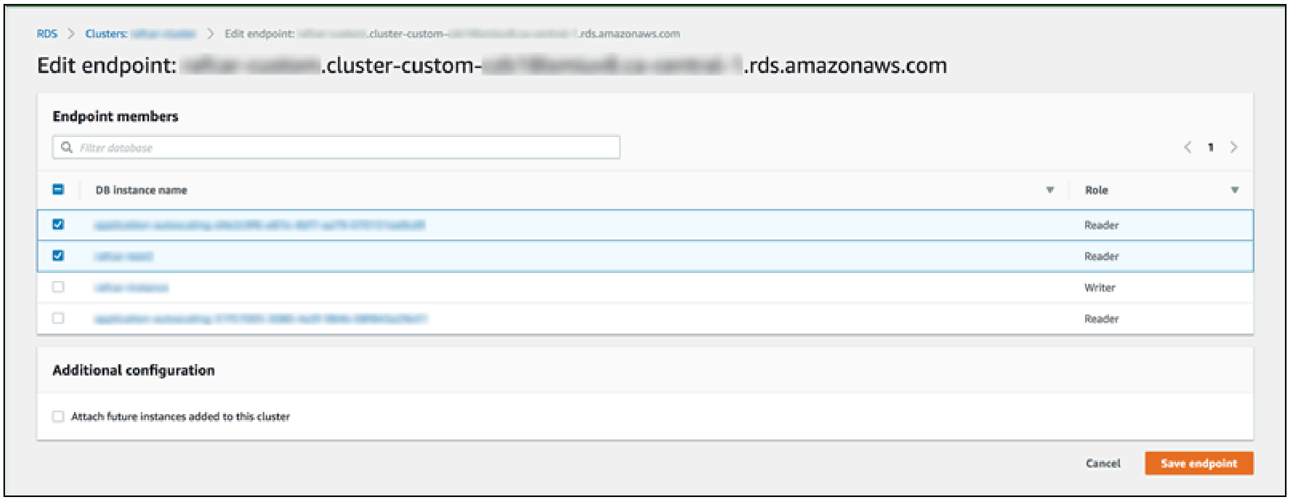
사용자 지정 엔드포인트 편집

사용자 지정 엔드포인트의 속성을 편집하여 엔드포인트와 연결된 DB 인스턴스를 변경할 수 있습니다. 또한 정적 목록과 제외 목록 간에 엔드포인트를 변경할 수도 있습니다. 이러한 엔드포인트 속성에 대한 세부 정보가 필요한 경우, 사용자 지정 엔드포인트의 멤버십 규칙 단원을 참조하세요.

편집 작업으로 인한 변경이 진행 중인 동안에는 사용자 지정 엔드포인트에 연결하거나 해당 엔드포인트를 사용할 수 없습니다.

콘솔

AWS Management Console을 사용하여 사용자 지정 엔드포인트를 편집하려면 클러스터 세부 정보 페이지에서 해당 엔드포인트를 선택하거나, 해당 엔드포인트의 세부 정보 페이지를 불러와서 편집 작업을 선택합니다.



AWS CLI

AWS CLI를 사용하여 사용자 지정 엔드포인트를 편집하려면 modify-db-cluster-endpoint 명령을 실행합니다.

다음 명령은 사용자 지정 엔드포인트에 적용되는 DB 인스턴스 집합을 변경하고 필요에 따라 정적 목록 또는 제외 목록의 동작 간에 전환합니다. `--static-members` 및 `--excluded-members` 파라미터는 공백으로 구분된 DB 인스턴스 식별자 목록을 가져옵니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

Windows의 경우:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

RDS API

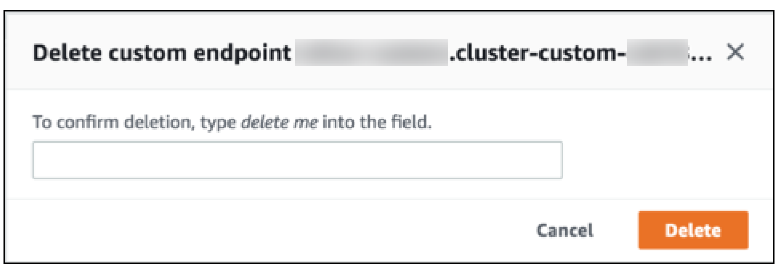
RDS API를 사용하여 사용자 지정 엔드포인트를 편집하려면 [ModifyDBClusterEndpoint.html] 작업을 실행합니다.

사용자 지정 엔드포인트 삭제

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 사용자 지정 엔드포인트를 삭제합니다.

콘솔

AWS Management Console을 사용하여 사용자 지정 엔드포인트를 삭제하려면 클러스터 세부 정보 페이지로 이동하고, 해당 사용자 지정 엔드포인트를 선택한 후, 삭제 작업을 선택합니다.



AWS CLI

AWS CLI를 사용하여 사용자 지정 엔드포인트를 삭제하려면 delete-db-cluster-endpoint 명령을 실행합니다.

다음 명령은 사용자 지정 엔드포인트를 삭제합니다. 연결된 클러스터는 지정하지 않아도 되지만, 리전은 지정해야 합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

Windows의 경우:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
  --region region_name
```

RDS API

RDS API를 사용하여 사용자 지정 엔드포인트를 삭제하려면 DeleteDBClusterEndpoint 작업을 실행합니다.

Amazon Aurora용 사용자 지정 엔드포인트의 AWS CLI 예제

다음 자습서에서는 Unix 쉘 구문과 함께 AWS CLI 예제를 사용하여 여러 '작은' DB 인스턴스와 몇 개의 '큰' DB 인스턴스가 있는 클러스터를 정의하고, 사용자 지정 엔드포인트를 만들어 각 DB 인스턴스 집합에 연결할 수 있는 방법을 보여줍니다. 자체 시스템에서 비슷한 명령을 실행하려면 리전, 서브넷 그룹, VPC 보안 그룹 등의 파라미터에 자체 값을 제공할 수 있도록 Aurora 클러스터와 AWS CLI 사용법의 기본을 충분히 숙지한 상태여야 합니다.

이 예제에서는 초기 설정 단계인 Aurora 클러스터 만들기와 클러스터에 DB 인스턴스 추가하기를 보여 줍니다. 이는 다른 유형의 클러스터로, 모든 DB 인스턴스의 용량이 같지 않음을 뜻합니다. 대부분의 인스턴스는 AWS 인스턴스 클래스 `db.r4.4xlarge`를 사용하지만, 마지막 두 DB 인스턴스는 `db.r4.16xlarge`를 사용합니다. 이러한 각 샘플 `create-db-instance` 명령은 화면에 출력을 인쇄하고 나중에 검사하기 위해 파일에 JSON 복사본을 저장합니다.

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.04.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
```

```
    --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
    --region $REGION \
    | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
    --region $REGION \
    | tee custom-endpoint-demo-${i}.json
done
```

더 큰 인스턴스는 특수한 종류의 쿼리 보고를 위해 예약되어 있습니다. 다음 예제에서는 이러한 인스턴스가 기본 인스턴스로 승격될 가능성이 없도록 해당 인스턴스의 승격 티어를 가장 낮은 우선 순위로 변경합니다. 이 예제에서는 마스터 사용자 암호를 생성하고 이를 Secrets Manager에서 관리하는 `--manage-master-user-password` 옵션을 지정합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 단원을 참조하십시오. 또는 `--master-password` 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

```
for i in 09 10
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

리소스를 가장 많이 사용하는 쿼리에만 “더 큰” 인스턴스 두 개를 사용하려고 한다고 가정하겠습니다. 이렇게 하려면 먼저 사용자 지정 읽기 전용 엔드포인트를 만든 후, 엔드포인트가 그러한 DB 인스턴스에만 연결되도록 정적 멤버 목록을 추가할 수 있습니다. 그러한 인스턴스는 이미 가장 낮은 승격 티어에 있으므로, 어느 인스턴스도 기본 인스턴스로 승격될 가능성이 낮습니다. 두 인스턴스 중 하나가 기본 인스턴스로 승격된 경우, READER 유형 대신에 ANY 유형으로 지정했으므로 이 엔드포인트를 통해 해당 인스턴스에 접근할 수 없게 됩니다.

다음 예제에서는 엔드포인트 생성 및 수정 명령과, 사용자 지정 엔드포인트의 초기 및 수정된 상태가 나와 있는 샘플 JSON 출력을 보여 줍니다.

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
```

```
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

클러스터의 기본 READER 엔드포인트는 '작거나' '큰' DB 인스턴스에 연결될 수 있으므로, 클러스터가 사용 중일 때 쿼리 성능과 확장성을 예측하는 것은 어렵습니다. DB 인스턴스 집합 간에 워크로드를 깔끔하게 나누려면 기본 READER 엔드포인트를 무시하고 다른 모든 DB 인스턴스에 연결되는 두 번째 사용자 지정 엔드포인트를 만들면 됩니다. 다음 예제에서는 사용자 지정 엔드포인트를 만든 후 제외 목록을 추가하여 이를 수행합니다. 클러스터에 나중에 추가하는 다른 모든 DB 인스턴스는 이 엔드포인

트에 자동으로 추가됩니다. ANY 유형은 이 엔드포인트가 총 여덟 개의 인스턴스(기본 인스턴스 하나와 Aurora 복제본 일곱 개)와 연결되어 있음을 뜻합니다. 이 예제에서 READER 유형을 사용한 경우, 사용자 지정 엔드포인트는 Aurora 복제본 일곱 개에만 연결됩니다.

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

다음 예제에서는 이 클러스터의 엔드포인트 상태를 확인합니다. 클러스터에 여전히 EndPointType이 WRITER인 원래 클러스터 엔드포인트가 있으며, 관리, ETL 및 기타 쓰기 작업에 여전히 이 엔드포인트를 사용합니다. 여전히 원래 READER 엔드포인트가 있는데, 이 엔드포인트로의 각 연결을 '작거나' '큰' DB 인스턴스로 보낼 수 있으므로 이 인스턴스를 사용하지 않습니다. 사용자 지정 엔드포인트는 이 동작을 예측 가능하게 합니다. 지정한 엔드포인트를 기반으로 '작거나' '큰' DB 인스턴스 중 하나를 사용하도록 연결이 보장되어 있기 때문입니다.

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
```

```
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
    ]
}
```

마지막 예제에서는 사용자 지정 엔드포인트로의 연속적인 데이터베이스 연결이 Aurora 클러스터의 다양한 DB 인스턴스에 연결되는 방법을 보여 줍니다. `small-instances` 엔드포인트는 항상 이 클러스터에서 낮은 번호의 호스트인 `db.r4.4xlarge` DB 인스턴스에 연결됩니다.

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-02 |
+-------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-07 |
+-------------------------+
```

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-01 |
+-------------------------+
```

big-instances 엔드포인트는 항상 이 클러스터에서 가장 높은 번호의 두 호스트인 db.r4.16xlarge DB 인스턴스에 연결됩니다.

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-10 |
+-------------------------+

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-09 |
+-------------------------+
```

Amazon Aurora DB 인스턴스 클래스

DB 인스턴스 클래스는 Amazon Aurora DB 인스턴스의 컴퓨팅 및 메모리 용량을 결정합니다. 필요한 DB 인스턴스 클래스는 DB 인스턴스의 처리력 및 메모리 요구 사항에 따라 다릅니다.

DB 인스턴스 클래스는 DB 인스턴스 클래스 유형과 크기로 구성됩니다. 예를 들어 db.r6g는 AWS Graviton2 프로세서로 구동되는 메모리 최적화 DB 인스턴스 클래스 유형입니다. db.r6g 인스턴스 유형 내에서 db.r6g.2xlarge는 DB 인스턴스 클래스입니다. 이 클래스의 크기는 2xlarge입니다.

인스턴스 클래스 요금에 대한 자세한 내용은 Amazon RDS 요금을 참조하세요.

DB 인스턴스 클래스 유형, 지원되는 DB 엔진, 지원되는 AWS 리전, 또는 DB 인스턴스 클래스의 하드웨어 사양에 대한 자세한 내용은 다음 섹션을 참조하세요.

주제

DB 인스턴스 클래스 유형

Amazon Aurora는 다음 사용 사례를 위한 DB 인스턴스 클래스를 지원합니다.

- Aurora Serverless v2
- 메모리 최적화
- 버스트 가능한 성능
- 최적화된 읽기

Amazon EC2 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 설명서의 인스턴스 유형을 참조하세요.

Aurora Serverless v2 인스턴스 클래스 유형

다음은 사용 가능한 Aurora Serverless v2 유형입니다.

- db.serverless - Aurora Serverless v2에서 사용하는 특수 DB 인스턴스 클래스 유형. Aurora는 워크로드의 변화에 따라 컴퓨팅, 메모리 및 네트워크 리소스를 동적으로 조정합니다. 자세한 내용은 Aurora Serverless v2 사용하기 단원을 참조하세요.

메모리 최적화 인스턴스 클래스 유형

메모리 최적화 X 패밀리는 다음과 같은 인스턴스 클래스를 지원합니다.

- db.x2g - 메모리 집약적 애플리케이션에 최적화되고 AWS Graviton2 프로세서로 구동되는 인스턴스 클래스입니다. 이러한 인스턴스 클래스는 메모리 GiB당 낮은 비용을 제공합니다.

  AWS Graviton2 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.

메모리 최적화 R 패밀리는 다음과 같은 인스턴스 클래스 유형을 지원합니다.

- db.r8g - AWS Graviton4 프로세서로 구동되는 인스턴스 클래스입니다. 이러한 인스턴스 클래스는 메모리 사용량이 많은 워크로드를 실행하는 데 적합합니다. 이러한 인스턴스는 7세대 AWS Graviton3 기반 db.r7g 인스턴스보다 3배 더 많은 vCPU와 메모리를 통해 더 큰 인스턴스 크기를 제공합니다.

  AWS Graviton4 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.

- db.r7g - AWS Graviton3 프로세서로 구동되는 인스턴스 클래스. 이러한 인스턴스 클래스는 메모리 사용량이 많은 워크로드를 실행하는 데 적합합니다.

  AWS Graviton3 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.

- db.r7i - 4세대 인텔 제온 스케일러블 프로세서로 구동되는 인스턴스 클래스입니다. 이러한 인스턴스는 SAP 인증을 받았으며 메모리 집약적인 워크로드를 실행하는 데 적합합니다. 4세대 인텔 제온 스케일러블 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.

- db.r6g - AWS Graviton2 프로세서로 구동되는 인스턴스 클래스 이러한 인스턴스 클래스는

  AWS Graviton2 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.

- db.r6i - 3세대 인텔 제온 스케일러블 프로세서로 구동되는 인스턴스 클래스입니다. SAP 인증을 받은 이러한 인스턴스 클래스는 메모리 사용량이 많은 워크로드를 실행하는 데 적합합니다.
- db.r5 – 메모리 집약적 애플리케이션에 최적화된 인스턴스 클래스. 이러한 인스턴스 클래스는 향상된 네트워킹과 Amazon Elastic Block Store(Amazon EBS) 성능을 제공합니다. 전용 하드웨어 및 경량 하이퍼바이저 결합된 AWS Nitro System을 기반으로 합니다.
- db.r4 – 이러한 인스턴스 클래스는 Aurora MySQL 2.x 및 Aurora PostgreSQL 11 및 12 버전에서만 지원됩니다. db.r4 DB 인스턴스 클래스를 사용하는 모든 Aurora DB 클러스터의 경우 최대한 빨리 더 높은 세대의 인스턴스 클래스로 업그레이드하는 것이 좋습니다.

  db.r4 인스턴스 클래스를 Aurora I/O-Optimized 클러스터 스토리지 구성에 사용할 수 없습니다.

버스트 가능한 성능 인스턴스 클래스 유형

다음은 사용 가능한 버스트 가능 성능 DB 인스턴스 클래스 유형입니다.

- db.t4g - Arm 기반 AWS Graviton2 프로세서로 구동되는 범용 인스턴스 클래스입니다. 이러한 인스턴스 클래스는 광범위한 범용 워크로드 집합에 대해 이전의 버스트 가능 성능 DB 인스턴스 클래스보다 더 나은 가성비를 제공합니다. Amazon RDS db.t4g 인스턴스는 무제한 모드로 구성됩니다. 즉, 추가 요금을 지불하면 24시간 동안 기준 이상으로 높일 수 있습니다.

  AWS Graviton2 프로세서로 구동되는 DB 인스턴스 클래스 중 하나를 사용하도록 DB 인스턴스를 수정할 수 있습니다. 이렇게 하려면 다른 DB 인스턴스 수정과 동일한 단계를 완료해야 합니다.
- db.t3 – CPU 사용률을 최대로 버스트할 수 있는 기능으로 기준 성능 수준을 제공하는 인스턴스 클래스입니다. db.t3 인스턴스는 무제한 모드로 구성됩니다. 이 인스턴스 클래스는 이전의 db.t2 인스턴스 클래스보다 더 많은 컴퓨팅 용량을 제공합니다. 전용 하드웨어 및 경량 하이퍼바이저 결합된 AWS Nitro System을 기반으로 합니다. 이러한 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비 프로덕션 서버에만 사용하는 것이 좋습니다.
- db.t2 – CPU 사용률을 최대로 버스트할 수 있는 기능으로 기준 성능 수준을 제공하는 인스턴스 클래스입니다. db.t2 인스턴스는 무제한 모드로 구성됩니다. 이러한 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비 프로덕션 서버에만 사용하는 것이 좋습니다.

  db.t2 인스턴스 클래스를 Aurora I/O-Optimized 클러스터 스토리지 구성에 사용할 수 없습니다.

> **Note**
>
> T DB 인스턴스 클래스는 개발, 테스트 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 개발 및 테스트에 T 인스턴스 클래스 사용 섹션을 참조하세요.

DB 인스턴스 클래스의 하드웨어 사양은 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양 단원을 참조하십시오.

최적화된 읽기 인스턴스 클래스 유형

다음은 사용 가능한 최적화된 읽기 인스턴스 클래스 유형입니다.

- db.r6g - AWS Graviton2 프로세서로 구동되는 인스턴스 클래스 이러한 인스턴스 클래스는 메모리 사용량이 많은 워크로드를 실행하는 데 이상적이며 빠르고 지연 시간이 짧은 로컬 스토리지가 필요한 애플리케이션에 로컬 NVMe 기반 SSD 블록 스토리지를 제공합니다.
- db.r6id - 3세대 인텔 제온 스케일러블 프로세서로 구동되는 인스턴스 클래스입니다. SAP 인증을 받은 이러한 인스턴스 클래스는 메모리 사용량이 많은 워크로드에 적합합니다. 최대 1TiB의 메모리 및 최대 7.6TB의 직접 연결 NVMe 기반 SSD 스토리지를 제공합니다.

DB 인스턴스 클래스에 지원되는 DB 엔진

다음 표에서는 Amazon Aurora DB 엔진에 대해 지원되는 DB 인스턴스 클래스를 보여줍니다.

db.serverless - 자동 용량 조정을 지원하는 Aurora Serverless v2 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.serverless	Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.	Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

db.x2g - AWS Graviton2 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.x2g.16xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.12xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.8xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.4xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.2xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.x2g.large	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상

db.r6g - AWS Graviton2 프로세서로 구동되는 최적화된 읽기 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6gd.16xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상
db.r6gd.12xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상
db.r6gd.8xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상
db.r6gd.4xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상
db.r6gd.2xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6gd.xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상

db.r6id – 최적화된 읽기 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6id.32xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상
db.r6id.24xlarge	아니요	16.1 이상, 15.4 이상, 14.9 이상

db.r7g – AWS Graviton3 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r7g.16xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.12xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.8xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.4xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.2xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.xlarge	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상
db.r7g.large	2.12.0 이상, 3.03.1 이상 버전	15.2 이상, 14.7 이상, 13.10 이상

db.r6g – AWS Graviton2 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6g.16xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6g.12xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.r6g.8xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.r6g.4xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.r6g.2xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.r6g.xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상
db.r6g.large	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.8 이상, 11.9, 11.12 이상

db.r6i – 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6i.32xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.24xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.16xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.12xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.8xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r6i.4xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.2xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상
db.r6i.large	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.5 이상, 12.9 이상

db.r5 – 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r5.24xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.16xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.12xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.8xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.4xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.2xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.xlarge	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전
db.r5.large	현재 사용 가능한 모든 버전	현재 사용 가능한 모든 버전

db.r4 – 메모리 최적화 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.r4.16xlarge	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.r4.8xlarge	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.r4.4xlarge	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.r4.2xlarge	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.r4.xlarge	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.r4.large	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요

db.t4g - AWS Graviton2 프로세서로 구동되는 성능 버스트 가능 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.t4g.2xlarge	아니요	아니요
db.t4g.xlarge	아니요	아니요
db.t4g.xlarge	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.7 이상, 11.12 이상
db.t4g.medium	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.7 이상, 11.12 이상
db.t4g.small	아니요	아니요

db.t3 – 버스트 가능 성능 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.t3.2xlarge	아니요	아니요
db.t3.xlarge	아니요	아니요
db.t3.large	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.7 이상, 11.12 이상
db.t3.medium	현재 사용 가능한 모든 버전	15.2 이상, 14.3 이상, 13.3 이상, 12.7 이상, 11.12 이상
db.t3.small	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.t3.micro	아니요	아니요

db.t2 – 버스트 가능 성능 인스턴스 클래스

인스턴스 클래스	Aurora MySQL	Aurora PostgreSQL
db.t2.medium	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요
db.t2.small	모든 2.x 버전, 3.x 버전에서 지원되지 않음	아니요

AWS 리전에서 DB 인스턴스 클래스 지원 확인

특정 AWS 리전의 각 DB 엔진에서 지원하는 DB 인스턴스 클래스를 확인하려면 여러 접근 방식 중 하나를 사용할 수 있습니다. AWS Management Console, Amazon RDS 요금 페이지 또는 describe-orderable-db-instance-options AWS CLI 명령을 사용할 수 있습니다.

> **Note**
>
> AWS Management Console에서 작업을 수행하면 특정 DB 엔진, DB 엔진 버전 및 AWS 리전에서 지원되는 DB 인스턴스 클래스가 자동으로 표시됩니다. 수행할 수 있는 작업의 예로는 DB 인스턴스 생성 및 수정 등이 있습니다.

목차

Amazon RDS 요금 페이지를 사용하여 AWS 리전에서 DB 인스턴스 클래스 지원 확인

Amazon Aurora 요금 페이지를 사용하여 특정 AWS 리전의 각 DB 엔진에서 지원하는 DB 인스턴스 클래스를 확인할 수 있습니다.

요금 페이지를 사용하여 리전의 각 엔진에서 지원하는 DB 인스턴스 클래스를 확인하려면

1. Amazon Aurora 요금으로 이동합니다.
2. AWS 요금 계산기 섹션에서 Amazon Aurora 엔진을 선택합니다.
3. 리전 선택에서 AWS 리전을 선택합니다.
4. 클러스터 구성 옵션에서 구성 옵션을 선택합니다.
5. 호환되는 인스턴스 섹션을 사용하여 지원되는 DB 인스턴스 클래스를 확인합니다.
6. (선택 사항) 계산기에서 다른 옵션을 선택한 다음 요약 저장 및 보기 또는 서비스 저장 및 추가를 선택합니다.

AWS CLI를 사용하여 AWS 리전에서 DB 인스턴스 클래스 지원 확인

AWS CLI를 사용하여 AWS 리전에서 특정 DB 엔진 및 DB 엔진 버전에 대해 지원되는 DB 인스턴스 클래스를 확인할 수 있습니다.

아래의 AWS CLI 예시를 사용하려면 유효한 DB 엔진, DB 엔진 버전, DB 인스턴스 클래스 및 AWS 리전 값을 입력해야 합니다. 다음 표에는 유효한 DB 엔진 값이 나와 있습니다.

엔진 이름	CLI 명령의 엔진 값	버전에 대한 추가 정보
MySQL 5.7 호환 및 8.0 호환 Aurora	aurora-mysql	Aurora MySQL 릴리스 정보의 Amazon Aurora MySQL 버전 2에 대한 데이터베이스 엔진 업데이트 및 Amazon Aurora MySQL 버전 3에 대한 데이터베이스 엔진 업데이트
Aurora PostgreSQL	aurora-postgresql	Aurora PostgreSQL 릴리스 정보

AWS 리전 이름에 대한 자세한 내용은 AWS 리전 섹션을 참조하세요.

다음 예는 describe-orderable-db-instance-options AWS 리전 명령을 사용하여 AWS CLI 리전에서 DB 인스턴스 클래스 지원을 확인하는 방법을 보여줍니다.

주제

- AWS 리전의 특정 DB 엔진 버전에서 지원하는 DB 인스턴스 클래스 나열
- AWS 리전에서 특정 DB 인스턴스 클래스를 지원하는 DB 엔진 버전 나열

AWS 리전의 특정 DB 엔진 버전에서 지원하는 DB 인스턴스 클래스 나열

AWS 리전 리전의 특정 DB 엔진 버전에서 지원하는 DB 인스턴스 클래스를 나열하려면 다음 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Windows의 경우:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
```

```
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

출력에는 각 DB 인스턴스 클래스에 대해 지원되는 엔진 모드도 표시됩니다.

예를 들어 다음 명령은 미국 동부(버지니아 북부)에서 Aurora PostgreSQL의 DB 엔진 버전 13.6에 대해 지원되는 DB 인스턴스 클래스를 나열합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Windows의 경우:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

AWS 리전에서 특정 DB 인스턴스 클래스를 지원하는 DB 엔진 버전 나열

AWS 리전 리전에서 특정 DB 인스턴스 클래스를 지원하는 DB 엔진 버전을 나열하려면 다음 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Windows의 경우:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

출력에는 각 DB 엔진 버전에 대해 지원되는 엔진 모드도 표시됩니다.

예를 들어 다음 명령은 US East (N. Virginia)에서 db.r5.large DB 인스턴스 클래스를 지원하는 Aurora PostgreSQL DB 엔진의 DB 엔진 버전을 나열합니다.

Linux, macOS, Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Windows의 경우:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양

이 섹션의 표에서 Aurora용 Amazon RDS DB 인스턴스 클래스에 대한 하드웨어 세부 정보를 확인할 수 있습니다.

각 DB 인스턴스 클래스에 대한 Aurora DB 엔진 지원에 관한 자세한 내용은 DB 인스턴스 클래스에 지원되는 DB 엔진 단원을 참조하십시오.

주제

- [Aurora에 대한 DB 인스턴스 클래스의 하드웨어 용어]
- [메모리 최적화 인스턴스 클래스에 대한 하드웨어 사양]
- [성능 버스트 가능 인스턴스 클래스에 대한 하드웨어 사양]

Aurora에 대한 DB 인스턴스 클래스의 하드웨어 용어

다음 용어는 DB 인스턴스 클래스의 하드웨어 사양을 기술하는 데 사용됩니다.

vCPU

가상 CPU(중앙 처리 유닛)의 수입니다. 가상 CPU는 DB 인스턴스 클래스를 비교하는 데 사용할 수 있는 용량 단위입니다. 특정 프로세서를 구매하거나 임차해 몇 개월 또는 몇 년간 사용하는 것이 아니라, 시간 단위로 용량을 임대합니다. 목표는 실제 기본 하드웨어의 제한 내에서 일정하고 구체적인 CPU 용량을 제공하는 것입니다.

ECU

Amazon EC2 인스턴스의 정수 처리 파워에 대한 상대적인 척도입니다. 개발자들이 다양한 인스턴스 클래스 간에 CPU 용량을 손쉽게 비교할 수 있도록 Amazon EC2 컴퓨팅 유닛(ECU)을 정의했습니다. 특정 인스턴스에 할당된 CPU의 용량은 이러한 EC2 컴퓨팅 유닛(ECU)으로 표현됩니다. 현재 ECU 한 개당 제공하는 CPU 용량은 1.0–1.2GHz 2007 Opteron 또는 2007 Xeon 프로세서와 동일합니다.

메모리(GiB)

DB 인스턴스에 할당되는 RAM(단위: 기비바이트)입니다. 메모리와 vCPU 간 일정한 비율이 존재하는 경우가 많다는 점에 유의하십시오. db.r5 인스턴스 클래스와 비슷한 vCPU 비율에 대한 메모리가 있는 db.r4 인스턴스 클래스를 예로 들 수 있습니다. 그러나 대부분의 사용 db.r5 인스턴스 클래스는 db.r4 인스턴스 클래스보다 더 낫고 더 일관성 있는 성능을 제공합니다.

최대 EBS 대역폭(Mbps)

초당 메가비트 단위의 최대 EBS 대역폭입니다. 이 값을 8로 나누면 초당 메가바이트 단위로 예상되는 처리량을 구할 수 있습니다.

> **Note**
>
> 이 수치는 DB 인스턴스 내 로컬 스토리지에 대한 I/O 대역폭을 나타냅니다. Aurora 클러스터 볼륨과의 통신에는 적용되지 않습니다.

네트워크 대역폭

다른 DB 인스턴스 클래스 대비 네트워크 속도입니다.

Amazon CloudWatch 지표를 사용하여 Aurora DB 인스턴스 처리량을 모니터링하는 방법에 대한 자세한 내용은 Amazon CloudWatch 지표를 사용한 Aurora MySQL에 대한 DB 인스턴스 사용량 평가 및 CloudWatch 지표를 사용한 Aurora PostgreSQL에 대한 DB 인스턴스 사용량 평가 섹션을 참조하세요.

메모리 최적화 인스턴스 클래스에 대한 하드웨어 사양

다음 표에는 메모리 최적화 인스턴스 클래스에 대한 컴퓨팅, 메모리, 스토리지 및 대역폭 사양이 나와 있습니다.

db.x2g – AWS Graviton2 프로세서를 지원하는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.x2g.16xlarge	64	—	1024	EBS 최적화 전용	19,000	25
db.x2g.12xlarge	48	—	768	EBS 최적화 전용	14,250	20
db.x2g.8xlarge	32	—	512	EBS 최적화 전용	9,500	12
db.x2g.4xlarge	16	—	256	EBS 최적화 전용	4,750	최대 10
db.x2g.2xlarge	8	—	128	EBS 최적화 전용	최대 4,750	최대 10
db.x2g.xlarge	4	—	64	EBS 최적화 전용	최대 4,750	최대 10
db.x2g.large	2	—	32	EBS 최적화 전용	최대 4,750	최대 10

db.r8g – AWS Graviton4 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r8g.48xlarge	192	—	1536	EBS 최적화 전용	40,000	50
db.r8g.24xlarge	96	—	768	EBS 최적화 전용	30,000개	40
db.r8g.16xlarge	64	—	512	EBS 최적화 전용	20,000건	30
db.r8g.12xlarge	48	—	384	EBS 최적화 전용	15,000	22.5
db.r8g.8xlarge	32	—	256	EBS 최적화 전용	10,000	15
db.r8g.4xlarge	16	—	128	EBS 최적화 전용	최대 10,000	최대 15
db.r8g.2xlarge	8	—	64	EBS 최적화 전용	최대 10,000	최대 15
db.r8g.xlarge	4	—	32	EBS 최적화 전용	최대 10,000	최대 12.5
db.r8g.large	2	—	16	EBS 최적화 전용	최대 10,000	최대 12.5

db.r7i - 4세대 인텔 제온 스케일러블 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r7i.48xlarge	192	—	1536	EBS 최적화 전용	40,000	50
db.r7i.24xlarge	96	—	768	EBS 최적화 전용	30,000개	37.5
db.r7i.16xlarge	64	—	512	EBS 최적화 전용	20,000건	25
db.r7i.12xlarge	48	—	384	EBS 최적화 전용	15,000	18.75
db.r7i.8xlarge	32	—	256	EBS 최적화 전용	10,000	12.5
db.r7i.4xlarge	16	—	128	EBS 최적화 전용	최대 10,000	최대 12.5
db.r7i.2xlarge	8	—	64	EBS 최적화 전용	최대 10,000	최대 12.5
db.r7i.xlarge	4	—	32	EBS 최적화 전용	최대 10,000	최대 12.5
db.r7i.large	2	—	16	EBS 최적화 전용	최대 10,000	최대 12.5

db.r7g – AWS Graviton3 프로세서를 포함되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r7g.16xlarge	64	—	512	EBS 최적화 전용	20,000건	30

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r7g.12xlarge	48	—	384	EBS 최적화 전용	15,000	22.5
db.r7g.8xlarge	32	—	256	EBS 최적화 전용	10,000	15
db.r7g.4xlarge	16	—	128	EBS 최적화 전용	최대 10,000	최대 15
db.r7g.2xlarge	8	—	64	EBS 최적화 전용	최대 10,000	최대 15
db.r7g.xlarge	4	—	32	EBS 최적화 전용	최대 10,000	최대 12.5
db.r7g.large	2	—	16	EBS 최적화 전용	최대 10,000	최대 12.5

db.r6id - 3세대 인텔 제온 스케일러블 프로세서 및 SSD 스토리지를 지원하는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6id.32xlarge	128	—	1,024	4 x 1900 NVMe SSD	40,000	50
db.r6id.24xlarge	96	—	768	4 x 1425 NVMe SSD	30,000개	37.5

db.r6gd – AWS Graviton2 프로세서 및 SSD 스토리지를 포함하는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	19,000	25
db.r6gd.12xlarge	48	—	384	2 x 1425 NVMe SSD	13,500	20
db.r6gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	9,000	12
db.r6gd.4xlarge	16	—	128	1 x 950 NVMe SSD	4,750	최대 10
db.r6gd.2xlarge	8	—	64	1 x 474 NVMe SSD	최대 4,750	최대 10
db.r6gd.xlarge	4	—	32	1 x 237 NVMe SSD	최대 4,750	최대 10

db.r6g – AWS Graviton2 프로세서를 포함되는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6g.16xlarge	64	—	512	EBS 최적화 전용	19,000	25
db.r6g.12xlarge	48	—	384	EBS 최적화 전용	13,500	20
db.r6g.8xlarge	32	—	256	EBS 최적화 전용	9,000	12
db.r6g.4xlarge	16	—	128	EBS 최적화 전용	4,750	최대 10

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6g.2xlarge	8	—	64	EBS 최적화 전용	최대 4,750	최대 10
db.r6g.xlarge	4	—	32	EBS 최적화 전용	최대 4,750	최대 10
db.r6g.large	2	—	16	EBS 최적화 전용	최대 4,750	최대 10

db.r6i - 3세대 인텔 제온 스케일러블 프로세서를 포함하는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6i.32xlarge	128	—	1,024	EBS 최적화 전용	40,000	50
db.r6i.24xlarge	96	—	768	EBS 최적화 전용	30,000개	37.5
db.r6i.16xlarge	64	—	512	EBS 최적화 전용	20,000건	25
db.r6i.12xlarge	48	—	384	EBS 최적화 전용	15,000	18.75
db.r6i.8xlarge	32	—	256	EBS 최적화 전용	10,000	12.5
db.r6i.4xlarge	16	—	128	EBS 최적화 전용	최대 10,000	최대 12.5
db.r6i.2xlarge	8	—	64	EBS 최적화 전용	최대 10,000	최대 12.5

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r6i.xlarge	4	—	32	EBS 최적화 전용	최대 10,000	최대 12.5
db.r6i.large	2	—	16	EBS 최적화 전용	최대 10,000	최대 12.5

db.r5 – 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r5.24xlarge	96	347	768	EBS 최적화 전용	19,000	25
db.r5.16xlarge	64	264	512	EBS 최적화 전용	13,600	20
db.r5.12xlarge	48	173	384	EBS 최적화 전용	9,500	12
db.r5.8xlarge	32	132	256	EBS 최적화 전용	6,800	10
db.r5.4xlarge	16	71	128	EBS 최적화 전용	4,750	최대 10
db.r5.2xlarge	8	38	64	EBS 최적화 전용	최대 4,750	최대 10
db.r5.xlarge	4	19	32	EBS 최적화 전용	최대 4,750	최대 10
db.r5.large	2	10	16	EBS 최적화 전용	최대 4,750	최대 10

db.r4 - 인텔 제온 스케일러블 프로세서를 포함하는 메모리 최적화 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.r4.16xlarge	64	195	488	EBS 최적화 전용	14,000	25
db.r4.8xlarge	32	99	244	EBS 최적화 전용	7,000	10
db.r4.4xlarge	16	53	122	EBS 최적화 전용	3,500	최대 10
db.r4.2xlarge	8	27	61	EBS 최적화 전용	1,700	최대 10
db.r4.xlarge	4	13.5	30.5	EBS 최적화 전용	850	최대 10
db.r4.large	2	7	15.25	EBS 최적화 전용	425	최대 10

성능 버스트 가능 인스턴스 클래스에 대한 하드웨어 사양

다음 표에는 성능 버스트 가능 인스턴스 클래스에 대한 컴퓨팅, 메모리, 스토리지 및 대역폭 사양이 나와 있습니다.

db.t4g - AWS Graviton2 프로세서로 구동되는 버스트 가능 성능 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.t4g.large	2	—	8	EBS 최적화 전용	최대 2,780	최대 5
db.t4g.medium	2	—	4	EBS 최적화 전용	최대 2,085개	최대 5

db.t3 – 버스트 가능 성능 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.t3.large	2	변수	8	EBS 최적화 전용	최대 2,048개	최대 5
db.t3.medium	2	변수	4	EBS 최적화 전용	최대 1,536개	최대 5
db.t3.small	2	변수	2	EBS 최적화 전용	최대 1,536개	최대 5

db.t2 – 버스트 가능 성능 인스턴스 클래스

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	인스턴스 스토리지(GiB)	최대 EBS 대역폭(Mbps)	네트워크 대역폭(Gbps)
db.t2.medium	2	변수	4	EBS 전용	—	보통
db.t2.small	1	변수	2	EBS 전용	—	낮음

Amazon Aurora 스토리지

다음에서 Aurora 스토리지 하위 시스템에 대해 자세히 배울 수 있습니다. Aurora는 Aurora 클러스터의 성능, 확장성 및 안정성에 중요한 요소인 분산 및 공유 스토리지 아키텍처를 사용합니다.

주제

- Amazon Aurora 스토리지 개요
- 클러스터 볼륨에 포함된 항목
- Amazon Aurora DB 클러스터의 스토리지 구성
- Aurora 스토리지 크기가 자동으로 조정되는 방법
- Aurora 데이터 스토리지 요금이 청구되는 방법

Amazon Aurora 스토리지 개요

Aurora 데이터는 SSD(Solid State Drive)를 사용하는 단일 가상 볼륨인 클러스터 볼륨에저장됩니다. 클러스터 볼륨은 동일한 AWS 리전에 속한 세 가용 영역의 데이터 사본으로 구성되어 있습니다. 이러한 가용 영역에서 데이터가 자동으로 복제되기 때문에 데이터 손실 가능성은 줄고 오히려 내구성이 크게 높아집니다. 이러한 복제를 통해 장애 조치 중에도 데이터베이스의 가용성이 높아집니다. 데이터 사본이 이미 나머지 가용 영역에 존재하여 DB 클러스터의 DB 인스턴스에 대한 데이터 요청을 계속 처리할 수 있기 때문입니다. 복제 양은 클러스터의 DB 인스턴스 수와는 관계가 없습니다.

Aurora는 비영구 임시 파일에 대해 별도의 로컬 스토리지를 사용합니다. 여기에는 쿼리 처리 중 대용량 데이터 세트를 정렬하고 인덱스를 작성하는 등의 목적으로 사용되는 파일이 포함됩니다. 자세한 내용은 Aurora MySQL에 대한 임시 스토리지 한도 및 Aurora PostgreSQL에 대한 임시 스토리지 한도 단원을 참조하세요.

클러스터 볼륨에 포함된 항목

Aurora 클러스터 볼륨에는 모든 사용자 데이터, 스키마 객체, 내부 메타데이터(예: 시스템 테이블 및 이진 로그)가 포함되어 있습니다. 예를 들어 Aurora는 클러스터 볼륨의 Aurora 클러스터에 대한 모든 테이블, 인덱스, BLOB(Binary Large Object), 저장 프로시저 등을 저장합니다.

Aurora 공유 스토리지 아키텍처는 데이터를 클러스터의 DB 인스턴스와 독립적으로 만듭니다. 예를 들어 Aurora가 테이블 데이터의 새 복사본을 만들지 않으므로 DB 인스턴스를 빠르게 추가할 수 있습니다. 대신에 DB 인스턴스는 이미 모든 데이터를 포함하는 공유 볼륨에 연결됩니다. 클러스터에서 기본 데이터를 제거하지 않고 클러스터에서 DB 인스턴스를 제거할 수 있습니다. 전체 클러스터를 삭제하는 경우에만 Aurora가 데이터를 제거합니다.

Amazon Aurora DB 클러스터의 스토리지 구성

Amazon Aurora에는 다음과 같은 두 가지 DB 클러스터 스토리지 구성이 있습니다.

- Aurora I/O-Optimized – I/O 집약적 애플리케이션을 위해 향상된 가격 대비 성능 및 예측 가능성을 제공합니다. 읽기 및 쓰기 I/O 작업에 대한 추가 비용 없이 DB 클러스터의 사용량 및 스토리지에 대한 비용만 지불하면 됩니다.

  Aurora I/O-Optimized는 I/O 지출이 총 Aurora 데이터베이스 지출의 25% 이상인 경우 가장 좋은 선택입니다.

  Aurora I/O-Optimized 클러스터 구성을 지원하는 DB 엔진 버전으로 DB 클러스터를 생성하거나 수정할 때 Aurora I/O-Optimized를 선택할 수 있습니다. 언제든지 Aurora Standard에서 Aurora I/O-Optimized로 전환할 수 있습니다.

- Aurora Standard – I/O 사용량이 보통인 대부분의 애플리케이션에 적합한 비용 효율적인 옵션입니다. DB 클러스터의 사용량 및 스토리지 외에도 I/O 작업에 대해 요청 1백만 건당의 표준 요금이 부과됩니다.

  Aurora Standard는 I/O 지출이 총 Aurora 데이터베이스 지출의 25% 미만인 경우 가장 좋은 선택입니다.

  30일마다 한 번 Aurora Standard에서 Aurora I/O-Optimized로 전환할 수 있습니다. NVMe 기반이 아닌 DB 인스턴스에 대해 Aurora Standard와 Aurora I/O-Optimized 스토리지 옵션 간에 전환하면 가동 중지 시간이 발생하지 않습니다. 그러나 NVMe 기반 DB 인스턴스의 경우 Aurora I/O-Optimized와 Aurora Standard 스토리지 옵션 간에 전환하려면 데이터베이스 엔진을 다시 시작해야 하므로 잠시 가동 중지 시간이 발생할 수 있습니다.

AWS 리전 및 버전 지원에 대해 자세히 알아보려면 클러스터 스토리지 구성을 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

Amazon Aurora 스토리지 구성 요금에 대해 자세히 알아보려면 Amazon Aurora 요금을 참조하세요.

DB 클러스터를 생성할 때 스토리지 구성을 선택하는 방법에 대한 자세한 내용은 DB 클러스터 생성 섹션을 참조하세요. DB 클러스터에 대한 스토리지 구성을 수정하는 방법에 대한 자세한 내용은 Amazon Aurora에 대한 설정 섹션을 참조하세요.

Aurora 스토리지 크기가 자동으로 조정되는 방법

데이터베이스의 데이터 용량이 늘어날수록 Aurora 클러스터 볼륨도 자동 확장됩니다. 각 엔진 버전의 최대 Aurora 클러스터 볼륨 크기에 대한 자세한 내용은 Amazon Aurora 크기 제한 섹션을 참조하세요. 이 자동 스토리지 조정은 고성능의 고도로 분산된 스토리지 하위 시스템을 통해 이루어집니다. 따라서 주요 목표가 안정성과 고가용성인 경우 중요한 엔터프라이즈 데이터에 Aurora를 선택하면 좋습니다.

볼륨 상태를 표시하려면 Aurora MySQL DB 클러스터를 위한 볼륨 상태 표시 또는 Aurora PostgreSQL DB 클러스터를 위한 볼륨 상태 표시 섹션을 참조하세요. 스토리지 비용과 다른 우선 순위의 균형을 맞추려면 스토리지 조정에서 CloudWatch를 통해 Amazon Aurora 지표 `AuroraVolumeBytesLeftTotal` 및 `VolumeBytesUsed`를 모니터링하는 방법을 참조하세요.

Aurora 데이터가 제거되면 해당 데이터에 할당된 공간이 복원됩니다. 데이터 제거의 예로는 테이블 삭제 또는 자르기 등이 있습니다. 이렇게 스토리지 사용량이 자동으로 줄어들면 스토리지 요금을 최소화할 수 있습니다.

> **Note**
>
> 여기서 설명하는 스토리지 제한 및 동적 크기 조정 동작은 클러스터 볼륨에 저장된 영구 테이블 및 기타 데이터에 적용됩니다.
> Aurora PostgreSQL의 경우 임시 테이블의 데이터가 로컬 DB 인스턴스에 저장됩니다.
> Aurora MySQL 버전 2의 경우 임시 테이블 데이터는 기본적으로 라이터 인스턴스의 경우 클러스터 볼륨에, 리더 인스턴스의 경우 로컬 스토리지에 저장됩니다. 자세한 내용은 온디스크 임시 테이블에 대한 스토리지 엔진 섹션을 참조하세요.
> Aurora MySQL 버전 3의 경우 임시 테이블 데이터는 로컬 DB 인스턴스 또는 클러스터 볼륨에 저장됩니다. 자세한 내용은 Aurora MySQL 버전 3의 새로운 임시 테이블 동작 섹션을 참조하세요.
> 로컬 스토리지에 있는 임시 테이블의 최대 크기는 DB 인스턴스의 최대 로컬 스토리지 크기로 제한됩니다. 로컬 스토리지 크기는 사용하는 인스턴스 클래스에 따라 다릅니다. 자세한 내용은 Aurora MySQL에 대한 임시 스토리지 한도 및 Aurora PostgreSQL에 대한 임시 스토리지 한도(을)를 참조하세요.

클러스터 볼륨의 최대 크기 및 데이터 제거 시 자동 크기 조정과 같은 일부 스토리지 기능은 클러스터의 Aurora 버전에 따라 다릅니다. 자세한 내용은 스토리지 조정 섹션을 참조하세요. 또한 스토리지 문제를 방지하는 방법과 클러스터에서 할당된 스토리지와 사용 가능한 공간을 모니터링하는 방법을 배울 수 있습니다.

Aurora 데이터 스토리지 요금이 청구되는 방법

Aurora 클러스터 볼륨은 특정 엔진 버전에서 최대 256테비바이트(TiB)까지 확장될 수 있지만 요금은 Aurora 클러스터 볼륨에서 사용한 공간에 대해서만 청구됩니다. 이전 Aurora 버전에서는 데이터를 제거할 때 복원된 공간을 클러스터 볼륨이 다시 사용할 수 있지만 할당된 스토리지 공간은 줄어들지 않습니다. 하지만 이제 테이블이나 데이터베이스를 삭제하는 등의 방법으로 Aurora 데이터를 제거하면 할당된 전체 스토리지 공간이 그만큼 감소합니다. 따라서 더 이상 필요하지 않은 테이블, 인덱스, 데이터베이스 등을 삭제하면 스토리지 요금을 줄일 수 있습니다.

> **Tip**
>
> 팁 동적 크기 조정 기능이 없는 이전 버전에서는 클러스터의 스토리지 사용량을 재설정하려면 논리적 덤프를 수행한 후 새 클러스터로 복구해야 했습니다. 데이터 양이 많을 경우 이 작업은 오랜 시간이 걸릴 수 있습니다. 이러한 상황이 발생하면 클러스터를 동적 볼륨의 크기 조정을 지원하는 버전으로 업그레이드하는 것이 좋습니다.

동적 크기 조정을 지원하는 Aurora 버전과 클러스터의 스토리지 사용량을 모니터링하여 스토리지 요금을 최소화하는 방법에 대한 내용은 스토리지 조정 단원을 참조하세요. Aurora 백업 스토리지 요금에 대한 자세한 정보는 Amazon Aurora 백업 스토리지 사용량 파악 단원을 참조하세요. Aurora 데이터 스토리지에 대한 요금 정보는 Amazon RDS for Aurora 요금을 참조하세요.

Amazon Aurora 안정성

Aurora는 안정성, 내구성 및 내결함성을 고려하여 설계되었습니다. Aurora DB 클러스터는 Aurora 복제본을 추가하여 다른 가용 영역에 배포하는 등의 방법으로 가용성을 높이도록 설계할 수 있습니다. 그 밖에도 Aurora에는 안정적인 데이터베이스 솔루션을 위한 자동 기능이 몇 가지 포함되어 있습니다.

주제

스토리지 자동 복구

Aurora는 3개의 가용 영역에 여러 개의 복사본을 보관하고 있기 때문에 디스크 결함으로 인한 데이터 손실 가능성이 최소화됩니다. 또한 Aurora는 클러스터 볼륨을 구성하는 디스크 볼륨에서 장애를 자동으로 감지합니다. 예를 들어 디스크 볼륨 세그먼트에 결함이 발생하면 Aurora가 즉시 해당 세그먼트를 복구합니다. Aurora가 디스크 세그먼트를 복구할 때 클러스터 볼륨을 구성하는 다른 볼륨의 데이터를 사용하여 복구된 세그먼트의 데이터가 최신 상태인지 확인합니다. 그 결과, Aurora는 데이터 손실을 방지하고 디스크 장애로부터 복구하기 위해 특정 시점 복구를 수행해야 할 필요성을 줄여줍니다.

유지 가능한 페이지 캐시

Aurora에서는 각 DB 인스턴스의 페이지 캐시가 데이터베이스와 별도의 프로세스로 관리되며, 이를 통해 페이지 캐시가 데이터베이스와 상관없이 유지됩니다. (페이지 캐시는 Aurora MySQL에서는 InnoDB 버퍼 풀, Aurora PostgreSQL에서는 버퍼 캐시라고도 합니다.)

드문 경우이지만 데이터베이스 장애가 발생할 경우, 페이지 캐시는 메모리에 남아 데이터베이스가 재시작될 때 현재 데이터 페이지를 페이지 캐시에 '웜' 상태로 유지합니다. 이렇게 하면 페이지 캐시를 '워밍업'하기 위해 초기 쿼리가 읽기 I/O 작업을 실행할 필요가 없으므로 성능이 향상됩니다.

Aurora MySQL의 경우 재부팅 및 장애 조치 시 페이지 캐시 동작은 다음과 같습니다.

- 리더 인스턴스를 재부팅하지 않고 라이터 인스턴스를 재부팅할 수 있습니다.
  - 라이터 인스턴스가 재부팅될 때 리더 인스턴스가 재부팅되지 않으면 재부팅되지 않은 리더 인스턴스의 페이지 캐시가 손실되지 않습니다.
  - 라이터 인스턴스가 재부팅될 때 리더 인스턴스가 재부팅되면 재부팅되는 리더 인스턴스의 페이지 캐시가 손실됩니다.
- 리더 인스턴스가 재부팅되면 라이터 및 리더 인스턴스의 페이지 캐시가 모두 유지됩니다.
- DB 클러스터가 장애 조치될 때 이 효과는 라이터 인스턴스가 재부팅될 때와 유사합니다. 새 라이터 인스턴스(이전 리더 인스턴스)에서는 페이지 캐시가 유지되지만, 리더 인스턴스(이전 라이터 인스턴스)에서는 페이지 캐시가 유지되지 않습니다.

Aurora PostgreSQL의 경우 클러스터 캐시 관리를 사용하여 장애 조치 후 라이터 인스턴스가 되는 지정된 리더 인스턴스의 페이지 캐시를 보관할 수 있습니다. 자세한 내용은 장애 조치 후 Aurora PostgreSQL용 클러스터 캐시 관리를 통한 신속한 복구 단원을 참조하십시오.

예기치 않은 재시작 시 복구

Aurora는 예기치 않은 재시작에서 거의 즉각적으로 복구하여 바이너리 로그 없이 애플리케이션 데이터를 계속 제공하도록 설계되었습니다. Aurora는 병렬 스레드에서 비동기적으로 복구를 수행하여 데이터베이스가 열리고 예기치 않은 재시작 후 즉시 사용할 수 있도록 합니다.

자세한 내용은 Aurora DB 클러스터의 내결함성 및 데이터베이스 재시작 시간 단축을 위한 최적화 단원을 참조하세요.

다음은 Aurora MySQL에서의 바이너리 로깅 및 예기치 않은 재시작 복구 시 고려 사항입니다.

- Aurora 바이너리 로깅을 활성화하면 DB 인스턴스로 하여금 강제로 바이너리 로그 복구를 수행하도록 하므로 예기치 않은 재시작 후 복구 시간에 직접적인 영향을 줍니다.
- 사용되는 이진 로깅 유형은 로깅의 크기와 효율에 영향을 미칩니다. 데이터베이스 활동의 양이 동일하더라도 형식에 따라 이진 로그에서 더 많은 정보가 로깅됩니다. 다음과 같은 `binlog_format` 파라미터 설정으로 로그 데이터의 양이 달라집니다.
  - `ROW` – 최대 로그 데이터
  - `STATEMENT` – 최소 로그 데이터
  - `MIXED` – 일반적으로 데이터 무결성과 성능의 최상의 조합을 제공하는 적당량의 로그 데이터

  이진 로그 데이터의 양은 복구 시간에 영향을 미칩니다. 이진 로그에 로깅된 데이터가 더 많은 경우, DB 인스턴스는 복구 도중 더 많은 데이터를 처리해야 하므로 복구 시간이 길어집니다.

- 이진 로깅으로 계산 오버헤드를 줄이고 복구 시간을 개선하기 위해 향상된 binlog를 사용할 수 있습니다. 향상된 binlog는 데이터베이스 복구 시간을 최대 99% 개선합니다. 자세한 내용은 Aurora MySQL에 대해 향상된 binlog 설정 단원을 참조하십시오.
- Aurora은 DB 클러스터 내에서 데이터를 복제하거나 특정 시점 복원(PITR)을 수행하기 위해 바이너리 로그를 필요로 하지 않습니다.
- 외부 복제(또는 외부 이진 로그 스트림)에 이진 로그가 필요하지 않은 경우, `binlog_format` 파라미터를 `OFF`로 설정하여 이진 로깅을 비활성화하는 것이 좋습니다. 이렇게 하면 복구 시간이 단축됩니다.

Aurora 이진 로깅 및 복제에 대한 자세한 내용은 Amazon Aurora를 사용한 복제 단원을 참조하십시오. 다양한 MySQL 복제 유형의 영향에 대한 자세한 내용은 MySQL 설명서의 Advantages and Disadvantages of Statement-Based and Row-Based Replication을 참조하세요.

Amazon Aurora 보안

Amazon Aurora 보안은 다음과 같이 세 가지 수준에서 관리됩니다.

- Aurora DB 클러스터 및 DB 인스턴스에서 Amazon RDS 관리 작업을 수행할 수 있는 사용자를 제어하려면 AWS Identity and Access Management(IAM)를 사용합니다. IAM 자격 증명을 사용하여 AWS에 연결할 때, AWS 계정은 Amazon RDS 관리 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책을 보유하고 있어야 합니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 단원을 참조하십시오.

  IAM을 사용해 Amazon RDS 콘솔에 액세스하려면 먼저 사용자 자격 증명으로 AWS Management Console에 로그인한 다음 https://console.aws.amazon.com/rds에서 Amazon RDS 콘솔로 이동합니다.
- Aurora DB 클러스터는 Amazon VPC 서비스를 기반으로 Virtual Private Cloud(VPC)에 생성해야 합니다. Aurora DB 클러스터에서 DB 인스턴스의 엔드포인트 및 포트에 연결할 수 있는 디바이스 또는 Amazon EC2 인스턴스를 제어하려면 VPC 보안 그룹을 사용합니다. 전송 계층 보안(TLS)/Secure Sockets Layer(SSL)를 사용하여 이러한 엔드포인트 및 포트 연결을 만들 수 있습니다. 그 밖에도 기업의 방화벽 규칙을 통해 기업에서 이용하는 디바이스의 DB 인스턴스 연결 여부를 제어하는 것도 가능합니다. VPC에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 단원을 참조하십시오.
- Amazon Aurora DB 클러스터에 대한 로그인 및 권한을 인증하기 위해서는 다음 접근 방식 중 하나를 따르거나 두 방식을 조합할 수 있습니다.
  - 독립형 MySQL 또는 PostgreSQL DB 인스턴스와 동일한 접근법을 사용할 수 있습니다.

SQL 명령을 사용하거나, 데이터베이스 스키마 테이블을 수정하는 등 독립형 MySQL 또는 PostgreSQL DB 인스턴스에서 로그인 및 권한을 인증하는 기법이 Aurora에서도 유효합니다. 자세한 내용은 Amazon Aurora MySQL를 사용한 보안 또는 Amazon Aurora PostgreSQL를 사용한 보안 섹션을 참조하세요.

- IAM 데이터베이스 인증을 사용할 수 있습니다.

  IAM 데이터베이스 인증의 경우, 사용자 또는 IAM 역할 및 인증 토큰을 이용해 Aurora DB 클러스터에 인증합니다. 인증 토큰은 서명 버전 4 서명 프로세스를 통해 생성하는 고유 값입니다. IAM 데이터베이스 인증을 사용하면 동일한 자격 증명을 사용해 AWS 리소스 및 데이터베이스에 대한 액세스를 제어할 수 있습니다. 자세한 내용은 IAM 데이터베이스 인증 단원을 참조하십시오.

- Aurora PostgreSQL 및 Aurora MySQL에 Kerberos 인증을 사용할 수 있습니다.

  사용자가 Aurora PostgreSQL 및 Aurora MySQL DB 클러스터에 연결할 때 Kerberos를 사용하여 사용자를 인증할 수 있습니다. 이 경우 DB 클러스터는 AWS Directory Service for Microsoft Active Directory와 함께 작동하여 Kerberos 인증을 활성화합니다. AWS Directory Service for Microsoft Active Directory는 AWS Managed Microsoft AD라고도 합니다. 모든 자격 증명을 동일한 디렉터리에 보관하면 시간과 노력을 절약할 수 있습니다. 여러 DB 클러스터에 대한 자격 증명을 보관하고 관리할 수 있는 중앙 집중식 공간이 있습니다. 디렉터리를 사용하면 전체 보안 프로필을 향상할 수도 있습니다. 자세한 내용은 Aurora PostgreSQL과 함께 Kerberos 인증 사용 및 Aurora MySQL에 Kerberos 인증 사용 단원을 참조하세요.

구성 보안에 대한 자세한 내용은 Amazon Aurora의 보안 단원을 참조하십시오.

Aurora DB 클러스터에 SSL 사용

Amazon Aurora DB 클러스터는 Amazon RDS DB 인스턴스와 동일한 프로세스 및 퍼블릭 키를 사용하여 애플리케이션의 Secure Sockets Layer(SSL) 연결을 지원합니다. 자세한 내용은 Amazon Aurora MySQL를 사용한 보안, Amazon Aurora PostgreSQL를 사용한 보안 또는 Aurora Serverless v1에서 TLS/SSL 사용 단원을 참조하십시오.

Amazon Aurora의 고가용성

Amazon Aurora 아키텍처는 스토리지와 컴퓨팅이 분리됩니다. Aurora에는 DB 클러스터의 데이터에 적용되는 몇 가지 고가용성 기능이 포함되어 있습니다. 클러스터의 일부 또는 모든 DB 인스턴스를 사용할 수 없는 경우에도 데이터는 안전하게 유지됩니다. 다른 고가용성 기능이 DB 인스턴스에 적용됩니다. 이러한 기능은 하나 이상의 DB 인스턴스가 애플리케이션의 데이터베이스 요청을 처리할 수 있도록 준비하는 데 도움이 됩니다.

주제

Aurora 데이터의 고가용성

Aurora는 단일 AWS 리전에서 다중 가용 영역에 걸쳐 DB 클러스터에 데이터 복사본을 저장합니다. Aurora는 DB 클러스터의 인스턴스가 여러 가용성 영역에 걸쳐 있는지 여부에 관계없이 이러한 복사본을 저장합니다. Aurora에 대한 자세한 내용은 Amazon Aurora DB 클러스터 관리 단원을 참조하십시오.

기본 DB 인스턴스에 데이터가 기록되면 Aurora에서 가용 영역의 데이터를 클러스터 볼륨과 연결된 6개의 스토리지 노드에 동기적으로 복제합니다. 이 방법은 데이터 중복을 제공하고, I/O 중지를 없애고, 시스템 백업 중에 지연 시간 스파이크를 최소화합니다. DB 인스턴스를 고가용성으로 실행하면 계획된 시스템 유지 관리 중 가용성을 향상시킬 수 있으며, 데이터베이스에서 오류 및 가용 영역 중단이 일어나는 것을 방지할 수 있습니다. 가용 영역에 대한 자세한 정보는 리전 및 가용 영역 단원을 참조하십시오.

Aurora DB 인스턴스의 고가용성

기본(라이터) 인스턴스를 생성한 후에 최대 15개의 읽기 전용 Aurora 복제본을 생성할 수 있습니다. Aurora 복제본은 리더 인스턴스로도 알려져 있습니다. Aurora 복제본은 비동기 복제를 사용하여 기본 인스턴스 성능에 영향을 주지 않고 고가용성을 지원합니다.

일상적인 작업 중에 리더 인스턴스에서 SELECT 쿼리를 처리하여 읽기 집약적 응용 프로그램의 일부 작업을 오프로드할 수 있습니다. 문제가 기본 인스턴스에 영향을 미치는 경우 이러한 리더 인스턴스 중 하나가 기본 인스턴스 역할을 인수합니다. 이 메커니즘을 장애 조치(failover)라고 합니다. 많은 Aurora 기능이 장애 조치 메커니즘에 적용됩니다. 예를 들어 Aurora는 데이터베이스 문제를 감지하고 필요한 경우 장애 조치 메커니즘을 자동으로 활성화합니다. 또한 Aurora는 장애 조치 완료 시간을 단축하는 기능을 갖추고 있습니다. 이렇게 하면 장애 조치 중에 데이터베이스를 쓰기에 사용할 수 없는 시간이 최소화됩니다.

Aurora는 최대한 빨리 복구하도록 설계되었으며, 가장 빠른 복구 경로는 대개 동일한 DB 인스턴스를 다시 시작하거나 장애 조치하는 것입니다. 재시작은 장애 조치보다 더 빠르고 오버헤드가 적습니다.

장애 조치로 인해 새 기본 인스턴스를 승격하는 경우에도 동일하게 유지되는 연결 문자열을 사용하려면 클러스터 엔드포인트에 연결합니다. 클러스터 엔드포인트는 항상 클러스터의 현재 기본 인스턴스를 나타냅니다. 클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora 엔드포인트 연결 섹션을 참조하세요.

> **Tip**
>
> 각 AWS 리전 내에서 가용 영역(AZ)은 중단이 발생할 경우 격리를 제공하기 위해 서로 구별되는 위치를 나타냅니다. DB 클러스터의 가용성을 개선하려면 여러 AZ의 DB 클러스터에 기본 인스턴스와 리더 인스턴스를 배포하는 것이 좋습니다. 이렇게 하면 전체 AZ에 영향을 미치는 문제로 인해 클러스터가 중단되지 않습니다.
> 클러스터를 생성할 때 간단한 선택을 수행하여 다중 AZ DB 클러스터를 설정할 수 있습니다. AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용할 수 있습니다. 또한 새 리더 DB 인스턴스를 추가하고 다른 AZ를 지정하여 기존 Aurora DB 클러스터를 다중 AZ DB 클러스터로 만들 수 있습니다.

Aurora Global Database를 사용한 AWS 리전 간 고가용성

여러 AWS 리전 간의 고가용성을 위해 Aurora Global Database를 설정할 수 있습니다. 각각의 Aurora Global Database는 여러 AWS 리전에 걸쳐 있어, 대기 시간이 짧은 글로벌 읽기와 AWS 리전에 걸쳐 중단에 대한 재해 복구를 지원합니다. Aurora는 기본 AWS 리전에서 각 보조 리전으로 모든 데이터 및 업데이트를 비동기식으로 복제합니다. 자세한 내용은 Amazon Aurora Global Database 사용 단원을 참조하십시오.

Aurora DB 클러스터의 내결함성

Aurora DB 클러스터는 내결함성을 고려하여 설계되었습니다. 클러스터 볼륨은 단일 AWS 리전에 속하는 다중 가용 영역(AZ)을 모두 아우르며, 각 가용 영역에는 클러스터 볼륨 데이터의 사본이 복사됩니다. 이 기능은 가용 영역 한 곳에서 결함이 발생하더라도 DB 클러스터가 잠시 서비스가 중단될 뿐 전혀 데이터 손실 없이 결함을 견딜 수 있음을 의미합니다.

DB 클러스터의 기본 인스턴스에 결함이 발생하면 Aurora는 다음 두 가지 방법 중 하나를 사용하여 자동으로 새 기본 인스턴스로 장애 조치합니다.

- 기존 Aurora 복제본을 새 기본 인스턴스로 승격시킴
- 새로운 기본 인스턴스 만들기

DB 클러스터에 Aurora 복제본이 하나 이상인 경우에는 장애가 발생하더라도 Aurora 복제본이 기본 인스턴스로 승격됩니다. 이 실패 이벤트로 인해 예외적으로 실패하는 읽기 및 쓰기 작업 동안 짧은 중단이 발생합니다. 하지만, 일반적인 서비스 복구 시간은 60초 미만이지만 대부분 30초 미만에 복원됩니다. DB 클러스터의 가용성을 높이려면 최소 하나 이상의 Aurora 복제본을 둘 이상의 다른 가용 영역에 생성하는 것이 바람직합니다.

> **Tip**
>
> Aurora MySQL에서는 클러스터에 리더 DB 인스턴스를 2개 이상 보유하여 장애 조치 중에 가용성을 개선할 수 있습니다. Aurora MySQL에서 Aurora는 장애 조치 대상인 라이터 DB 인스턴스와 리더 인스턴스만 다시 시작합니다. 클러스터의 다른 리더 인스턴스는 장애 조치 중에 리더 엔드포인트에 대한 연결을 통해 쿼리를 처리하는 데 계속해서 사용할 수 있습니다.
> Aurora DB 클러스터와 함께 RDS 프록시를 사용하여 장애 조치 중에 가용성을 개선할 수도 있습니다. 자세한 내용은 Amazon RDS 프록시를 사용하는 고가용성 단원을 참조하십시오.

각 복제본에 우선 순위를 지정함으로써 장애 이후 기본 인스턴스로 승격할 Aurora 복제본 순서를 사용자 지정할 수 있습니다. 우선 순위 범위는 가장 높은 값인 0부터 가장 낮은 값인 15까지입니다. 기본 인스턴스에 결함이 발생하면 Amazon RDS는 우선순위가 가장 높은 Aurora 복제본을 새로운 기본 인스턴스로 승격시킵니다. Aurora 복제본의 우선 순위는 언제든지 수정할 수 있습니다. 우선 순위 수정으로 인해 장애 조치가 트리거되지는 않습니다.

둘 이상의 Aurora 복제본이 동일한 우선 순위를 공유하여 승격 계층을 만들 수도 있습니다. 둘 이상의 Aurora 복제본이 동일한 우선 순위를 공유하면 Amazon RDS는 크기가 가장 큰 복제본을 승격시킵니다. 둘 이상의 Aurora Replicas가 동일한 우선 순위와 크기를 공유하면 Amazon RDS는 동일한 승격 티어에서 임의의 복제본을 승격시킵니다.

> **Note**
>
> 장애 조치 대상을 식별하는 데는 몇 가지 요인이 관여합니다. 장애 조치 시도가 5회 실패하면 승격 계층이 더 이상 고려되지 않습니다.

DB 클러스터에 Aurora 복제본이 포함되어 있지 않으면 기본 인스턴스가 실패 이벤트 중에 동일한 AZ에 다시 생성됩니다. 이 실패 이벤트로 인해 예외적으로 실패하는 읽기 및 쓰기 작업 동안 중단이 발생합니다. 새로운 기본 인스턴스가 생성도면 서비스도 복구되지만 보통 10분 미만의 시간이 걸립니다. Aurora 복제본을 기본 인스턴스로 승격시키는 것이 기본 인스턴스를 새로 생성하는 것보다 훨씬 빠릅니다.

전체 AZ에 영향을 미치는 중단으로 인해 클러스터의 기본 인스턴스를 사용할 수 없다고 가정해 보겠습니다. 이 경우, 새 기본 인스턴스를 온라인 상태로 만드는 방법은 클러스터가 다중 AZ 구성을 사용하는지 여부에 따라 달라집니다.

- 프로비저닝 클러스터 또는 Aurora Serverless v2 클러스터에 다른 AZ의 리더 인스턴스가 포함되어 있는 경우, Aurora은(는) 장애 조치 메커니즘을 사용하여 해당 리더 인스턴스 중 하나를 새 기본 인스턴스로 승격시킵니다.
- 프로비저닝된 클러스터 또는 Aurora Serverless v2 클러스터에 단일 DB 인스턴스만 포함되어 있거나 기본 인스턴스와 모든 리더 인스턴스가 동일한 AZ에 있는 경우, 다른 AZ에 하나 이상의 새 DB 인스턴스를 수동으로 생성해야 합니다.
- 클러스터에서 Aurora Serverless v1을(를) 사용하는 경우 Aurora은(는) 다른 AZ에 새 DB 인스턴스를 자동으로 생성합니다. 그러나 이 프로세스에는 호스트 교체가 수반되므로 장애 조치보다 시간이 더 오래 걸립니다.

> **Note**
>
> Amazon Aurora는 외부 MySQL 데이터베이스 또는 RDS MySQL DB 인스턴스의 복제도 지원합니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 (이진 로그 복제본) 간의 복제 단원을 참조하십시오.

Amazon RDS 프록시를 사용하는 고가용성

RDS 프록시를 사용하면 복잡한 장애 처리 코드를 작성할 필요 없이 데이터베이스 장애를 투명하게 견딜 수 있는 애플리케이션을 구축할 수 있습니다. 프록시는 애플리케이션 연결을 유지하면서 트래픽을 새 데이터베이스 인스턴스로 자동 라우팅합니다. 또한 도메인 이름 시스템(DNS) 캐시를 우회하여 Aurora 다중 AZ 데이터베이스의 장애 조치 시간을 최대 66% 단축합니다. 자세한 내용은 Amazon RDS Proxy for Aurora 단원을 참조하십시오.

Amazon Aurora를 사용한 복제

Aurora에서는 몇 가지 복제 옵션이 제공됩니다. 각 Aurora DB 클러스터에는 동일한 클러스터에 있는 여러 DB 인스턴스 간의 기본 제공 복제본이 있습니다. Aurora 클러스터를 소스 또는 타겟으로 하여 복제를 설정할 수도 있습니다. 클러스터에서 데이터를 복제하거나 Aurora 클러스터 외부로 복제할 때 Aurora 글로벌 데이터베이스와 같은 기본 제공 기능이나 MySQL 또는 PostgreSQL DB 엔진의 기존 복제 메커니즘 중에서 선택할 수 있습니다. 필요에 따라 고가용성, 편의성 및 성능을 적절하게 조합한 옵션을 선택할 수 있습니다. 다음 단원에서는 각 기법을 선택하는 방법과 시기를 설명합니다.

주제

Aurora 복제본

Aurora 프로비저닝된 DB 클러스터에서 두 번째, 세 번째 등의 DB 인스턴스를 생성하면 Aurora는 라이터 DB 인스턴스에서 다른 모든 DB 인스턴스로의 복제를 자동으로 설정합니다. 이러한 다른 DB 인스턴스는 읽기 전용이며 Aurora 복제본이라고 합니다. 또한 클러스터 내에서 라이터 DB 인스턴스와 리더 DB 인스턴스를 결합할 수 있는 방법에 대해 논의할 때, 이 인스턴스를 리더 인스턴스라고 합니다.

Aurora 복제본에는 두 가지 주요 목적이 있습니다. 애플리케이션에 대한 읽기 작업을 확장하기 위해 쿼리를 실행할 수 있습니다. 일반적으로 클러스터의 읽기 장치 엔드포인트에 연결하여 이 작업을 수행합니다. 이렇게 하면 Aurora 는 읽기 전용 연결에 대한 로드를 클러스터에 있는 여러 Aurora 복제본에 분산할 수 있습니다. 또한 Aurora 복제본은 가용성을 높이는 데 도움이 됩니다. 클러스터의 작성기 인스턴스를 사용할 수 없게 되면 Aurora는 읽기 장치 인스턴스 중 하나를 자동으로 승격하여 새 작성기로 사용합니다.

Aurora DB 클러스터는 최대 15 개의 Aurora 복제본을 포함할 수 있습니다. Aurora 복제본을 AWS 리전 내 DB 클러스터에 포함된 가용 영역에 배포할 수 있습니다.

DB 클러스터의 데이터에는 클러스터 내 DB 인스턴스와 무관하게 자체 고가용성 및 안정성 기능이 있습니다. Aurora 스토리지 기능에 익숙하지 않은 경우 Amazon Aurora 스토리지 개요 단원을 참조하십시오. DB 클러스터 볼륨은 물리적으로 DB 클러스터에 대한 데이터의 여러 복사본으로 구성됩니다. DB 클러스터의 기본 인스턴스 및 Aurora 복제본에는 클러스터 볼륨 데이터가 단 하나의 논리 볼륨으로 표시됩니다.

따라서 모든 Aurora 복제본은 쿼리 결과에 대해 동일한 데이터를 반환하며 복제본 지연이 최소화됩니다. 이러한 지연은 대개 기본 인스턴스에서 업데이트를 쓴 후 100밀리초를 넘지 않습니다. 데이터베이스 변경률에 따라 달라집니다. 즉, 데이터베이스의 쓰기 연산이 많은 기간에는 복제본 지연 시간이 증가할 수 있습니다.

> **Note**
>
> 다음 Aurora PostgreSQL 버전에서 라이터 DB 인스턴스와의 통신이 60초 이상 끊기면 Aurora 복제본이 자동으로 다시 시작됩니다.

- 14.6 이하 버전
- 13.9 이하 버전
- 12.13 이하 버전
- 모든 Aurora PostgreSQL 11 버전

읽기 가용성 기능을 사용하면 Aurora 복제본이 자동으로 다시 시작되지 않습니다. 읽기 가용성 기능과 해당하는 버전에 대한 자세한 내용은 Aurora 복제본의 읽기 가용성 향상 섹션을 참조하세요.

Aurora 복제본은 클러스터 볼륨의 읽기 연산에 전적으로 사용되므로 읽기 조정에 유용합니다. 쓰기 연산은 기본 인스턴스에서 관리합니다. 클러스터 볼륨은 DB 클러스터의 모든 DB 인스턴스가 공유하기 때문에 각 Aurora 복제본의 데이터 사본을 추가로 복제할 필요가 거의 없습니다.

가용성을 높이려면 Aurora 복제본을 장애 조치 대상으로 사용할 수 있습니다. 즉 기본 인스턴스에 장애가 발생하면 Aurora 복제본이 기본 인스턴스로 승격됩니다. 기본 인스턴스에 대한 읽기/쓰기 요청이 예외로 인해 장애가 발생하는 동안에는 시스템이 짧게 중단됩니다.

장애 조치를 통해 Aurora 복제본을 승격시키는 것이 기본 인스턴스를 재생성하는 것보다 훨씬 빠릅니다. Aurora DB 클러스터에 Aurora 복제본이 포함되어 있지 않으면 DB 인스턴스가 장애에서 복구되는 동안 DB 클러스터를 사용할 수 없습니다.

장애 조치가 발생하는 경우 DB 엔진 버전에 따라 일부 Aurora 복제본이 재부팅될 수 있습니다. 예를 들어, Aurora MySQL에서 Aurora는 장애 조치 중에 라이터 DB 인스턴스와 장애 조치 대상만 다시 시작합니다. 서로 다른 Aurora DB 엔진 버전의 재부팅 동작에 대한 자세한 내용은 Amazon Aurora DB 클러스터 또는 Amazon Aurora DB 인스턴스 재부팅 섹션을 참조하세요. 재부팅 또는 장애 조치 시 페이지 캐시에 어떤 일이 발생하는지에 대한 자세한 내용은 유지 가능한 페이지 캐시 섹션을 참조하세요.

고가용성 시나리오에서는 Aurora 복제본을 1개 이상 생성하는 것이 좋습니다. 이때 복제본은 DB 인스턴스 클래스가 기본 인스턴스의 클래스와 동일해야 하고, 가용 영역이 Aurora DB 클러스터의 가용 영역과 달라야 합니다. 장애 조치 대상인 Aurora 복제본에 대한 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.

암호화되지 않은 Aurora DB 클러스터의 암호화된 Aurora 복제본을 생성할 수 없습니다. 암호화된 Aurora DB 클러스터의 암호화되지 않은 Aurora 복제본을 생성할 수 없습니다.

> **Tip**
>
> Aurora 클러스터 내 Aurora 복제본을 유일한 복제 형식으로 사용하여 데이터의 고가용성을 유지할 수 있습니다. 기본 제공 Aurora 복제본을 다른 유형의 복제본과 결합할 수도 있습니다. 이렇게 하면 데이터의 고가용성과 지리적 분포를 한층 더 높일 수 있습니다.

Aurora 복제본 생성 방법에 대한 자세한 내용은 DB 클러스터에 Aurora 복제본 추가 단원을 참조하십시오.

Aurora MySQL를 사용한 복제

Aurora 복제본 외에도 다음과 같이 Aurora MySQL로 복제에 사용할 수 있는 옵션이 있습니다:

- 여러 다른 AWS 리전에 있는 Aurora MySQL DB 클러스터
  - Aurora 전역 데이터베이스를 사용하여 여러 리전에서 데이터를 복제할 수 있습니다. 자세한 내용은 Aurora Global Database를 사용한 AWS 리전 간 고가용성 섹션을 참조하세요.
  - MySQL 이진 로그(binlog) 복제를 사용하여 다른 AWS 리전에 Aurora MySQL DB 클러스터의 Aurora 읽기 전용 복제본을 생성할 수 있습니다. 각 클러스터는 서로 다른 각 리전에 최대 5개의 읽기 전용 복제본을 생성할 수 있습니다.
- MySQL 이진 로그(binlog) 복제를 사용하여 동일한 리전에 있는 Aurora MySQL DB 클러스터 2개
- RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 데이터의 소스인 RDS for MySQL DB 인스턴스와 Aurora MySQL DB 클러스터를 설정합니다. 일반적으로 이 방법은 진행 중인 복제보다는 Aurora MySQL으로의 마이그레이션에 사용됩니다.

Aurora MySQL을 사용한 복제에 대한 자세한 내용은 Amazon Aurora MySQL을 사용한 복제 단원을 참조하십시오.

Aurora PostgreSQL를 사용한 복제

Aurora 복제본 외에도 다음과 같이 Aurora PostgreSQL로 복제에 사용할 수 있는 옵션이 있습니다:

- Aurora 글로벌 데이터베이스를 사용하여 한 리전의 Aurora 프라이머리 DB 클러스터와 서로 다른 리전의 읽기 전용 보조 DB 클러스터 최대 10개를 보유합니다. 즉, Aurora PostgreSQL은 교차 리전 Aurora 복제본을 지원하지 않습니다. 하지만 Aurora Global Database를 사용하여 Aurora PostgreSQL DB 클러스터의 읽기 기능을 둘 이상의 AWS 리전으로 확장하고 가용성 목표를 달성할 수 있습니다. 자세한 내용은 Amazon Aurora Global Database 사용 섹션을 참조하세요.

- PostgreSQL의 논리적 복제 기능을 사용하여 2개의 Aurora PostgreSQL DB 클러스터는 동일한 리전에 위치합니다.
- RDS for PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 데이터의 소스인 RDS for PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터를 설정합니다. 일반적으로 이 방법은 진행 중인 복제보다는 Aurora PostgreSQL으로의 마이그레이션에 사용됩니다.

Aurora PostgreSQL을 사용한 복제에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 사용한 복제 단원을 참조하십시오.

Aurora에 대한 DB 인스턴스 결제

Amazon Aurora 클러스터의 Amazon RDS 프로비저닝된 인스턴스는 다음 구성 요소를 기준으로 청구됩니다.

- DB 인스턴스 시간(시간별) – DB 인스턴스의 DB 인스턴스 클래스를 기준으로 합니다(예: db.t2.small 또는 db.m4.large). 요금은 시간 단위로 고시되지만, 청구서는 초 단위로 계산되고 시간을 10진수 형식으로 표시합니다. RDS 사용량은 1초 단위로 청구되며 최소 청구 시간은 10분입니다. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.
- 스토리지(월별 GiB별) – DB 인스턴스에 프로비저닝한 스토리지 용량입니다. 해당 월에 프로비저닝된 스토리지 용량의 크기를 조정하는 경우 청구서 금액은 비례 할당으로 계산됩니다. 자세한 내용은 Amazon Aurora 스토리지 단원을 참조하십시오.
- 입출력(I/O) 요청(1백만 요청별) – 결제 주기에 요청한 총 스토리지 I/O 요청 수입니다. Aurora Standard DB 클러스터 구성에만 해당됩니다.

  Amazon Aurora I/O 결제에 대한 자세한 정보는 Amazon Aurora DB 클러스터의 스토리지 구성 섹션을 참조하세요.
- 백업 스토리지(월별 GiB별) – 백업 스토리지는 자동화된 데이터베이스 백업 및 생성한 활성 데이터베이스 스냅샷과 연결된 스토리지입니다. 백업 보존 기간을 연장하거나 추가 데이터베이스 스냅샷을 찍으면 데이터베이스가 사용하는 백업 스토리지가 증가합니다. 초 단위 결제는 백업 스토리지(GB-월 단위로 측정됨)에는 적용되지 않습니다.

  자세한 내용은 Amazon Aurora DB 클러스터 백업 및 복구 섹션을 참조하세요.
- 데이터 전송(GB별) - DB 인스턴스와 인터넷 및 기타 AWS 리전 간의 양방향 데이터 전송입니다. 유용한 예는 AWS 관리형 데이터베이스의 데이터 전송 비용 살펴보기 AWS 블로그 게시물을 참조하세요.

Amazon RDS는 사용자가 요구 사항에 따라 비용을 최적화할 수 있도록 다음과 같은 구입 옵션을 제공합니다.

- 온디맨드 인스턴스 – 사용하는 DB 인스턴스 시간에 대해 시간별로 지불합니다. 요금은 시간 단위로 고시되지만, 청구서는 초 단위로 계산되고 시간을 10진수 형식으로 표시합니다. RDS 사용량은 이제 1초 단위로 청구되며 최소 청구 시간은 10분입니다.
- 예약 인스턴스 – 1년 또는 3년 기간으로 DB 인스턴스를 예약하고 온디맨드 DB 인스턴스 요금에 비해 상당한 할인을 받습니다. 예약 인스턴스 사용 시 여러 인스턴스를 1시간 내에 시작, 삭제, 사용 또는 종료하고 모든 인스턴스에 대해 예약 인스턴스 혜택을 적용받을 수 있습니다.
- Aurora Serverless v2 - Aurora Serverless v2에서는 결제 단위가 DB 인스턴스 시간이 아닌 Aurora 용량 단위(ACU) 시간인 온디맨드 용량을 제공합니다. Aurora Serverless v2 용량은 데이터베이스의 로드에 따라 사용자가 지정한 범위 내에서 증가하거나 감소합니다. 모든 용량이 Aurora Serverless v2인 클러스터를 구성할 수 있습니다. 또는 Aurora Serverless v2 및 온디맨드 또는 예약 프로비저닝된 인스턴스의 조합으로 클러스터를 구성할 수 있습니다. Aurora Serverless v2 ACU 작동 방식에 대한 자세한 내용은 Aurora Serverless v2 작동 방식 단원을 참조하세요.
- Aurora PostgreSQL Limitless Database - Aurora PostgreSQL Limitless Database는 단일 DB 인스턴스의 쓰기 처리량 및 스토리지 한도를 초과하여 확장되는 자동 수평적 스케일링 기능입니다. Limitless Database는 단일 데이터베이스로 운영하는 용이성은 유지하면서 여러 Aurora 라이터 인스턴스에 워크로드를 분산합니다. Limitless Database는 DB 샤드 그룹에서 요금 청구 단위가 Aurora 용량 단위(ACU) 시간인 온디맨드 용량을 제공합니다. Limitless Database ACU 작동 방식에 대한 자세한 내용은 Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 만들기 섹션을 참조하세요.

Aurora요금에 대한 자세한 내용은 Aurora 요금 페이지를 참조하십시오.

주제

- Aurora용 온디맨드 DB 인스턴스
- Amazon Aurora용 예약 DB 인스턴스

Aurora용 온디맨드 DB 인스턴스

Amazon RDS 온디맨드 DB 인스턴스는 DB 인스턴스 클래스를 기반으로 청구됩니다(예: db.t3.small 또는 db.m5.large). Amazon RDS 요금에 대한 자세한 정보는 Amazon RDS 제품 페이지를 참조하십시오.

DB 인스턴스가 사용 가능하면 즉시 DB 인스턴스에 대한 청구가 시작됩니다. 요금은 시간 단위로 고시되지만, 청구서는 초 단위로 계산되고 시간을 10진수 형식으로 표시합니다. Amazon RDS 사용량은 1초 단위로 청구되며 최소 청구 시간은 10분입니다. 컴퓨팅 또는 스토리지 용량 조정과 같은 청구 대상 구성 변경의 경우 최소 시간인 10분에 대해 요금이 부과됩니다. DB 인스턴스를 삭제하거나 DB 인스턴스에 장애가 발생하여 DB 인스턴스가 종료될 때까지 청구가 계속됩니다.

DB 인스턴스 요금이 더 이상 부과되지 않도록 하려면 추가 DB 인스턴스 시간에 대해 청구되지 않도록 인스턴스를 중지하거나 삭제해야 합니다. 청구되는 DB 인스턴스 상태에 대한 자세한 정보는 Aurora 클러스터에서 DB 인스턴스 상태 보기 단원을 참조하십시오.

중지된 DB 인스턴스

DB 인스턴스가 중지되는 동안 프로비저닝된 IOPS를 포함하여 프로비저닝된 스토리지에 대해 요금이 부과됩니다. 지정된 보존 기간 내의 수동 스냅샷 및 자동 백업용 스토리지를 포함하여 백업 스토리지에 대한 요금도 부과됩니다. DB 인스턴스 시간에 대해서는 요금이 부과되지 않습니다

다중 AZ DB 인스턴스

다중 AZ 설정은 다른 가용 영역에서 동기식 대기 복제본을 자동으로 프로비저닝하고 유지 관리하여 데이터 내구성과 가용성을 향상시킵니다. 추가 리소스와 가용성 증가로 인해 다중 AZ 배포는 단일 AZ 배포보다 요금이 높으며 추가 대기 인스턴스 및 관련 리소스 때문에 약 2배의 비용이 발생할 수 있습니다.

다중 AZ 요금에 대한 다음과 같은 중요한 세부 정보를 고려하세요.

- 컴퓨팅 비용: 기본 인스턴스와 대기 인스턴스 모두에 대해 DB 인스턴스 시간당 요금이 청구됩니다.
- 스토리지 비용: 기본 인스턴스와 대기 인스턴스 모두에 프로비저닝된 스토리지에 대해 월별 GB당 요금이 부과됩니다.
- 데이터 전송 비용: 기본 인스턴스와 대기 인스턴스 간의 복제는 비용에 포함되지만 사용량에 따라 기타 데이터 전송 요금이 적용될 수 있습니다.

특정 사용 사례 및 AWS 리전을 기반으로 월별 비용을 정확하게 추정하려면 AWS Pricing Calculator를 사용하세요. 이 도구를 사용하면 구성 세부 정보를 입력하고 포괄적인 비용 분석을 제공받을 수 있습니다.

Note

요금은 변경될 수 있습니다. 최신 정보는 Amazon RDS 요금 페이지를 참조하세요.

Amazon Aurora용 예약 DB 인스턴스

예약 DB 인스턴스를 사용하면 1년 또는 3년 단위로 DB 인스턴스를 예약할 수 있습니다. 예약 DB 인스턴스는 온디맨드 DB 인스턴스 요금과 비교하여 대폭 할인된 요금을 제공합니다. 예약 DB 인스턴스는 물리적 인스턴스가 아니고 오히려 계정에서 온디맨드 DB 인스턴스를 사용할 때 적용되는 결제 할인에 가깝습니다. 예약 DB 인스턴스의 할인 요금은 인스턴스 유형 및 AWS 리전에 따라 결정됩니다.

DB 인스턴스를 예약하기 위한 프로세스는 다음과 같습니다. 먼저 구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 확인합니다. 그런 다음 DB 인스턴스 예약 상품을 구매하고 마지막으로 기존에 예약되어 있는 DB 인스턴스에 대한 정보를 확인합니다.

예약 DB 인스턴스 구매 및 예약 DB 인스턴스 요금 청구서 보기에 대한 자세한 내용은 다음 섹션을 참조하세요.

- Amazon Aurora용 예약 DB 인스턴스 구매
- Amazon Aurora의 예약 DB 인스턴스에 대한 청구서 보기

예약 DB 인스턴스 개요

Amazon RDS에서 예약 DB 인스턴스를 구매할 때는 예약 DB 인스턴스의 기간 동안 특정 DB 인스턴스 유형에 대해 할인 요금을 이용하는 약정을 구매하는 것입니다. Amazon RDS 예약 DB 인스턴스를 사용하려면 온디맨드 인스턴스와 똑같은 방법으로 새로운 DB 인스턴스를 생성해야 합니다.

새롭게 생성한 DB 인스턴스는 다음에 대해 예약 DB 인스턴스의 사양과 일치해야 합니다.

- AWS 리전
- DB 엔진(DB 엔진의 버전 번호가 일치할 필요는 없습니다.)
- DB 인스턴스 유형

새로운 DB 인스턴스의 사양이 계정의 기존 DB 예약 인스턴스와 일치하면 예약 DB 인스턴스에 제공되는 할인 요금이 청구됩니다. 그렇지 않으면 DB 인스턴스에 대해 온디맨드 요금이 청구됩니다.

예약형 DB 인스턴스로 사용 중인 DB 인스턴스를 수정할 수 있습니다. 변경 사항이 예약 DB 인스턴스의 사양 내에 있는 경우 수정된 DB 인스턴스에도 할인의 일부 또는 전부가 적용됩니다. 인스턴스 클래스를 변경하는 경우와 같이 변경 사항이 사양을 벗어나면 할인이 더 이상 적용되지 않습니다. 자세한 내용은 유연한 크기의 예약 DB 인스턴스 단원을 참조하십시오.

주제

- [제공 유형]
- [Aurora DB 클러스터 구성 유연성]
- [유연한 크기의 예약 DB 인스턴스]
- [Aurora 예약 DB 인스턴스 결제 예제]
- [예약 DB 인스턴스 삭제]

요금을 포함하여 예약 DB 인스턴스에 대한 자세한 내용은 Amazon RDS 예약 인스턴스를 참조하십시오.

제공 유형

예약 DB 인스턴스는 세 가지 유형(No Upfront, Partial Upfront 및 All Upfront)으로 제공되며 예상되는 사용률에 따라 Amazon RDS 비용을 최적화할 수 있습니다.

> **Note**
> 모든 RDS 인스턴스 클래스가 모든 예약 인스턴스 오퍼링 유형을 지원하는 것은 아닙니다. 예를 들어 일부 인스턴스 클래스는 선결제 없음 옵션을 제공하지 않을 수 있습니다. 이용 가능 여부를 확인하려면 AWS Management Console에서 예약 인스턴스 오퍼링을 검토하거나 `describe-reserved-db-instances-offerings` AWS CLI 명령을 사용합니다.

선수금 없음

선결제 없이 예약 DB 인스턴스에 액세스할 수 있는 옵션입니다. 비선결제 예약 DB 인스턴스는 사용 기간 동안 사용량에 상관없이 할인된 시간당 요금이 청구되며, 선결제가 필요하지 않습니다. 이 옵션은 1년 예약만 가능합니다.

부분 선결제

예약 DB 인스턴스 사용비의 일부를 먼저 결제해야 하는 옵션입니다. 결제하지 않은 시간에 대해서는 사용 기간 동안 사용량에 상관없이 할인된 시간당 요금이 청구됩니다. 이 옵션은 이전 Heavy 사용률 옵션을 대신합니다.

전체 선결제

약관이 시작되는 시점에서 모든 금액을 결제하고 사용 기간 동안 추가 비용 없이 무제한으로 사용할 수 있습니다.

통합 결제를 사용하는 경우, 결제의 편의를 위해 조직 내 모든 계정은 하나의 계정으로 취급됩니다. 즉 조직 내 모든 계정은 다른 계정에서 구입한 예약 DB 인스턴스에 대해 시간당 비용 혜택을 받을 수 있습니다. 통합 결제에 대한 자세한 내용은 AWS Billing and Cost Management 사용 설명서에서 Amazon RDS 예약 DB 인스턴스를 참조하세요.

Aurora DB 클러스터 구성 유연성

Aurora 예약 DB 인스턴스는 다음 두 DB 클러스터 구성 모두에서 사용할 수 있습니다.

- Aurora I/O-Optimized – 읽기 및 쓰기 I/O 작업에 대한 추가 비용 없이 DB 클러스터의 사용량 및 스토리지에 대한 비용만 지불하면 됩니다.
- Aurora Standard – DB 클러스터의 사용량 및 스토리지 외에도 I/O 작업에 대해 요청 1백만 건당의 표준 요금이 부과됩니다.

Aurora는 이러한 구성 간의 가격 차이를 자동으로 반영합니다. Aurora I/O-Optimized는 Aurora Standard보다 시간당 정규화된 유닛을 30% 더 소모합니다.

Aurora 클러스터 스토리지 구성에 대한 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 섹션을 참조하세요. Aurora 클러스터 스토리지 구성에 대한 요금 정보는 Amazon Aurora 요금을 참조하세요.

유연한 크기의 예약 DB 인스턴스

예약 DB 인스턴스를 구매할 때 지정해야 하는 것 중 하나가 인스턴스 클래스(db.r5.large 등)입니다. DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

이미 DB 인스턴스가 있지만 용량을 확장해야 하는 경우에는 예약 DB 인스턴스가 확장된 DB 인스턴스에 자동으로 적용됩니다. 다시 말해서 예약 DB 인스턴스는 모든 DB 인스턴스 클래스 크기에 자동으로 적용됩니다. 동일한 AWS 리전 및 데이터베이스 엔진에서 유연한 크기의 예약 DB 인스턴스를 DB 인스턴스에 사용할 수 있습니다. 유연한 크기의 예약 DB 인스턴스는 해당 인스턴스 클래스 유형에서만 확장할 수 있습니다. 예를 들어 db.r5.large의 예약 DB 인스턴스는 db.r5.large에 적용할 수 있지만, db.r6g.large에는 적용할 수 없습니다. db.r5와 db.r6g는 다른 인스턴스 클래스 유형이기 때문입니다.

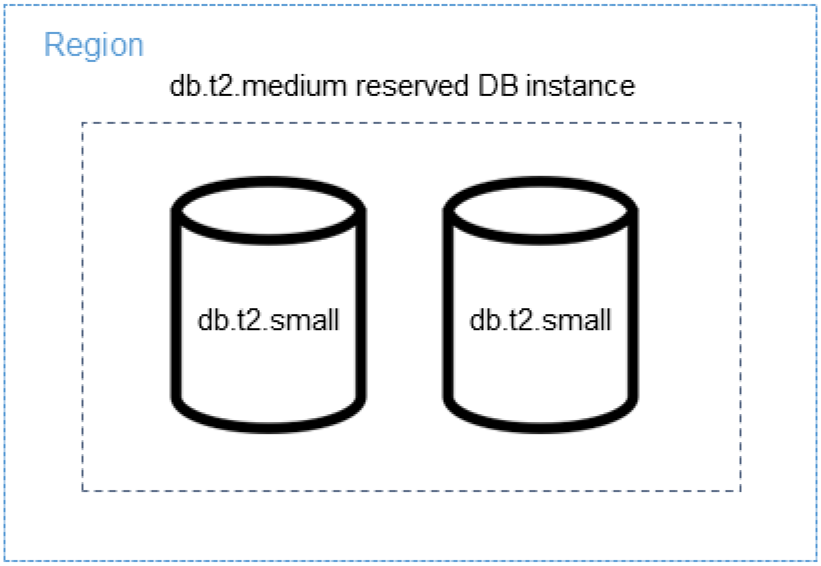
유연한 크기의 예약 DB 인스턴스는 다음 Aurora 데이터베이스 엔진에서 제공됩니다.

- Aurora MySQL
- Aurora PostgreSQL

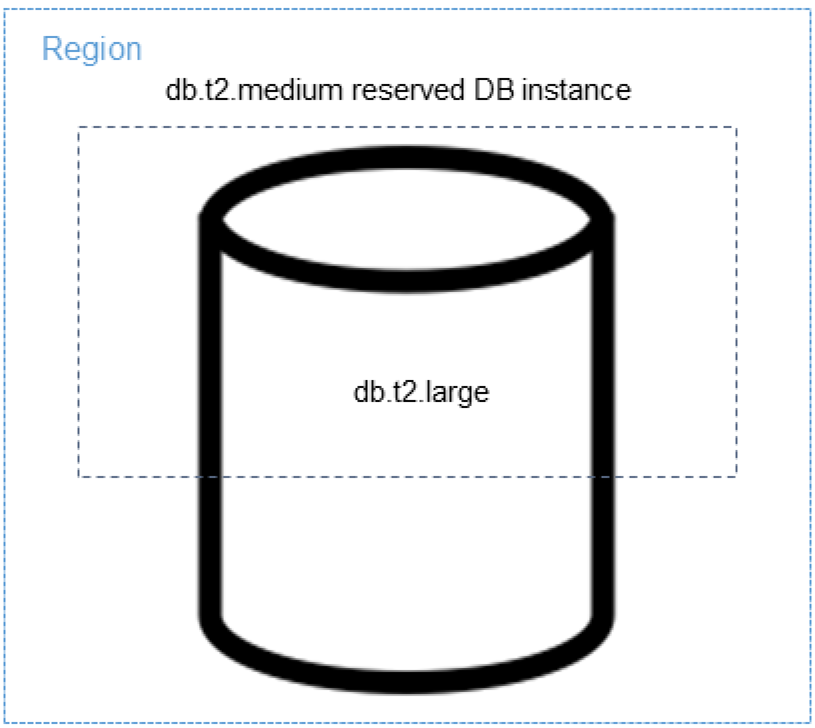
예약 DB 인스턴스의 크기에 따른 사용량은 시간당 정규화된 유닛을 사용하여 비교할 수 있습니다. 예를 들어 db.r3.large DB 인스턴스 2개일 때 사용량의 유닛 1개는 db.r3.small 1개일 때 사용량의 시간당 정규화된 유닛 8개와 같습니다. 다음 표는 각 DB 인스턴스 크기에 따른 시간당 정규화된 유닛의 수를 나타낸 것입니다.

인스턴스 크기	DB 인스턴스 1개에 대한 시간당 정규화된 유닛, Aurora Standard	DB 인스턴스 1개에 대한 시간당 정규화된 유닛, Aurora I/O-Optimized	DB 인스턴스 3개(라이터 및 리더 2개)에 대한 시간당 정규화된 유닛, Aurora Standard	DB 인스턴스 3개(라이터 및 리더 2개)에 대한 시간당 정규화된 유닛, Aurora I/O-Optimized
small	1	1.3	3	3.9
medium	2	2.6	6	7.8
large	4	5.2	12	15.6
xlarge	8	10.4	24	31.2
2xlarge	16	20.8	48	62.4
4xlarge	32	41.6	96	124.8
8xlarge	64	83.2	192	249.6
12xlarge	96	124.8	288	374.4
16xlarge	128	166.4	384	499.2
24xlarge	192	249.6	576	748.8
32xlarge	256	332.8	768	998.4

예약 DB 인스턴스로 `db.t2.medium`을 1개 구매하고, 동일한 AWS 리전의 계정에서 `db.t2.small` DB 인스턴스를 2개 실행하는 경우를 예로 들어 보겠습니다. 이 경우 결제 혜택은 두 인스턴스에 100% 적용됩니다.



또는 동일한 AWS 리전의 계정에서 실행 중인 `db.t2.large` 인스턴스 1개가 있는 경우 결제 혜택은 DB 인스턴스 사용량의 50%에 적용됩니다.



> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

Aurora 예약 DB 인스턴스 결제 예제

다음 예제는 Aurora Standard 및 Aurora I/O-Optimized DB 클러스터 구성을 모두 사용하는 Aurora DB 클러스터에 대한 예약 DB 인스턴스 요금을 보여 줍니다.

Aurora Standard 사용 예제

예약된 DB 인스턴스 요금은 스토리지, 백업 및 I/O와 관련된 비용의 할인이 제공되지 않습니다. 다음 예제에서는 예약된 DB 인스턴스의 월 총비용을 보여 줍니다.

- 미국 동부(버지니아 북부)의 Aurora MySQL 예약 단일 AZ db.r5.large DB 인스턴스 클래스, 시간당 요금 0.19 USD(월 138.70 USD)
- 월 기준 GiB당 0.10 USD(이 예에서는 월 45.60 USD로 가정)의 Aurora 스토리지
- 월 기준 요청 100만 개당 0.20 USD(이 예에서는 월 20 USD로 가정)의 Aurora I/O
- 월 기준 GiB당 0.021 USD(이 예에서는 월 30 USD로 가정)의 Aurora 백업 스토리지

예약 DB 인스턴스에 이러한 옵션을 모두 추가할 경우(138.70 USD + 45.60 USD + 20 USD + 30 USD), 월 총 요금은 234.30 USD입니다.

예약 DB 인스턴스 대신 온디맨드 DB 인스턴스를 사용하기로 선택하는 경우, 미국 동부(버지니아 북부)의 Aurora MySQL 단일 AZ db.r5.large DB 인스턴스 클래스 요금은 시간당 0.29 USD(월 217.50 USD)입니다. 따라서 온디맨드 DB 인스턴스의 경우, 이러한 옵션을 모두 추가하면(217.50 USD + 45.60 USD + 20 USD + 30 USD), 월 총 요금은 313.10 USD입니다. 예약 DB 인스턴스를 사용하면 매달 약 79USD의 비용을 절감할 수 있습니다.

2개의 리더 인스턴스가 있는 Aurora Standard DB 클러스터를 사용하는 예제

Aurora DB 클러스터용 예약 인스턴스를 사용하려면 클러스터의 각 DB 인스턴스마다 예약 인스턴스를 하나씩 구매하면 됩니다.

첫 번째 예제를 확장하면 라이터 DB 인스턴스 1개와 Aurora 복제본 2개로 구성된 Aurora MySQL DB 클러스터가 생성되어 클러스터에는 총 3개의 DB 인스턴스가 있습니다. 두 개의 Aurora 복제본에는 추

가 스토리지 또는 백업 요금이 부과되지 않습니다. db.r5.large Aurora MySQL 예약 DB 인스턴스를 3개 구매하는 경우 요금은 234.30 USD(라이터 DB 인스턴스) + 2 * (Aurora 복제본당 I/O 138.70 USD + 20 USD)이므로 월 총요금은 551.70 USD가 됩니다.

라이터 DB 인스턴스 1개와 Aurora 복제본 2개로 구성된 Aurora MySQL DB 클러스터에 해당하는 온디맨드 요금은 313.10 USD + 2 * (217.50 USD + 인스턴스당 20 USD I/O)로, 매월 총 788.10 USD입니다. 예약 DB 인스턴스를 사용하면 매달 약 236.40 USD를 절감할 수 있습니다.

Aurora I/O-Optimized 사용 예제

Aurora I/O-Optimized로 기존 Aurora Standard 예약 DB 인스턴스를 재사용할 수 있습니다. Aurora I/O-Optimized의 예약 인스턴스 할인 혜택을 최대한 활용하려면 현재 예약 인스턴스와 비슷한 예약 인스턴스를 30% 추가로 구매하면 됩니다.

다음 표에서는 Aurora I/O-Optimized 사용 시 추가 예약 인스턴스를 추정하는 방법의 예를 보여 줍니다. 필요한 예약 인스턴스가 분수 값일 경우 예약 인스턴스에서 제공되는 크기 유연성을 활용하여 정수 값에 이를 수 있습니다. 이 예제에서 '현재'는 현재 보유하고 있는 Aurora Standard 예약 인스턴스를 나타냅니다. 추가 예약 인스턴스는 Aurora I/O-Optimized 사용 시 현재 예약 인스턴스 할인을 유지하기 위해 구매해야 하는 Aurora Standard 예약 인스턴스의 수입니다.

DB 인스턴스 클래스	현재 Aurora Standard 예약 인스턴스	Aurora I/O-Optimized에 필요한 예약 인스턴스	필요한 추가 예약 인스턴스	크기 유연성 사용 시 필요한 추가 예약 인스턴스
db.r6g.large	10	10 * 1.3 = 13	3 * db.r6g.large	3 * db.r6g.large
db.r6g.4xlarge	20	20 * 1.3 = 26	6 * db.r6g.4xlarge	6 * db.r6g.4xlarge
db.r6g.12xlarge	5	5 * 1.3 = 6.5	1.5 * db.r6g.12xlarge	db.r6g.12xlarge, r6g.4xlarge 및 r6g.2xlarge 하나씩 (0.5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge)

DB 인스턴스 클래스	현재 Aurora Standard 예약 인스턴스	Aurora I/O-Optimized에 필요한 예약 인스턴스	필요한 추가 예약 인스턴스	크기 유연성 사용 시 필요한 추가 예약 인스턴스
db.r6i.24xlarge	15	15 * 1.3 = 19.5	4.5 * db.r6i.24 xlarge	4 * db.r6i.24 xlarge + 1 * db.r6i.12xlarge (0.5 * db.r6i.24 xlarge = 1 * db.r6i.12xlarge)

2개의 리더 인스턴스가 있는 Aurora I/O-Optimized DB 클러스터를 사용하는 예제

라이터 DB 인스턴스 1개와 Aurora 복제본 2개로 구성된 Aurora MySQL DB 클러스터가 생성되어 클러스터에는 총 3개의 DB 인스턴스가 있습니다. 이들은 Aurora I/O-Optimized DB 클러스터 구성을 사용합니다. 이 클러스터에 예약 DB 인스턴스를 사용하려면 동일한 DB 인스턴스 클래스의 예약 DB 인스턴스 4개를 구매해야 합니다. Aurora I/O-Optimized를 사용하는 3개의 DB 인스턴스는 시간당 정규화된 유닛 3.9개를 소비하는 반면, Aurora Standard를 사용하는 DB 인스턴스 3개는 시간당 정규화된 유닛 3개를 소비합니다. 하지만 각 DB 인스턴스의 월별 I/O 비용은 절감됩니다.

> **Note**
>
> 이 예의 요금은 샘플 요금이며 실제 요금과 다를 수 있습니다. Amazon Aurora 요금에 대한 자세한 내용은 Amazon Aurora 요금 페이지를 참조하세요.

예약 DB 인스턴스 삭제

예약 DB 인스턴스에 대한 약정 기간은 1년 또는 3년입니다. 예약 DB 인스턴스는 취소할 수 없습니다. 하지만 예약 DB 인스턴스 할인이 적용되는 DB 인스턴스를 삭제할 수는 있습니다. 예약 DB 인스턴스 할인이 적용되는 DB 인스턴스의 삭제 프로세스는 다른 DB 인스턴스를 삭제할 때와 동일합니다.

리소스 사용 여부에 관계없이 선결제 비용이 청구됩니다.

예약 DB 인스턴스 할인이 적용되는 DB 인스턴스를 삭제할 경우에는 다르지만 서로 사양이 호환되는 DB 인스턴스를 시작할 수 있습니다. 이 경우 예약 기간(1년 또는 3년)에 요금 할인을 계속 받을 수 있습니다.

Amazon Aurora용 예약 DB 인스턴스 구매

AWS Management Console, AWS CLI 및 RDS API를 사용하여 예약 DB 인스턴스 작업을 수행할 수 있습니다.

콘솔

예약 DB 인스턴스에 대한 작업은 AWS Management Console에서 다음 절차에 따라 진행할 수 있습니다.

사용 가능한 예약 DB 인스턴스 상품에 대한 요금과 정보를 가져오려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 예약 인스턴스를 선택합니다.
3. [Purchase Reserved DB Instance]를 선택합니다.
4. 제품 설명에서 DB 엔진과 라이선스 유형을 선택합니다.
5. DB 인스턴스 클래스에서 DB 인스턴스 클래스를 선택합니다.
6. 배포 옵션에서 단일 AZ 배포 또는 다중 AZ 배포를 사용할지 선택합니다.

   > **Note**
   >
   > 예약된 Amazon Aurora 인스턴스에서는 배포 옵션을 항상 단일 AZ DB 인스턴스로 설정합니다. 하지만 Aurora DB 클러스터를 생성할 때 기본 배포 옵션은 다른 AZ에 Aurora 복제본 또는 리더 생성(다중 AZ)입니다.
   > Aurora 복제본을 포함하여 사용하려는 각 인스턴스에 대한 예약 DB 인스턴스를 구매해야 합니다. 따라서 Aurora에 다중 AZ 배포를 하려면 예약 DB 인스턴스를 추가로 구매해야 합니다.

7. 기간에서 DB 인스턴스를 예약할 기간을 선택합니다.
8. 제공 유형에서 해당 제공 유형을 선택합니다.

   상품 유형을 선택하면 요금 정보가 표시됩니다.

   > **Important**
   >
   > 취소를 선택하면 예약 DB 인스턴스를 구입하지 않으며 요금이 발생하지 않습니다.

구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 확인하였으면 이제 정보를 사용하여 다음 절차에 따라 상품을 구매할 수 있습니다.

예약 DB 인스턴스를 구입하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 예약 인스턴스를 선택합니다.
3. Purchase reserved DB instance(예약 DB 인스턴스 구매)를 선택합니다.
4. 제품 설명에서 DB 엔진과 라이선스 유형을 선택합니다.
5. DB 인스턴스 클래스에서 DB 인스턴스 클래스를 선택합니다.
6. 다중 AZ 배포에서 단일 AZ 또는 다중 AZ DB 인스턴스 배포를 사용할지 여부를 선택합니다.

   > **Note**
   >
   > 예약된 Amazon Aurora 인스턴스에서는 배포 옵션을 항상 단일 AZ DB 인스턴스로 설정합니다. 예약 DB 인스턴스에서 Amazon Aurora DB 클러스터를 생성할 때 DB 클러스터가 자동으로 다중 AZ로 생성됩니다. Aurora 복제본을 포함하여 사용하려는 각 DB 인스턴스에 대한 예약 DB 인스턴스를 구매해야 합니다.

7. [Term]에서 DB 인스턴스를 예약할 기간을 선택합니다.
8. 제공 유형에서 해당 제공 유형을 선택합니다.

   오퍼링 유형을 선택하면 요금 정보가 표시됩니다.
9. (선택 사항) - 예약 DB 인스턴스를 조회할 수 있도록 구매하는 예약 인스턴스에 자체 식별자를 할당할 수 있습니다. [Reserved Id]에 자신이 예약한 DB 인스턴스 식별자를 입력하면 됩니다.
10. 제출을 선택합니다.

    예약 DB 인스턴스를 구매하면 예약 인스턴스 목록에 표시됩니다.

예약한 DB 인스턴스를 구매한 후에는 다음 절차에 따라 예약한 DB 인스턴스에 대한 정보를 가져올 수 있습니다.

AWS 계정에 대한 예약 DB 인스턴스 관련 정보를 가져오려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 예약 인스턴스를 선택합니다.

   현재 계정에서 예약한 DB 인스턴스가 나타납니다. 특정 예약 DB 인스턴스의 세부 정보를 보려면 목록에서 해당 인스턴스를 선택합니다. 그러면 콘솔 아래쪽의 세부 정보 창에 인스턴스에 대한 세부 정보가 표시됩니다.

AWS CLI

예약 DB 인스턴스에 대한 작업은 다음 예제와 같이 AWS CLI를 사용하여 진행할 수 있습니다.

Example 사용 가능한 예약 DB 인스턴스 오퍼링 가져오기

구매 가능한 DB 인스턴스 상품에 대한 정보를 가져오려면 AWS CLI 명령 describe-reserved-db-instances-offerings를 호출합니다.

```
aws rds describe-reserved-db-instances-offerings
```

이 호출은 다음과 비슷한 출력을 반환합니다.

```

OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 확인하였으면 이제 정보를 사용하여 오퍼링을 구매할 수 있습니다.

예약 DB 인스턴스를 구매하려면 다음 파라미터와 함께 AWS CLI 명령 purchase-reserved-db-instances-offering을 사용합니다.

- `--reserved-db-instances-offering-id` – 구매하려는 오퍼링의 ID입니다. 위의 예제를 참조하여 상품 ID를 가져옵니다.
- `--reserved-db-instance-id` – 구매하는 예약 DB 인스턴스에 자체 식별자를 할당하여 관리할 수 있습니다.

Example 예약 DB 인스턴스 구매

다음은 ID가 *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*인 DB 인스턴스 예약 상품을 구매하고 식별자로 *MyReservation*을 할당하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

Windows의 경우:

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

이 명령은 다음과 비슷한 출력을 반환합니다.

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State             Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql         Partial  Upfront
```

예약 DB 인스턴스를 구매한 후에는 예약 DB 인스턴스에 대한 정보를 가져올 수 있습니다.

AWS 계정에서 예약 DB 인스턴스에 대한 정보를 가져오려면 다음 예제와 같이 AWS CLI 명령 `describe-reserved-db-instances`를 호출합니다.

Example 예약 DB 인스턴스 가져오기

```
aws rds describe-reserved-db-instances
```

이 명령은 다음과 비슷한 출력을 반환합니다.

```
RESERVATION  ReservationId     Class          Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

RDS API를 사용하여 예약 DB 인스턴스 작업을 수행할 수 있습니다.

- 구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 가져오려면 Amazon RDS API 작업 `DescribeReservedDBInstancesOfferings`를 호출합니다.
- 구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 확인하였으면 이제 정보를 사용하여 오퍼링을 구매할 수 있습니다. 다음 파라미터로 `PurchaseReservedDBInstancesOffering` RDS API 작업을 호출합니다.
  - `--reserved-db-instances-offering-id` – 구매하려는 오퍼링의 ID입니다.
  - `--reserved-db-instance-id` – 구매하는 예약 DB 인스턴스에 자체 식별자를 할당하여 관리할 수 있습니다.
- 예약 DB 인스턴스를 구매한 후에는 예약 DB 인스턴스에 대한 정보를 가져올 수 있습니다. RDS API 작업 `DescribeReservedDBInstances`를 호출합니다.

Amazon Aurora의 예약 DB 인스턴스에 대한 청구서 보기

예약 DB 인스턴스에 대한 결제는 AWS Management Console의 결제 대시보드(Billing Dashboard)에서 확인할 수 있습니다.

예약 DB 인스턴스 결제 확인

1. AWS Management Console에 로그인합니다.
2. 오른쪽 상단의 계정 메뉴(account menu)에서 결제 대시보드(Billing Dashboard)를 선택합니다.
3. 대시보드 오른쪽 상단의 청구 세부 정보(Service Charge)를 선택합니다.
4. AWS 서비스 요금(Service Charges)에서 관계형 데이터베이스 서비스(Relational Database Service)를 확장합니다.
5. 미국 서부(오레곤)과 같이 예약 DB 인스턴스가 있는 AWS 리전을 확장합니다.

   예약 DB 인스턴스와 현재 월의 시간당 요금은 ****** ##**용 Amazon Relational Database Service 예약 인스턴스에서 볼 수 있습니다.

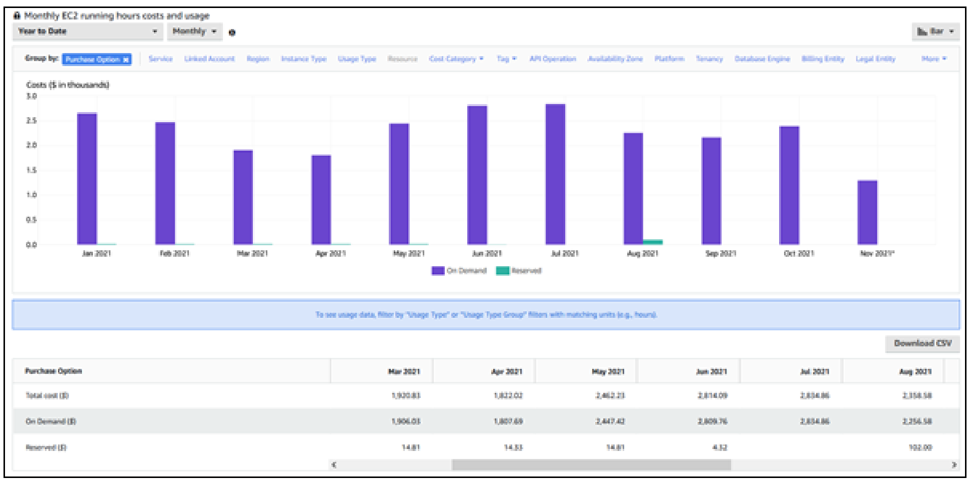
Amazon Relational Database Service for MySQL Community Edition Reserved Instances		$0.00
MySQL, db.t3.micro reserved instance applied, db.t3.micro instance used	395.000 Hrs	$0.00
USD 0.0 hourly fee per MySQL, db.t3.micro instance	720.000 Hrs	$0.00

이 예시의 예약 DB 인스턴스는 전체 선결제 구매되었으므로 시간당 요금이 부과되지 않습니다.

6. 예약 인스턴스(Reserved Instances) 제목 옆의 비용 탐색기(Cost Explorer)(막대 그래프) 아이콘을 선택합니다.

   Cost Explorer는 월별 EC2 운영 시간 비용 및 사용량(Monthly EC2 running hours costs and usage) 그래프를 보여줍니다.
7. 그래프 오른쪽에 있는 사용 유형 그룹(Usage Type Group) 필터를 해제합니다.
8. 사용 비용을 검사할 기간 및 시간 단위를 선택합니다.

   다음 예에서는 해당 연도의 온디맨드 및 예약 DB 인스턴스의 사용 비용을 월별로 보여줍니다.



2021년 1월부터 6월까지 예약 DB 인스턴스 비용은 부분 선결제 인스턴스에 대한 월별 요금이며, 2021년 8월 요금은 전체 선결제 인스턴스에 대한 일회성 요금입니다.

부분 선결제 인스턴스에 대한 예약 인스턴스 할인은 2021년 6월에 만료되었지만 DB 인스턴스는 삭제되지 않았습니다. 만료일 이후에는 온디맨드 요금으로 간단히 청구되었습니다.

Amazon Aurora 환경 설정

Amazon Aurora를 처음 사용한다면 먼저 다음 태스크를 완료해야 합니다.

주제

- AWS 계정에 등록
- 관리자 액세스 권한이 있는 사용자 생성
- 프로그래밍 방식 액세스 권한 부여
- 요구 사항 결정
- 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다.

AWS 계정가 이미 있고 Aurora 요구 사항을 알고 있으며 IAM 및 VPC 보안 그룹에 모두 기본값을 사용하려는 경우에는 Amazon Aurora 시작하기 섹션으로 건너뜁니다.

AWS 계정에 등록

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정 루트 사용자에 가입하면 AWS 계정 루트 사용자가 만들어집니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS는 가입 절차 완료된 후 사용자에게 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자에 보안 조치를 한 다음, AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화(콘솔)를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center 설정을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 기본 IAM Identity Center 디렉터리로 사용자 액세스 구성을 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS Management Console 외부에서 AWS와 상호 작용하려면 프로그래밍 방식의 액세스가 필요합니다. 프로그래밍 방식으로 액세스를 부여하는 방법은 AWS에 액세스하는 사용자 유형에 따라 다릅니다.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 AWS IAM Identity Center을 사용하도록 AWS CLI 구성을 참조하세요. • AWS SDK, 도구,AWS API에 대해서는 AWS SDK 및 도구 참조 가이드에서 IAM Identity Center 인증을 참조하세요.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	IAM 사용자 설명서의 AWS 리소스와 함께 임시 자격 증명 사용에 나와 있는 지침을 따르세요.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 IAM 사용자 자격 증명을 사용한 인증을 참조하세요. • AWS SDK와 도구는 AWS SDK 및 도구 참조 가이드에서 장기 자격 증명을 사용한 인증을 참조하세요. • AWS API는 IAM 사용자 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

요구 사항 결정

Aurora의 기본 빌딩 블록은 DB 클러스터입니다. 하나의 DB 클러스터에 한 개 이상의 DB 인스턴스가 속할 수 있습니다. DB 클러스터는 클러스터 엔드포인트라고 하는 네트워크 주소를 제공합니다. 애플리케이션은 DB 클러스터에서 생성된 데이터베이스에 액세스할 때마다 해당 DB 클러스터가 할당한 클러스터 엔드포인트에 연결됩니다. 또한 DB 클러스터를 생성할 때 지정하는 정보에 따라서 메모리, 데이터베이스 엔진 및 버전, 네트워크 구성, 보안, 유지 관리 기간 등의 구성 요소가 제어됩니다.

DB 클러스터와 보안 그룹을 생성하기 전에 DB 클러스터 및 네트워크 요구 사항을 알아야 합니다. 고려해야 할 몇 가지 중요 사항은 다음과 같습니다:

- 리소스 요구 사항 – 애플리케이션 또는 서비스의 메모리 및 프로세서 요구 사항은 무엇입니까? DB 클러스터를 생성할 때는 사용할 DB 인스턴스 클래스를 결정하면서 이러한 설정을 사용하게 됩니다. DB 인스턴스 클래스에 대한 사양은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하세요.
- VPC, 서브넷 및 보안 그룹 – DB 클러스터는 Virtual Private Cloud(VPC)에 위치합니다. DB 클러스터에 연결하려면 보안 그룹 규칙을 구성해야 합니다. 다음은 각 VPC 옵션의 규칙을 설명한 목록입니다.

- 기본 VPC - AWS 계정에 AWS 리전의 기본 VPC가 있는 경우 해당 VPC를 구성하여 DB 클러스터를 지원할 수 있습니다. DB 클러스터 생성 시 기본 VPC를 지정할 경우 다음과 같이 실행해야 합니다.
  - 애플리케이션 또는 서비스에서 Aurora DB 인스턴스로의 연결 권한을 부여하는 VPC 보안 그룹을 생성해야 합니다. VPC 콘솔의 [보안 그룹(Security Group)] 옵션 또는 AWS CLI를 사용하여 VPC 보안 그룹을 생성합니다. 자세한 정보는 3단계: VPC 보안 그룹 만들기 섹션을 참조하세요.
  - 기본 DB 서브넷 그룹을 지정해야 합니다. AWS 리전에서 처음 DB 클러스터를 생성하는 경우에는 Amazon RDS가 DB 클러스터 생성과 함께 기본 DB 서브넷 그룹을 생성합니다.
- 사용자 정의 VPC — DB 클러스터 생성 시 사용자 정의 VPC를 지정할 경우 다음과 같이 실행해야 합니다.
  - 애플리케이션 또는 서비스에서 Aurora DB 인스턴스로의 연결 권한을 부여하는 VPC 보안 그룹을 생성해야 합니다. VPC 콘솔의 [보안 그룹(Security Group)] 옵션 또는 AWS CLI를 사용하여 VPC 보안 그룹을 생성합니다. 자세한 정보는 3단계: VPC 보안 그룹 만들기 섹션을 참조하세요.
  - VPC가 DB 클러스터를 호스팅하려면 별도의 가용 영역에서 각각 최소 2개 이상씩 서브넷을 구성하는 등 특정 요구 사항을 충족해야 합니다. 자세한 정보는 Amazon VPC 및 Amazon Aurora 섹션을 참조하세요.
  - DB 서브넷 그룹을 지정하여 DB 클러스터에서 VPC를 사용할 서브넷을 정의해야 합니다. 자세한 정보는 VPC에서 DB 클러스터를 사용한 작업의 DB 서브넷 그룹 단원을 참조하세요.
- 높은 가용성: 장애 조치 지원이 필요합니까 Aurora에서 다중 AZ 배포는 기본 인스턴스와 Aurora 복제본을 생성합니다. 장애 조치 지원을 위해 기본 인스턴스와 Aurora 복제본을 서로 다른 가용 영역에 두도록 구성할 수 있습니다. 고가용성을 유지하기 위해 프로덕션 워크로드에는 다중 AZ 배포를 권장합니다. 개발 및 테스트 목적으로는 비 다중 AZ 배포를 사용할 수 있습니다. 자세한 내용은 Amazon Aurora의 고가용성 섹션을 참조하세요.
- IAM 정책: AWS 계정에 Amazon RDS 작업을 수행하는 데 필요한 권한을 부여하는 정책이 있습니까? IAM 자격 증명을 사용하여 AWS에 연결하는 경우 IAM 계정에는 Amazon RDS 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책이 있어야 합니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.
- 개방 포트: 데이터베이스가 어떤 TCP/IP 포트에서 수신 대기합니까? 일부 기업에서는 방화벽이 데이터베이스 엔진의 기본 포트 연결을 차단하는 경우도 있습니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 새로운 DB 클러스터에 다른 포트를 선택해야 합니다. 단, 생성된 DB 클러스터가 지정 포트에서 수신 대기할 경우 해당 DB 클러스터를 수정하여 포트를 변경할 수 있습니다.

- AWS 리전: 데이터베이스를 구성하려고 하는 AWS 리전은 어디입니까? 애플리케이션이나 웹 서비스에 가깝게 데이터베이스를 구성하면 네트워크 지연 시간을 줄일 수 있습니다. 자세한 내용은 리전 및 가용 영역 섹션을 참조하세요.

보안 그룹과 DB 클러스터 생성에 필요한 정보를 확인하였으면 다음 단계로 진행합니다.

보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다.

DB 클러스터는 VPC에 생성됩니다. 보안 그룹은 VPC에서 실행되는 DB 클러스터에 대한 액세스를 제공합니다. 이들은 연결된 DB 클러스터에 대한 방화벽 역할을 하여 클러스터 수준에서 인바운드 트래픽과 아웃바운드 트래픽을 모두 제어합니다. 기본적으로 DB 클러스터는 DB 클러스터에 대한 액세스를 방지하는 방화벽 및 기본 보안 그룹과 함께 생성됩니다. 따라서 DB 클러스터에 연결할 수 있는 규칙을 보안 그룹에 추가해야 합니다. 이전 단계에서 파악한 네트워크 및 구성 정보를 사용하여 DB 클러스터에 액세스할 수 있는 규칙을 만듭니다.

예를 들어 애플리케이션이 VPC 내에 생성한 DB 클러스터의 데이터베이스에 액세스할 경우 데이터베이스에 액세스하는 데 필요한 포트 범위와 IP 주소를 지정한 사용자 정의 TCP 규칙을 추가해야 합니다. Amazon EC2 인스턴스에 애플리케이션이 있는 경우 Amazon EC2 인스턴스에 대해 설정한 보안 그룹을 사용할 수 있습니다.

DB 클러스터를 만들 때 Amazon EC2 인스턴스와 DB 클러스터 간의 연결을 구성할 수 있습니다. 자세한 내용은 EC2 인스턴스와의 자동 네트워크 연결 구성 섹션을 참조하세요.

> **Tip**
>
> DB 클러스터를 생성할 때 Amazon EC2 인스턴스와 DB 클러스터 간의 네트워크 연결을 자동으로 설정할 수 있습니다. 자세한 내용은 EC2 인스턴스와의 자동 네트워크 연결 구성 섹션을 참조하세요.

Amazon Lightsail의 리소스를 DB 클러스터에 연결하는 방법에 대한 자세한 내용은 Connect Lightsail resources to AWS 서비스 using VPC peering을 참조하세요.

Aurora에서 사용할 VPC 생성에 대한 자세한 내용은 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 섹션을 참조하세요. DB 인스턴스 액세스의 일반적인 시나리오에 대한 자세한 내용은 VPC에서 DB 클러스터에 액세스하는 시나리오 섹션을 참조하세요.

VPC 보안 그룹의 생성 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc에서 Amazon VPC 콘솔을 엽니다.

> **Note**
>
> RDS 콘솔이 아니라 VPC 콘솔을 사용해야 합니다.

2. AWS Management Console의 오른쪽 상단에서 VPC 보안 그룹 및 DB 클러스터를 생성할 AWS 리전을 선택합니다. 해당 AWS 리전의 Amazon VPC 리소스 목록에 1개 이상의 VPC와 몇 개의 서브넷이 표시되어야 합니다. 그렇지 않으면 해당 AWS 리전에 기본 VPC가 없는 것입니다.
3. 탐색 창에서 보안 그룹을 선택합니다.
4. 보안 그룹 생성을 선택합니다.

   [보안 그룹 생성(Create security group)] 페이지가 나타납니다.
5. [기본 세부 정보(Basic details)]에서 [보안 그룹 이름(Security group name)]과 [설명(Description)]을 입력합니다. [VPC]에서 DB 클러스터를 생성할 VPC를 선택합니다.
6. [인바운드 규칙(Inbound rules)]에서 [규칙 추가(Add rule)]를 선택합니다.

   a. 유형에 대해 사용자 지정 TCP를 선택합니다.

   b. [포트 범위(Port range)]에 DB 클러스터에 사용할 포트 값을 입력합니다.

   c. [소스(Source)]에서 DB 클러스터에 액세스할 IP 주소 범위(CIDR 값)를 입력하거나 보안 그룹 이름을 선택합니다. [내 IP(My IP)]를 선택하면 브라우저에서 감지된 IP 주소에서 DB 클러스터에 액세스할 수 있습니다.
7. IP 주소 또는 다른 포트 범위를 추가해야 하는 경우 [규칙 추가(Add rule)]를 선택하고 규칙에 대한 정보를 입력합니다.
8. (선택 사항) [아웃바운드 규칙(Outbound rules)]에서 아웃바운드 트래픽에 대한 규칙을 추가합니다. 기본적으로 모든 아웃바운드 트래픽이 허용됩니다.
9. 보안 그룹 생성을 선택합니다.

이제 방금 생성한 VPC 보안 그룹을 DB 클러스터 생성 시 보안 그룹으로 사용할 수 있습니다.

Note

기본 VPC를 사용하는 경우 VPC의 모든 서브넷을 포괄하는 기본 서브넷 그룹이 자동으로 생성됩니다. DB 클러스터를 생성할 때 기본 VPC를 선택하고 [DB 서브넷 그룹(DB Subnet Group)]의 기본값을 사용할 수 있습니다.

설정 요구 사항을 완료한 후에는 Amazon Aurora DB 클러스터 생성의 지침에 따라 요구 사항과 보안 그룹을 사용하여 DB 클러스터를 생성할 수 있습니다. 특정 DB 엔진을 사용하는 DB 클러스터를 생성하여 시작하는 방법에 대한 자세한 내용은 Amazon Aurora 시작하기 섹션을 참조하세요.

Amazon Aurora 시작하기

이 섹션에서는 Amazon RDS를 사용하여 DB 클러스터를 생성하고 Aurora에 연결하는 방법을 설명합니다.

다음 절차는 Aurora를 사용하여 시작하기 위한 기본 정보를 제공하는 튜토리얼입니다. 후속 섹션에서는 다양한 종류의 엔드포인트와 Aurora 클러스터를 확장/축소하는 방법을 포함하여 Aurora의 개념과 절차에 대한 보다 고급 정보를 제공합니다.

> **Important**
>
> DB 클러스터를 생성하거나 DB 클러스터에 연결하려면 먼저 Amazon Aurora 환경 설정 섹션의 태스크를 완료해야 합니다.

주제

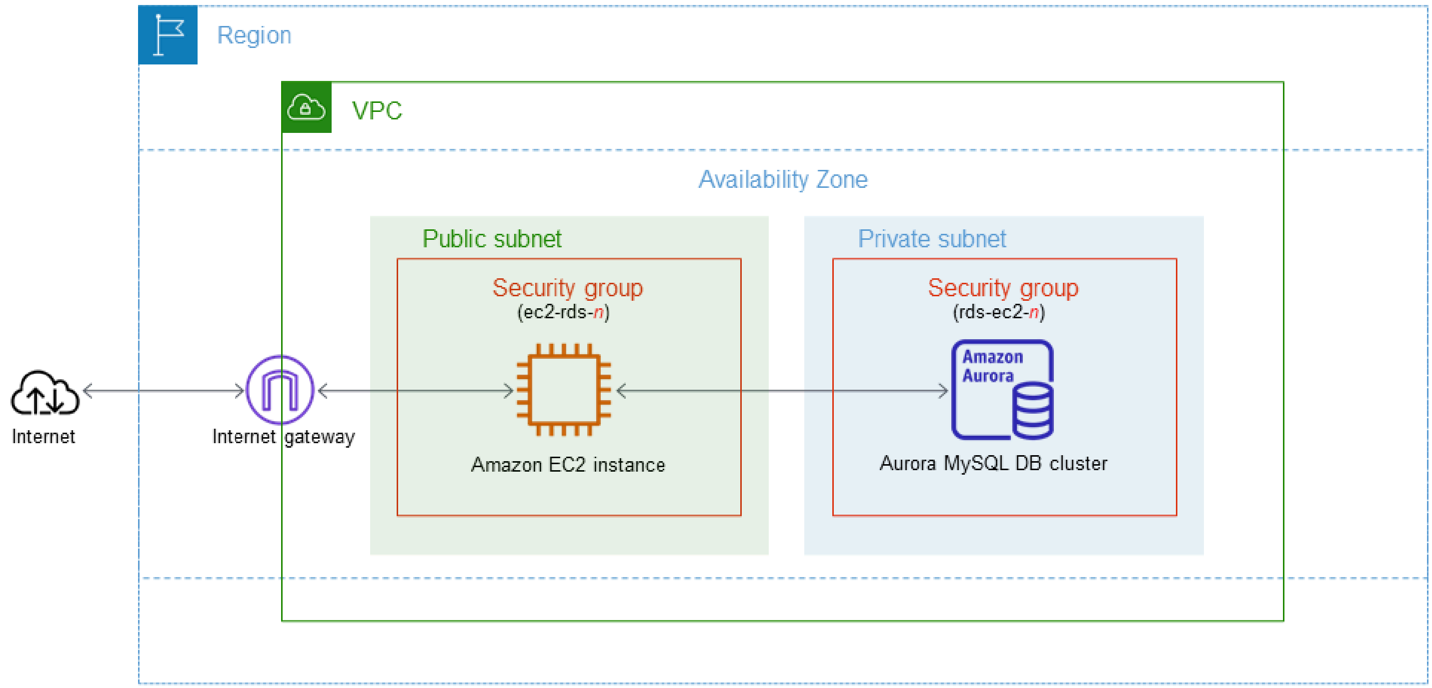
Aurora MySQL DB 클러스터 생성 및 연결

이 튜토리얼에서는 EC2 인스턴스 및 Aurora MySQL DB 클러스터를 생성합니다. 튜토리얼에서는 표준 MySQL 클라이언트를 사용하여 EC2 인스턴스에서 DB 클러스터에 액세스하는 방법을 보여줍니다. 이 튜토리얼에서는 모범 사례를 따라 Virtual Private Cloud(VPC)에서 프라이빗 DB 클러스터를 생성합니다. 대부분의 경우 EC2 인스턴스와 같이 동일한 VPC에 있는 다른 리소스는 DB 인스턴스에 액세스할 수 있지만 VPC 외부의 리소스는 DB 클러스터에 액세스할 수 없습니다.

자습서를 완료하면 VPC의 각 가용 영역에 퍼블릭 서브넷과 프라이빗 서브넷이 있을 것입니다. 한 가용 영역에서 EC2 인스턴스는 퍼블릭 서브넷에 있고 DB 인스턴스는 프라이빗 서브넷에 있습니다.

> **Important**
>
> AWS 계정 생성은 무료입니다. 그러나 이 튜토리얼을 완료하면 사용하는 AWS 리소스에 대한 비용이 발생할 수 있습니다. 자습서가 더 이상 필요하지 않은 경우 자습서를 완료한 후에 이러한 리소스를 삭제할 수 있습니다.

다음 다이어그램은 이 자습서를 완료했을 때 구성을 보여 줍니다.



이 자습서에서는 다음 방법 중 하나를 사용하여 리소스를 생성할 수 있습니다.

1. AWS Management Console 사용 - 1단계: EC2 인스턴스 생성 및 2단계: Aurora MySQL DB 클러스터 생성
2. 데이터베이스 인스턴스 및 EC2 인스턴스를 생성하는 데 AWS CloudFormation 사용 - (선택 사항) AWS CloudFormation을 사용하여 VPC, EC2 인스턴스 및 Aurora MySQL 클러스터 생성

첫 번째 방법은 간편 생성을 사용하여 AWS Management Console을 통해 프라이빗 Aurora MySQL DB 클러스터를 생성합니다. 여기에서는 DB 엔진 유형, DB 인스턴스 크기, DB 클러스터 식별자만 지정합니다. [간편 생성(Easy create)]은 다른 구성 옵션에서도 기본 설정을 사용합니다.

표준 생성을 대신 사용하는 경우에는 DB 클러스터를 생성할 때 더 많은 구성 옵션을 지정할 수 있습니다. 이러한 옵션에는 가용성, 보안, 백업 및 유지 관리에 대한 설정이 포함됩니다. 퍼블릭 DB 클러스터를 만들려면 표준 생성을 사용해야 합니다. 자세한 내용은 the section called “DB 클러스터 생성” 섹션을 참조하세요.

주제

- 사전 조건
- 1단계: EC2 인스턴스 생성
- 2단계: Aurora MySQL DB 클러스터 생성

사전 조건

시작하기 전에 다음 섹션에서 다음 단계를 완료하세요.

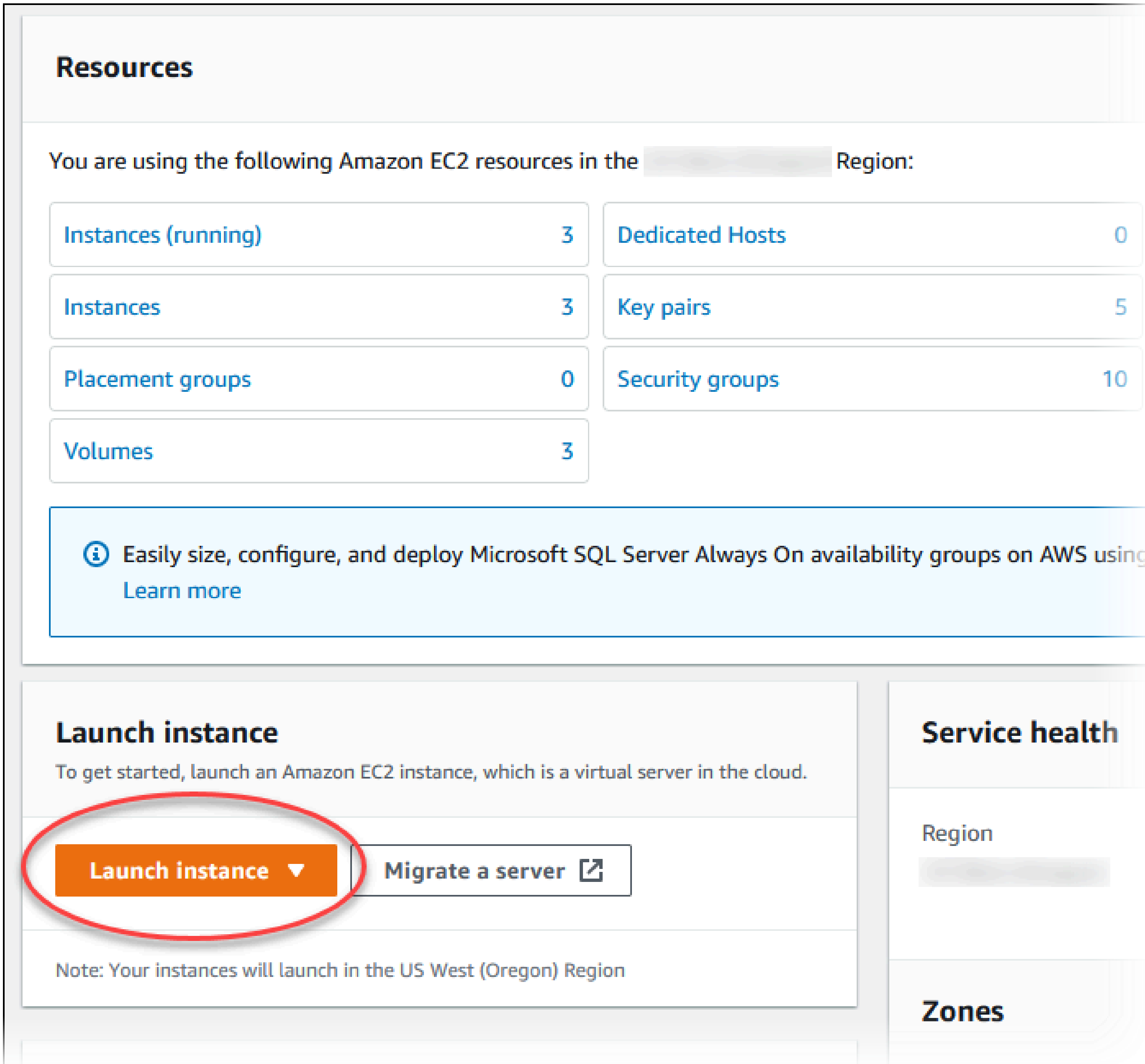
- AWS 계정에 등록
- 관리자 액세스 권한이 있는 사용자 생성

1단계: EC2 인스턴스 생성

데이터베이스에 연결하는 데 사용할 Amazon EC2 인스턴스를 생성합니다.

EC2 인스턴스를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 EC2 인스턴스를 생성하려는 AWS 리전을 선택합니다.
3. 다음 이미지에 나와 있는 것처럼 EC2 대시보드를 선택한 다음, 인스턴스 시작을 선택합니다.

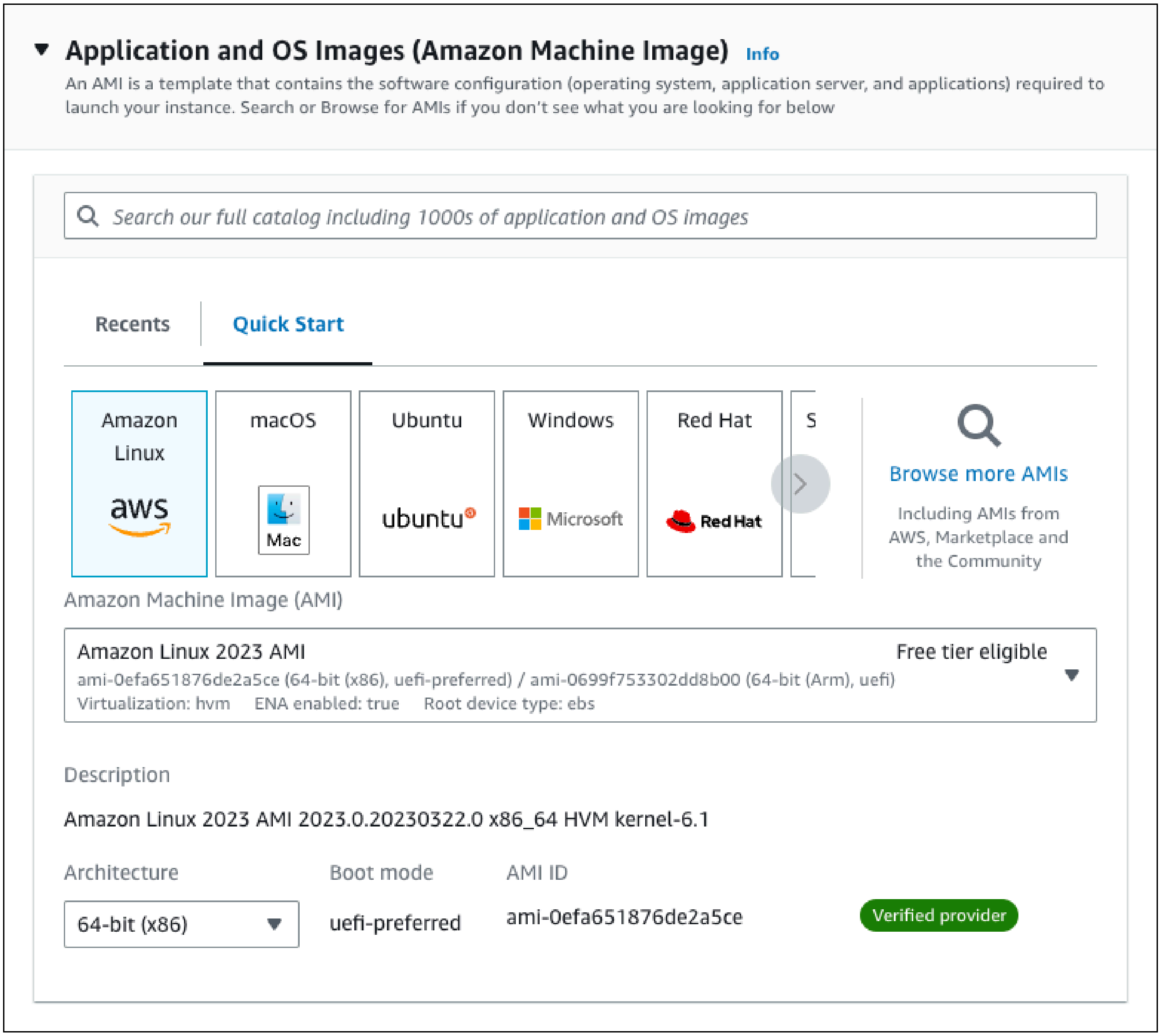


인스턴스 시작 페이지가 열립니다.

4. 인스턴스 시작 페이지에서 다음 설정을 선택합니다.

   a. Name and tags(이름 및 태그) 아래의 Name(이름)에 **ec2-database-connect**을 입력하세요.

   b. Application and OS Images (Amazon Machine Image)(애플리케이션 및 OS 이미지(Amazon Machine Image))에서 Amazon Linux를 선택한 다음 Amazon Linux 2023 AMI를 선택합니다. 다른 선택 항목에 대해서는 기본값을 그대로 유지합니다.



c. 인스턴스 유형에서 t2.micro를 선택합니다.

d. 키 페어(로그인)에서 기존 키 페어를 사용할 키 페어 이름을 선택합니다. Amazon EC2 인스턴스에 대한 새 키 페어를 생성하려면 새 키 페어 생성을 선택한 다음 키 페어 생성 창을 사용하여 생성합니다.

    새 키 페어 생성에 대한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 키 페어 생성을 참조하세요.

e. 네트워크 설정의 SSH 트래픽 허용에서 EC2 인스턴스에 대한 SSH 연결 소스를 선택합니다.

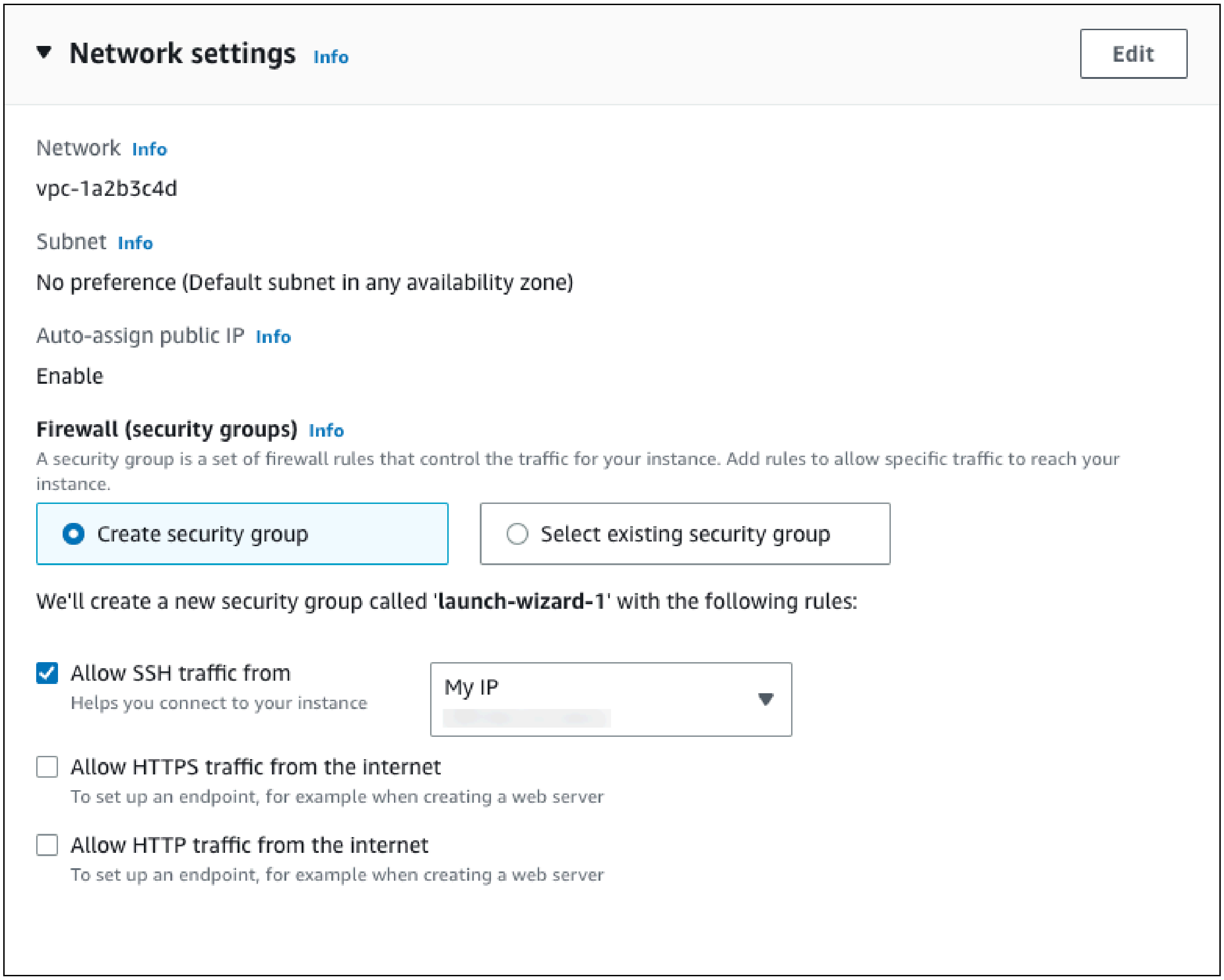
    표시된 IP 주소가 SSH 연결에 대해 올바른 경우 내 IP를 선택할 수 있습니다. 그렇지 않으면 SSH(Secure Shell)를 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를

결정할 수 있습니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다. IP 주소의 예는 192.0.2.1/32입니다.

대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 결정합니다.

> **Warning**
>
> SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 EC2 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 EC2 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

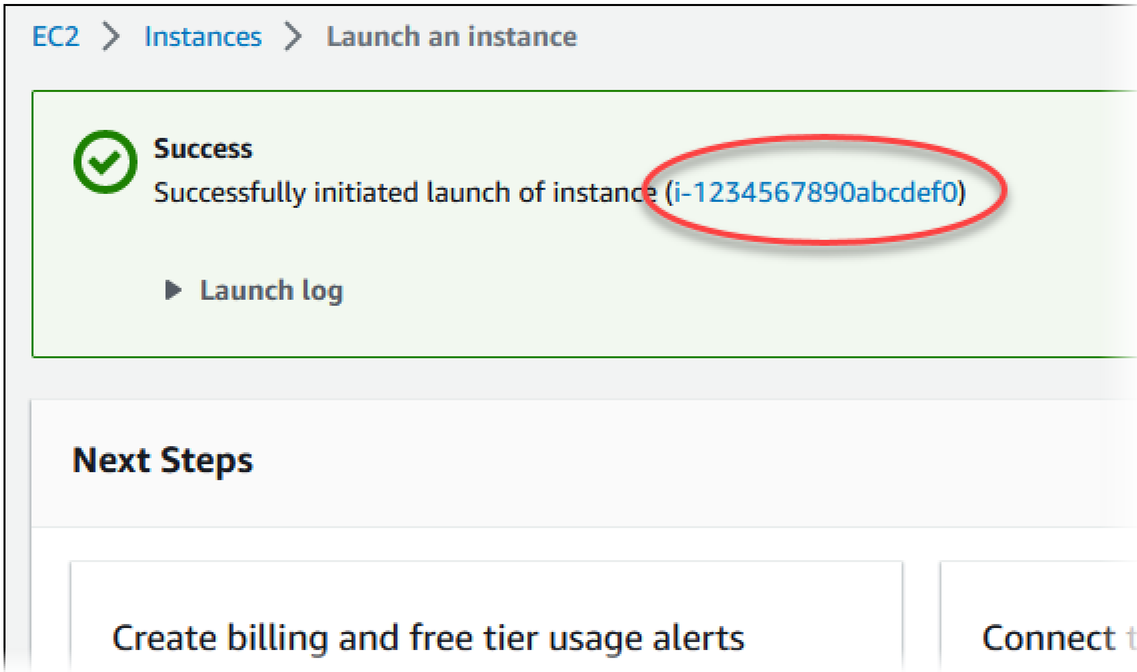
다음 이미지는 네트워크 설정 섹션의 예를 보여 줍니다.



f. 나머지 섹션에서 기본값은 그대로 둡니다.

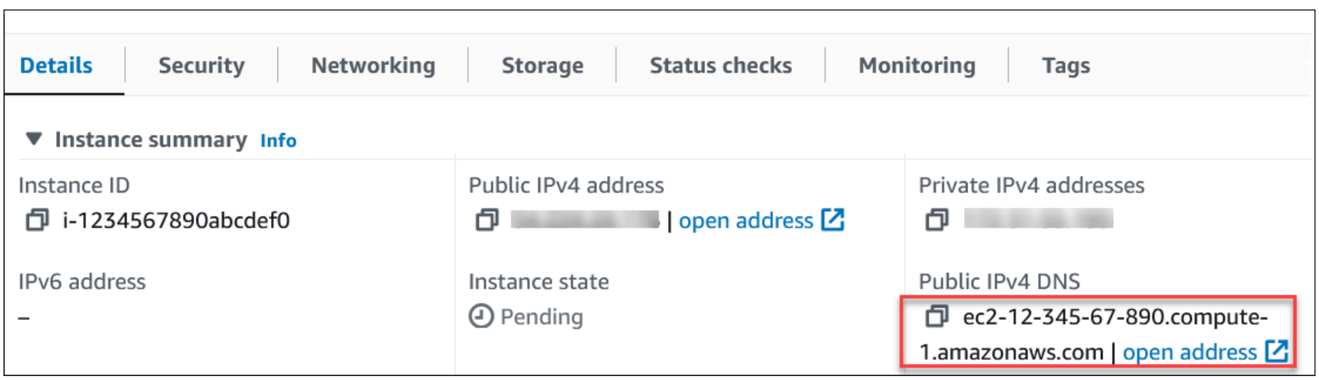
g. 요약 패널에서 EC2 인스턴스 구성 요약을 검토하고 준비가 되면 인스턴스 시작을 선택합니다.

5. 시작 상태 페이지에서, 새 EC2 인스턴스의 식별자(예: `i-1234567890abcdef0`)를 기록해 둡니다.

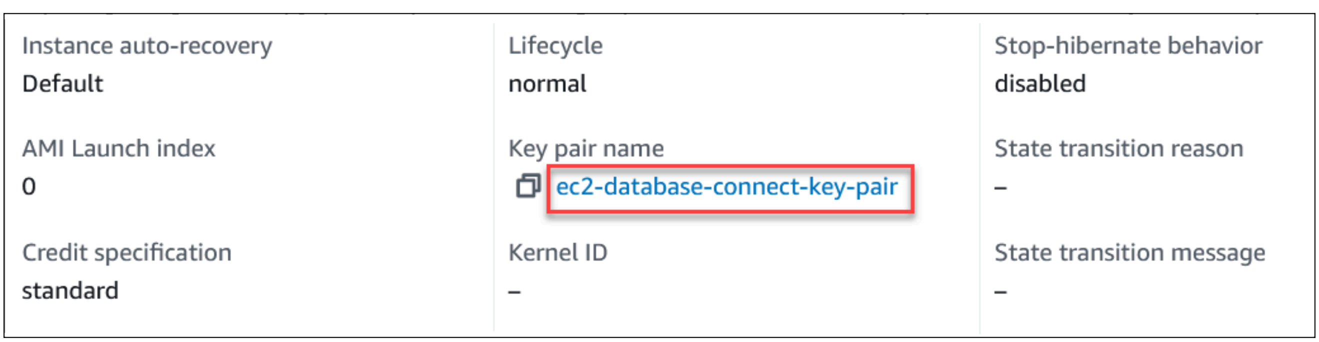


6. EC2 인스턴스 식별자를 선택하여 EC2 인스턴스 목록을 열고 EC2 인스턴스를 선택합니다.
7. 세부 정보 탭에서 SSH를 사용하여 연결할 때 필요한 다음 값을 기록해 둡니다.

   a. 인스턴스 요약에서 퍼블릭 IPv4 DNS의 값을 기록해 둡니다.



   b. 인스턴스 세부 정보에서 키 페어 이름의 값을 기록해 둡니다.



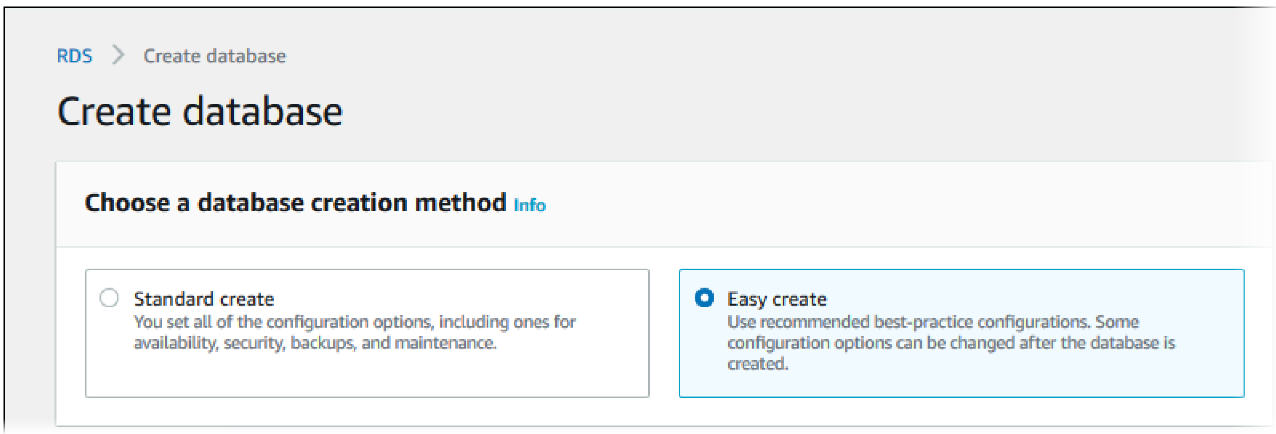
8. 계속하기 전에 EC2 인스턴스의 인스턴스 상태가 실행 중이 될 때까지 기다립니다.

2단계: Aurora MySQL DB 클러스터 생성

이 예시에서는 간편 생성을 사용하여 db.r6g.large DB 인스턴스 클래스로 Aurora MySQL DB 클러스터를 생성합니다.

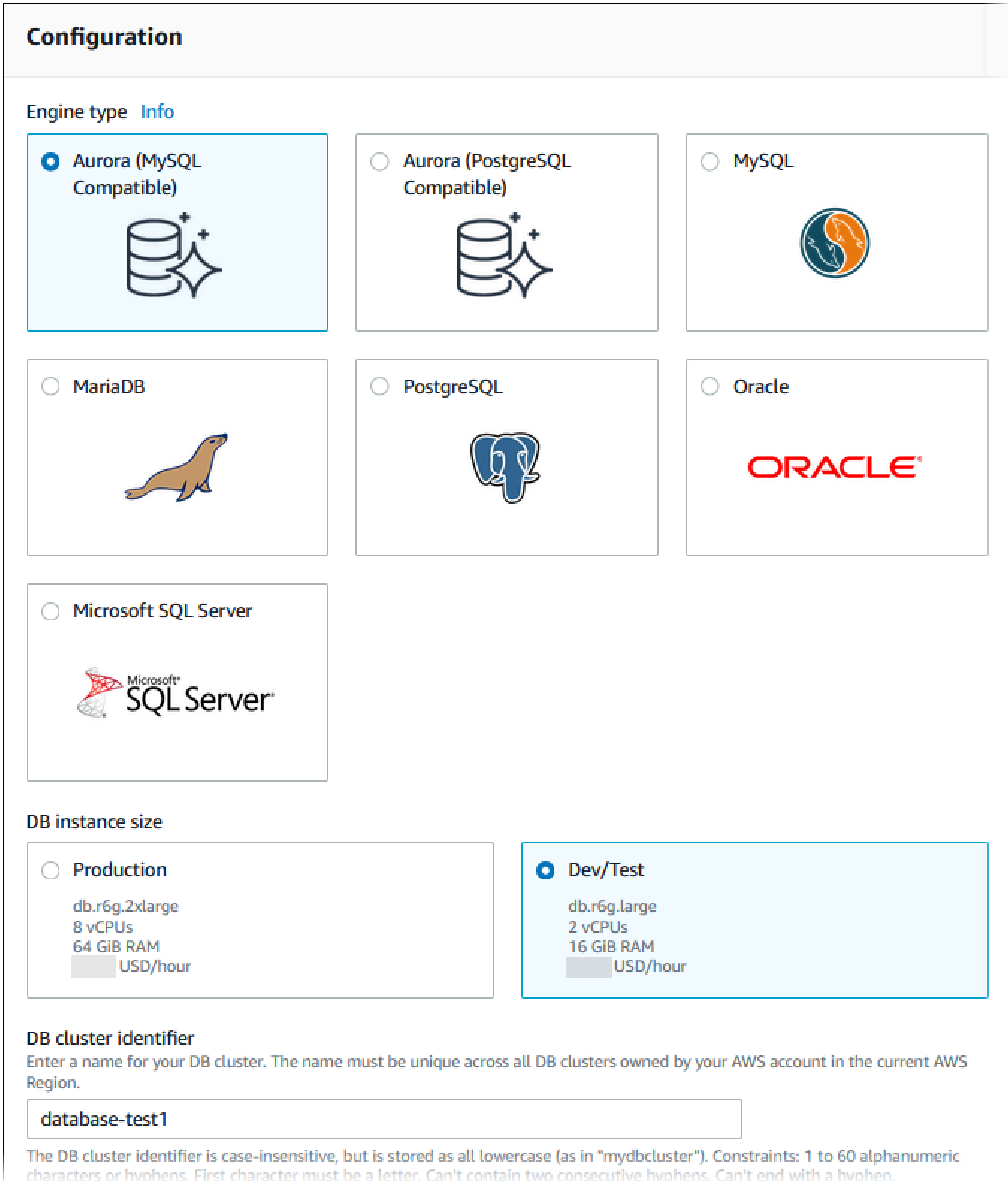
간편 생성을 사용하여 Aurora MySQL DB 클러스터를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 상단에서 DB 클러스터를 생성하려는 AWS 리전을 선택합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. [데이터베이스 생성(Create database)]을 선택하고 [간편 생성(Easy Create)]이 선택되어 있는지 확인합니다.



5. 구성의 엔진 유형에서 Aurora(MySQL 호환)를 선택합니다.
6. DB instance size(DB 인스턴스 크기)에서 개발/테스트를 선택합니다.
7. DB 클러스터 식별자에서 **`database-test1`**을 입력합니다.

   데이터베이스 생성 페이지는 다음 이미지와 비슷해야 합니다.

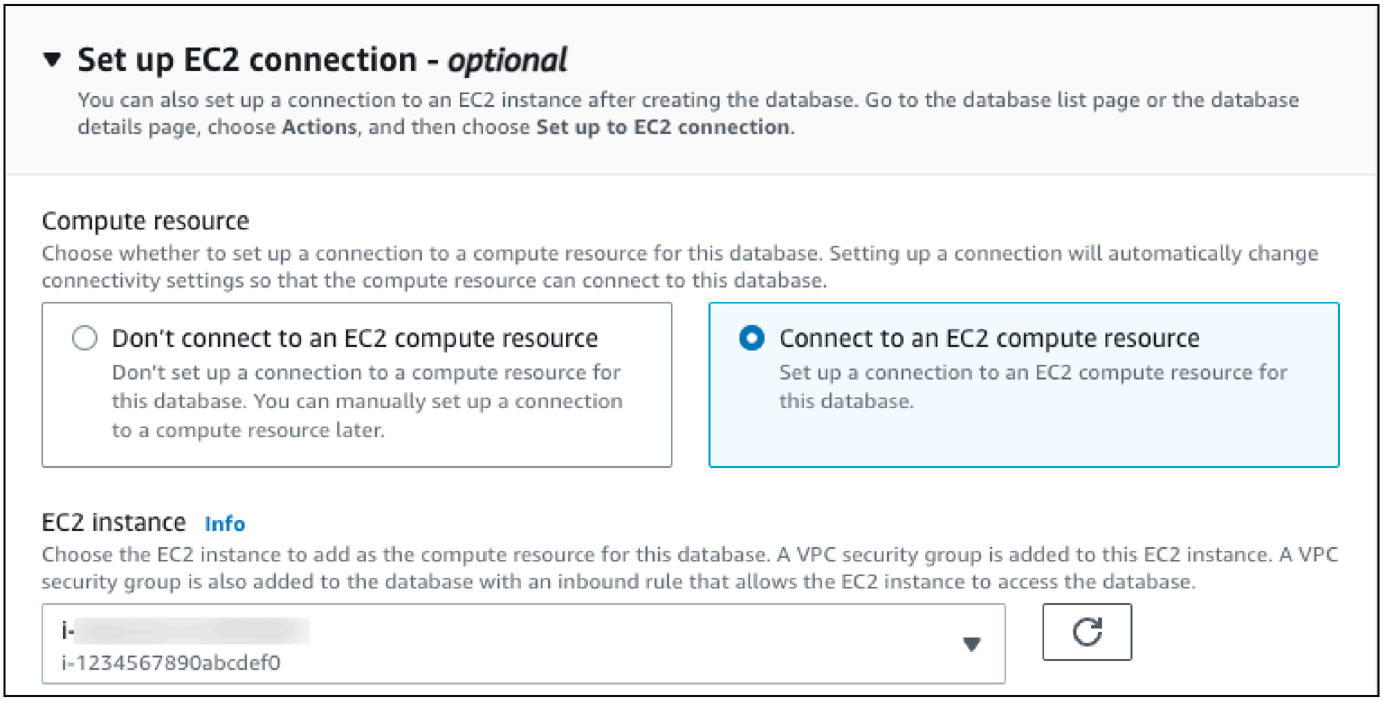


8. 마스터 사용자 이름에 마스터 사용자의 이름을 입력하거나 기본 이름을 그대로 유지합니다.
9. DB 클러스터에 자동 생성된 마스터 암호를 사용하려면 암호 자동 생성을 선택합니다.

마스터 암호를 입력하려면 암호 자동 생성 선택을 해제한 다음, 마스터 암호와 암호 확인에 동일한 암호를 입력합니다.

10. 이전에 생성한 EC2 인스턴스와의 연결을 설정하려면 EC2 연결 설정 - 선택 사항을 엽니다.

    Connect to an EC2 compute resource(EC2 컴퓨팅 리소스에 연결)를 선택합니다. 이전에 생성한 EC2 인스턴스를 선택합니다.



11. 간편 생성 기본 설정 보기를 엽니다.

▼ View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-mysql-8-0	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	8.0.mysql_aurora.3.02.0	Yes
DB parameter group	default.aurora-mysql8.0	Yes
DB cluster parameter group	default.aurora-mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

[간편 생성(Easy Create)]과 함께 사용되는 기본 설정을 검토할 수 있습니다. 데이터베이스 생성 후 편집 가능 열에는 데이터베이스 생성 후 어떤 옵션을 변경할 수 있는지 나와 있습니다.

- 설정의 해당 열에 아니요라고 되어 있지만 다른 설정을 원하는 경우 표준 생성을 사용하여 DB 클러스터를 만들 수 있습니다.
- 설정의 해당 열에 예라고 되어 있으며 다른 설정을 원하는 경우 표준 생성을 사용하여 DB 클러스터를 만들거나, DB 클러스터를 생성한 후 수정하여 설정을 변경할 수 있습니다.

12. 데이터베이스 생성를 선택합니다.

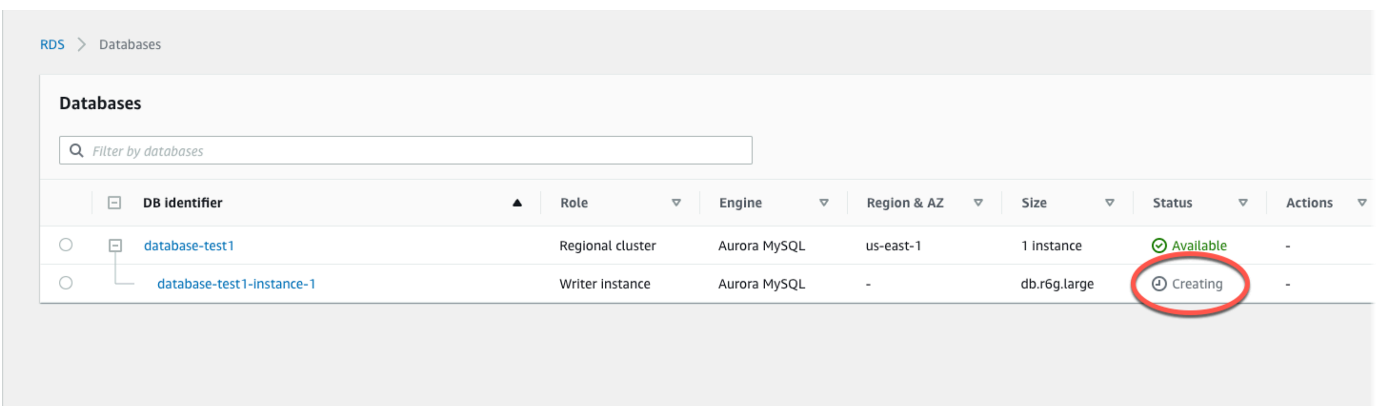
    DB 클러스터의 마스터 사용자 이름 및 암호를 보려면 자격 증명 세부 정보 보기를 선택합니다.

    DB 클러스터를 마스터 사용자로 연결하려면 화면에 나타난 사용자 이름과 암호를 사용합니다.

    > **Important**
    >
    > 마스터 사용자 암호를 다시 볼 수는 없습니다. 따라서 기록을 해두지 않으면 이를 변경해야 합니다.
    > DB 클러스터가 사용 가능한 상태가 되고 난 후에 마스터 사용자 암호를 변경해야 하는 경우에는 다음과 같은 방법으로 DB 클러스터를 수정할 수 있습니다. DB 클러스터 수정에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 수정](#) 단원을 참조하십시오.

13. 데이터베이스 목록에서 새 Aurora MySQL DB 클러스터의 이름을 선택하면 세부 정보가 표시됩니다.

    DB 클러스터를 사용할 준비가 될 때까지 라이터 인스턴스의 상태는 생성 중입니다.



    라이터 인스턴스의 상태가 사용 가능으로 변경되면 DB 클러스터에 연결할 수 있습니다. DB 인스턴스 클래스와 스토리지의 용량에 따라 새 DB 인스턴스를 사용할 수 있을 때까지 최대 20분이 걸릴 수 있습니다.

(선택 사항) AWS CloudFormation을 사용하여 VPC, EC2 인스턴스 및 Aurora MySQL 클러스터 생성

콘솔을 사용하여 VPC, EC2 인스턴스 및 Aurora MySQL DB 클러스터를 생성하는 대신 AWS CloudFormation을 통해 코드형 인프라로 처리하여 AWS 리소스를 프로비저닝할 수 있습니다. AWS 리소스를 더 작고 관리하기 쉬운 단위로 구성하는 데 도움이 되도록 AWS CloudFormation 중첩 스택 기능을 사용할 수 있습니다. 자세한 내용은 AWS CloudFormation 콘솔에서 스택 생성 및 중첩된 스택 작업을 참조하세요.

> **Important**
>
> AWS CloudFormation은 무료이지만, CloudFormation에서 생성하는 리소스는 라이브입니다. 이러한 리소스를 종료하지 않으면 해당 리소스에 대한 표준 사용 요금이 발생합니다. 자세한 내용은 Amazon Aurora 요금을 참조하세요.

AWS CloudFormation 콘솔을 사용하여 리소스를 생성하려면 다음 단계를 완료합니다.

- 1단계: CloudFormation 템플릿 다운로드
- 2단계: CloudFormation을 사용하여 리소스 구성

CloudFormation 템플릿 파일을 다운로드하십시오.

CloudFormation 템플릿은 JSON 또는 YAML 텍스트 파일로, 스택에서 생성하려는 리소스에 대한 구성 정보가 들어 있습니다. 또한 이 템플릿은 Aurora 클러스터와 함께 VPC와 Bastion Host를 생성합니다.

템플릿 파일을 다운로드하려면 다음 링크인 Aurora MySQL CloudFormation 템플릿을 엽니다.

Github 페이지에서 원시 파일 다운로드 버튼을 클릭하여 템플릿 YAML 파일을 저장합니다.

CloudFormation을 사용하여 리소스 구성

> **Note**
>
> 이 프로세스를 시작하기 전에 AWS 계정에 EC2 인스턴스용 키 페어가 있는지 확인합니다. 자세한 내용은 Amazon EC2 키 페어 및 Linux 인스턴스를 참조하세요.

AWS CloudFormation 템플릿을 사용할 때는 리소스가 제대로 생성되도록 올바른 파라미터를 선택해야 합니다. 다음 단계를 따릅니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택합니다.
3. 템플릿 지정 섹션에서 컴퓨터에서 템플릿 파일 업로드를 선택하고 다음을 선택합니다.
4. 스택 세부 정보 지정 페이지에서 다음 파라미터를 설정합니다.
   a. 스택 이름을 AurMySQLTestStack으로 설정합니다.
   b. 파라미터에서 가용 영역 2개를 선택하여 가용 영역을 설정합니다.
   c. Linux Bastion Host 구성에서 키 이름에 대해 EC2 인스턴스에 로그인할 키 페어를 선택합니다.
   d. Linux Bastion Host 구성 설정에서 허용된 IP 범위를 IP 주소로 설정합니다. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하려면 https://checkip.amazonaws.com의 서비스를 사용하여 퍼블릭 IP 주소를 지정합니다. IP 주소의 예는 192.0.2.1/32입니다.

      > **Warning**
      >
      > SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 EC2 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 EC2 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

   e. 데이터베이스 일반 구성에서 데이터베이스 인스턴스 클래스를 db.r6g.large로 설정합니다.
   f. 데이터베이스 이름을 **`database-test1`**으로 설정합니다.
   g. 데이터베이스 마스터 사용자 이름에 마스터 사용자 이름을 입력합니다.
   h. 이 자습서에서는 Secrets Manager를 사용하여 DB 마스터 사용자 암호 관리를 `false`로 설정합니다.
   i. 데이터베이스 암호의 경우 원하는 암호를 설정합니다. 자습서의 향후 단계에 사용할 수 있도록 암호를 기억해 둡니다.
   j. 다중 AZ 배포를 `false`로 설정합니다.
   k. 다른 모든 설정은 기본값으로 둡니다. 계속하려면 다음을 클릭합니다.
5. 스택 옵션 구성 페이지에서는 모든 기본 옵션을 그대로 둡니다. 계속하려면 다음을 클릭합니다.
6. 스택 검토 페이지에서 데이터베이스 및 Linux Bastion Host 옵션을 확인한 후 제출을 선택합니다.

스택 생성 프로세스가 완료되면 BastionStack 및 AMSNS라는 이름의 스택을 보고 데이터베이스에 연결하는 데 필요한 정보를 기록해 둡니다. 자세한 내용은 AWS Management Console에서 AWS CloudFormation 스택 데이터 및 리소스 보기를 참조하세요.

3단계: Aurora MySQL DB 클러스터에 연결

표준 SQL 클라이언트 애플리케이션을 사용해 DB 클러스터에 연결할 수 있습니다. 이 예시에서는 mysql 명령줄 클라이언트를 사용하여 Aurora MySQL DB 클러스터에 연결합니다.

Aurora MySQL DB 클러스터에 연결하려면

1. DB 클러스터에 대한 라이트 인스턴스의 엔드포인트(DNS 이름)와 포트 번호를 찾습니다.

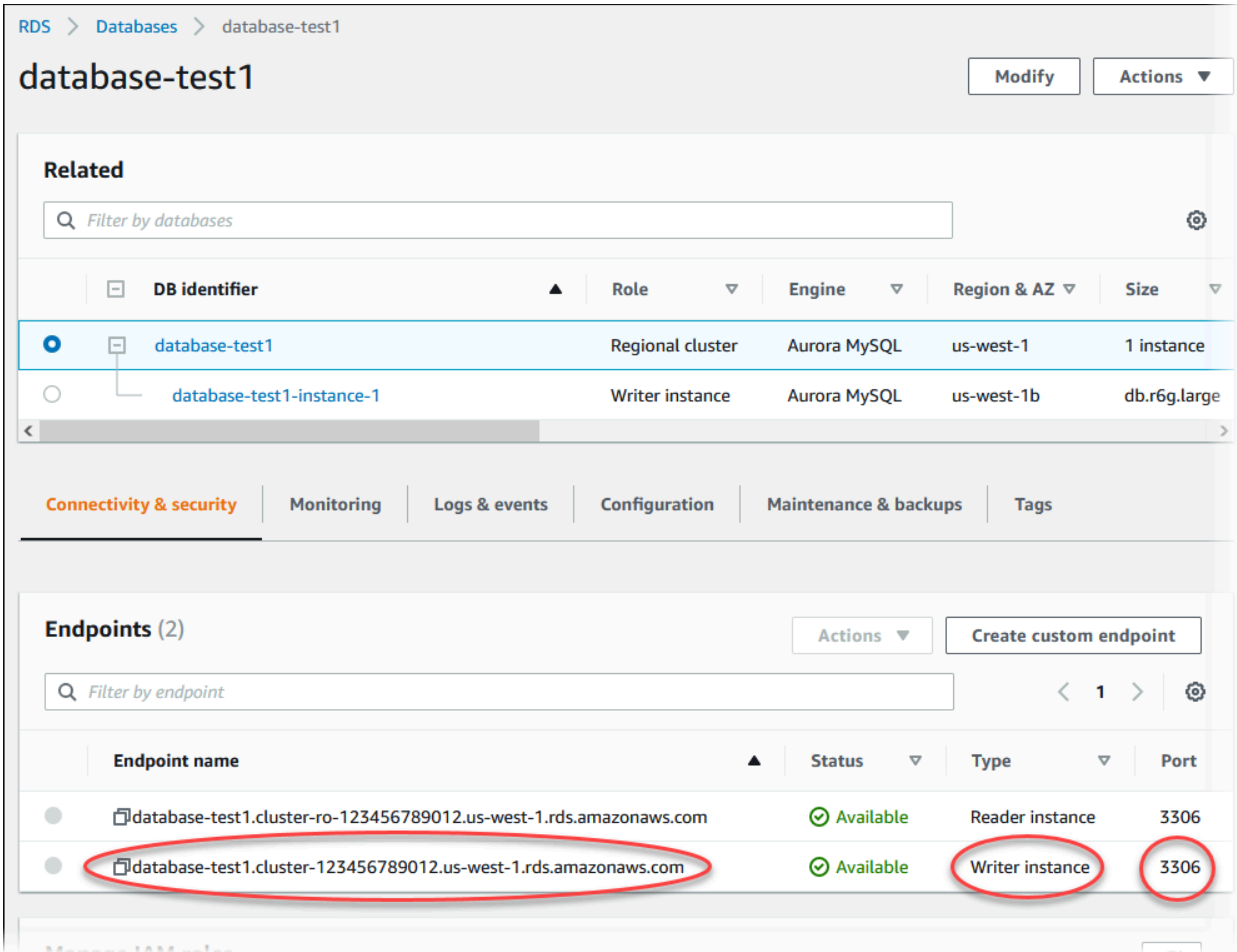
    a. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

    b. Amazon RDS 콘솔의 오른쪽 상단에서 DB 클러스터의 AWS 리전을 선택합니다.

    c. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

    d. Aurora MySQL DB 클러스터 이름을 선택하여 세부 정보를 표시합니다.

    e. 연결 및 보안 탭에서 라이터 인스턴스의 엔드포인트를 복사합니다. 또한 포트 번호를 적어 둡니다. DB 클러스터에 연결하려면 엔드포인트와 포트 번호가 모두 필요합니다.



2. Amazon EC2 사용 설명서에 있는 [Linux 인스턴스에 연결](#)의 단계를 따라 앞에서 만든 EC2 인스턴스에 연결합니다.

   SSH를 사용하여 EC2 인스턴스에 연결하는 것이 좋습니다. Windows, Linux 또는 Mac에 SSH 클라이언트 유틸리티가 설치된 경우 다음 명령 형식을 사용하여 인스턴스에 연결할 수 있습니다.

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

   예를 들어 `ec2-database-connect-key-pair.pem`이 Linux의 `/dir1`에 저장되어 있고, EC2 인스턴스의 퍼블릭 IPv4 DNS가 `ec2-12-345-678-90.compute-1.amazonaws.com`이라고 가정해 보겠습니다. 그러면 SSH 명령은 다음과 같이 표시됩니다.

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. EC2 인스턴스에서 소프트웨어를 업데이트하여 최신 버그 수정 및 보안 업데이트를 받습니다. 이렇게 하려면 다음 명령을 사용합니다.

> **Note**
>
> -y 옵션을 사용하면 확인 여부를 묻지 않고 업데이트를 설치합니다. 설치 전에 업데이트 정보를 확인하려면 이 옵션을 생략합니다.

```
sudo dnf update -y
```

4. Amazon Linux 2023에서 MariaDB의 mysql 명령줄 클라이언트를 설치하려면 다음 명령을 실행합니다.

```
sudo dnf install mariadb105
```

5. Aurora MySQL DB 클러스터에 연결합니다. 예를 들어, 다음 명령을 입력합니다. 이 작업을 통해 MySQL 클라이언트를 사용하여 Aurora MySQL DB 클러스터에 연결할 수 있습니다.

   *endpoint*는 라이터 인스턴스 엔드포인트로 대체하고, *admin*는 사용된 마스터 사용자 이름으로 대체합니다. 암호를 묻는 메시지가 표시되면 사용한 마스터 암호를 제공합니다.

```
mysql -h endpoint -P 3306 -u admin -p
```

   사용자에 대한 암호를 입력하면 다음과 유사한 출력이 나타납니다.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

   Aurora MySQL DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora MySQL DB 클러스터에 연결 섹션을 참조하세요. DB 클러스터에 연결할 수 없는 경우 Amazon RDS DB 인스턴스에 연결할 수 없음 섹션을 참조하세요.

보안을 위해서는 암호화된 연결을 사용하는 것이 가장 좋습니다. 클라이언트와 서버가 동일한 VPC에 있고 네트워크를 신뢰할 수 있는 경우에만 암호화되지 않은 MySQL 연결을 사용합니다. 암호화된 연결 사용에 대한 자세한 내용은 SSL을 사용하여 Aurora MySQL에 연결 섹션을 참조하세요.

6. SQL 명령을 실행합니다.

   예를 들어, 다음 SQL 명령은 현재 날짜 및 시간을 보여줍니다.

```
SELECT CURRENT_TIMESTAMP;
```

4단계: EC2 인스턴스 및 DB 클러스터 삭제

생성한 샘플 EC2 인스턴스 및 DB 클러스터에 연결하고 탐색한 후에는 요금이 더 이상 부과되지 않도록 삭제합니다.

AWS CloudFormation을 사용하여 리소스를 생성했다면 이 단계를 건너뛰고 다음 단계로 이동합니다.

EC2 인스턴스를 삭제하는 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.
3. EC2 인스턴스를 선택하고 인스턴스 상태, 인스턴스 종료를 차례로 선택합니다.
4. 확인 메시지가 나타나면 종료를 선택합니다.

EC2 인스턴스 삭제에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료를 참조하세요.

DB 클러스터를 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 DB 클러스터에 연결된 DB 인스턴스를 선택합니다.
3. [Actions]에 대해 [Delete]를 선택합니다.
4. 최종 스냅샷 생성 여부를 선택 해제합니다.
5. 확인을 완료하고 삭제를 선택합니다.

DB 클러스터에 연결된 모든 DB 인스턴스가 삭제되고 나면 DB 클러스터가 자동으로 삭제됩니다.

(선택 사항) CloudFormation으로 생성한 EC2 인스턴스 및 DB 클러스터 삭제

AWS CloudFormation을 사용하여 리소스를 생성한 경우 샘플 EC2 인스턴스 및 DB 클러스터에 연결하고 탐색한 후 CloudFormation 스택을 삭제하여 더 이상 비용이 청구되지 않도록 합니다.

CloudFormation 리소스를 삭제하려면

1. AWS CloudFormation 콘솔을 엽니다.
2. CloudFormation 콘솔의 스택 페이지에서 루트 스택(VPCStack, BastionStack 또는 AMSNS라는 이름이 없는 스택)을 선택합니다.
3. 삭제를 선택합니다.
4. 확인 메시지가 나타나면 스택 삭제를 선택합니다.

CloudFormation에서 스택을 삭제하는 방법에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation 콘솔에서 스택 삭제를 참조하세요.

(선택 사항) DB 클러스터를 Lambda 함수에 연결

Aurora MySQL DB 클러스터를 Lambda 서버리스 컴퓨팅 리소스에 연결할 수도 있습니다. Lambda 함수를 사용하면 인프라를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있습니다. 또한 하루 12개에서 초당 수백 개에 이르는 모든 규모의 코드 실행 요청에 자동으로 응답할 수 있습니다. 자세한 내용은 Lambda 함수와 Aurora DB 클러스터 자동 연결 섹션을 참조하세요.

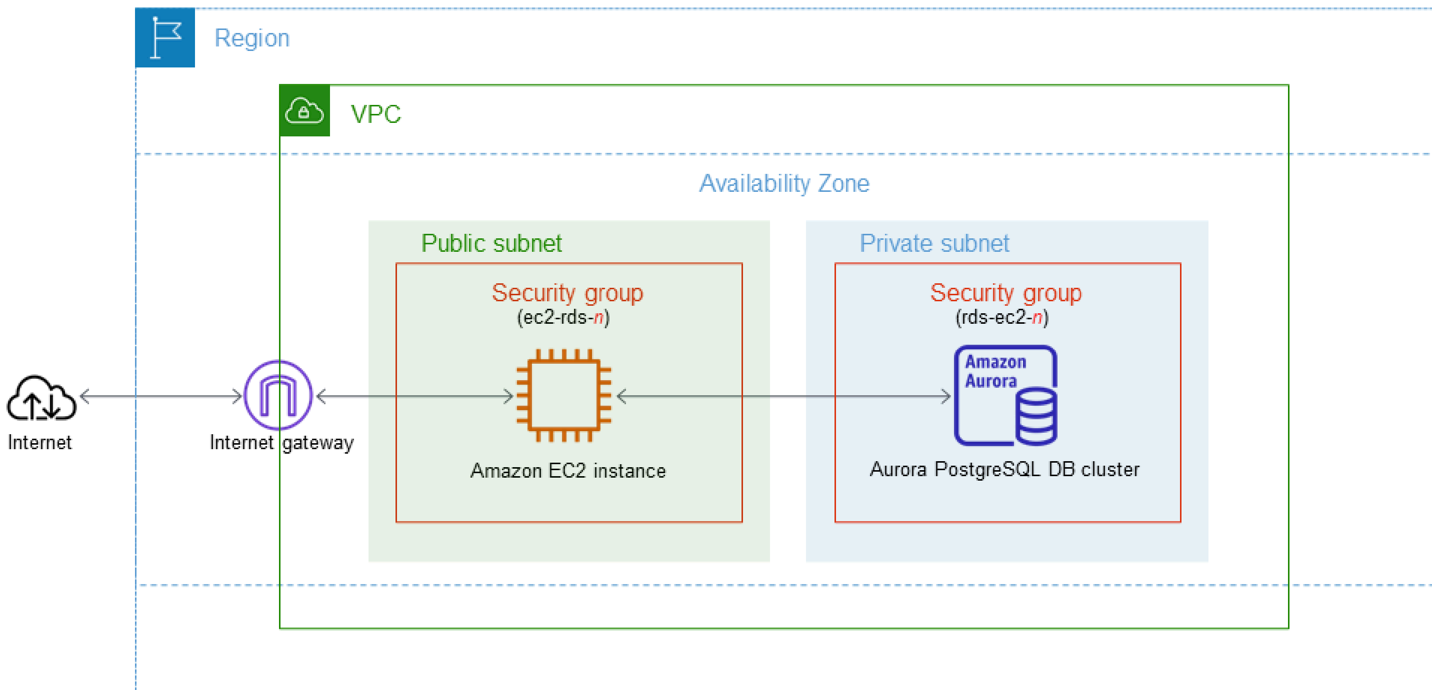
Aurora PostgreSQL DB 클러스터 생성 및 연결

이 튜토리얼에서는 EC2 인스턴스 및 Aurora PostgreSQL DB 클러스터를 생성합니다. 튜토리얼에서는 표준 PostgreSQL 클라이언트를 사용하여 EC2 인스턴스에서 DB 클러스터에 액세스하는 방법을 보여줍니다. 이 튜토리얼에서는 모범 사례를 따라 Virtual Private Cloud(VPC)에서 프라이빗 DB 클러스터를 생성합니다. 대부분의 경우 EC2 인스턴스와 같이 동일한 VPC에 있는 다른 리소스는 DB 인스턴스에 액세스할 수 있지만 VPC 외부의 리소스는 DB 클러스터에 액세스할 수 없습니다.

자습서를 완료하면 VPC의 각 가용 영역에 퍼블릭 서브넷과 프라이빗 서브넷이 있을 것입니다. 한 가용 영역에서 EC2 인스턴스는 퍼블릭 서브넷에 있고 DB 인스턴스는 프라이빗 서브넷에 있습니다.

> **Important**
>
> AWS 계정 생성은 무료입니다. 그러나 이 튜토리얼을 완료하면 사용하는 AWS 리소스에 대한 비용이 발생할 수 있습니다. 자습서가 더 이상 필요하지 않은 경우 자습서를 완료한 후에 이러한 리소스를 삭제할 수 있습니다.

다음 다이어그램은 이 자습서를 완료했을 때 구성을 보여 줍니다.



이 자습서에서는 다음 방법 중 하나를 사용하여 리소스를 생성할 수 있습니다.

1. AWS Management Console 사용 - 1단계: EC2 인스턴스 생성 및 2단계: Aurora PostgreSQL DB 클러스터 생성
2. 데이터베이스 인스턴스 및 EC2 인스턴스를 생성하는 데 AWS CloudFormation 사용 - (선택 사항) AWS CloudFormation을 사용하여 VPC, EC2 인스턴스 및 Aurora PostgreSQL 클러스터 생성

첫 번째 방법은 간편 생성을 사용하여 AWS Management Console을 통해 프라이빗 Aurora PostgreSQL DB 클러스터를 생성합니다. 여기에서는 DB 엔진 유형, DB 인스턴스 크기, DB 클러스터 식별자만 지정합니다. [간편 생성(Easy create)]은 다른 구성 옵션에서도 기본 설정을 사용합니다.

표준 생성을 대신 사용하는 경우에는 DB 클러스터를 생성할 때 더 많은 구성 옵션을 지정할 수 있습니다. 이러한 옵션에는 가용성, 보안, 백업 및 유지 관리에 대한 설정이 포함됩니다. 퍼블릭 DB 클러스터

를 만들려면 표준 생성을 사용해야 합니다. 자세한 내용은 the section called “DB 클러스터 생성” 섹션을 참조하세요.

주제

사전 조건

시작하기 전에 다음 섹션에서 다음 단계를 완료하세요.

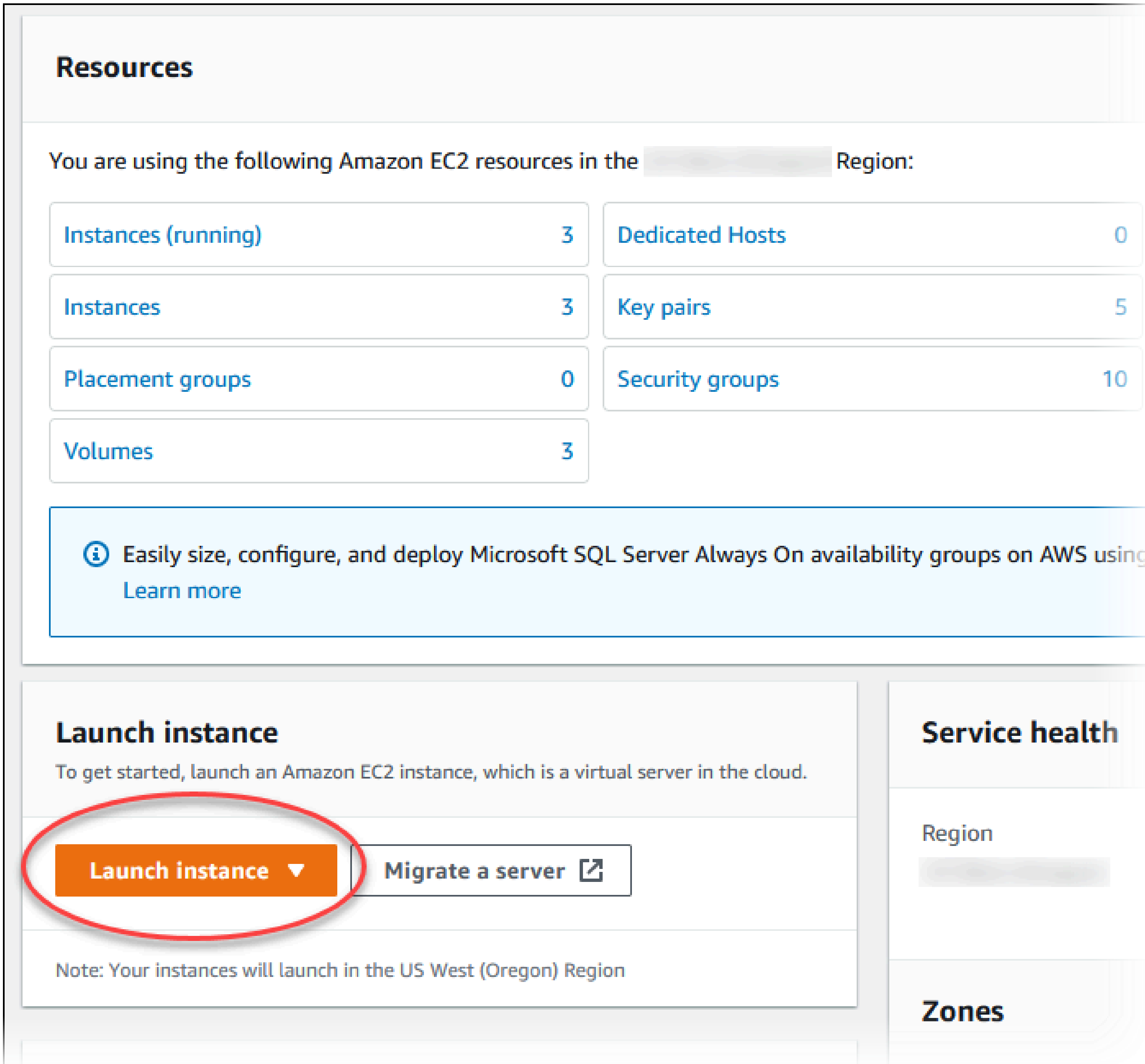
- AWS 계정에 등록
- 관리자 액세스 권한이 있는 사용자 생성

1단계: EC2 인스턴스 생성

데이터베이스에 연결하는 데 사용할 Amazon EC2 인스턴스를 생성합니다.

EC2 인스턴스를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 EC2 인스턴스를 생성하려는 AWS 리전을 선택합니다.
3. 다음 이미지에 나와 있는 것처럼 EC2 대시보드를 선택한 다음, 인스턴스 시작을 선택합니다.

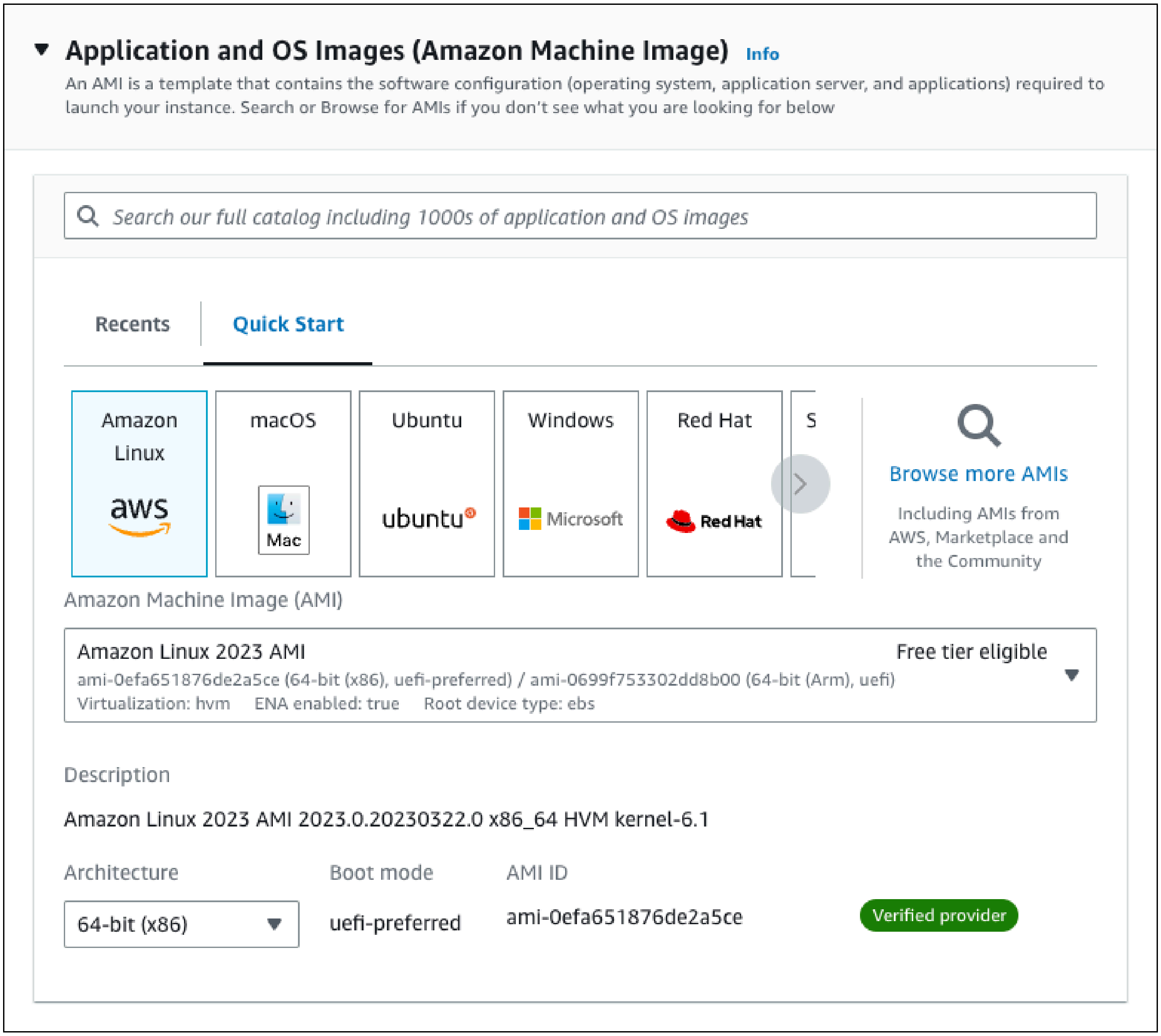


인스턴스 시작 페이지가 열립니다.

4. 인스턴스 시작 페이지에서 다음 설정을 선택합니다.

   a. Name and tags(이름 및 태그) 아래의 Name(이름)에 **`ec2-database-connect`**을 입력하세요.

   b. Application and OS Images (Amazon Machine Image)(애플리케이션 및 OS 이미지(Amazon Machine Image))에서 Amazon Linux를 선택한 다음 Amazon Linux 2023 AMI를 선택합니다. 다른 선택 항목에 대해서는 기본값을 그대로 유지합니다.



c. 인스턴스 유형에서 t2.micro를 선택합니다.

d. 키 페어(로그인)에서 기존 키 페어를 사용할 키 페어 이름을 선택합니다. Amazon EC2 인스턴스에 대한 새 키 페어를 생성하려면 새 키 페어 생성을 선택한 다음 키 페어 생성 창을 사용하여 생성합니다.

   새 키 페어 생성에 대한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 키 페어 생성을 참조하세요.

e. 네트워크 설정의 SSH 트래픽 허용에서 EC2 인스턴스에 대한 SSH 연결 소스를 선택합니다.

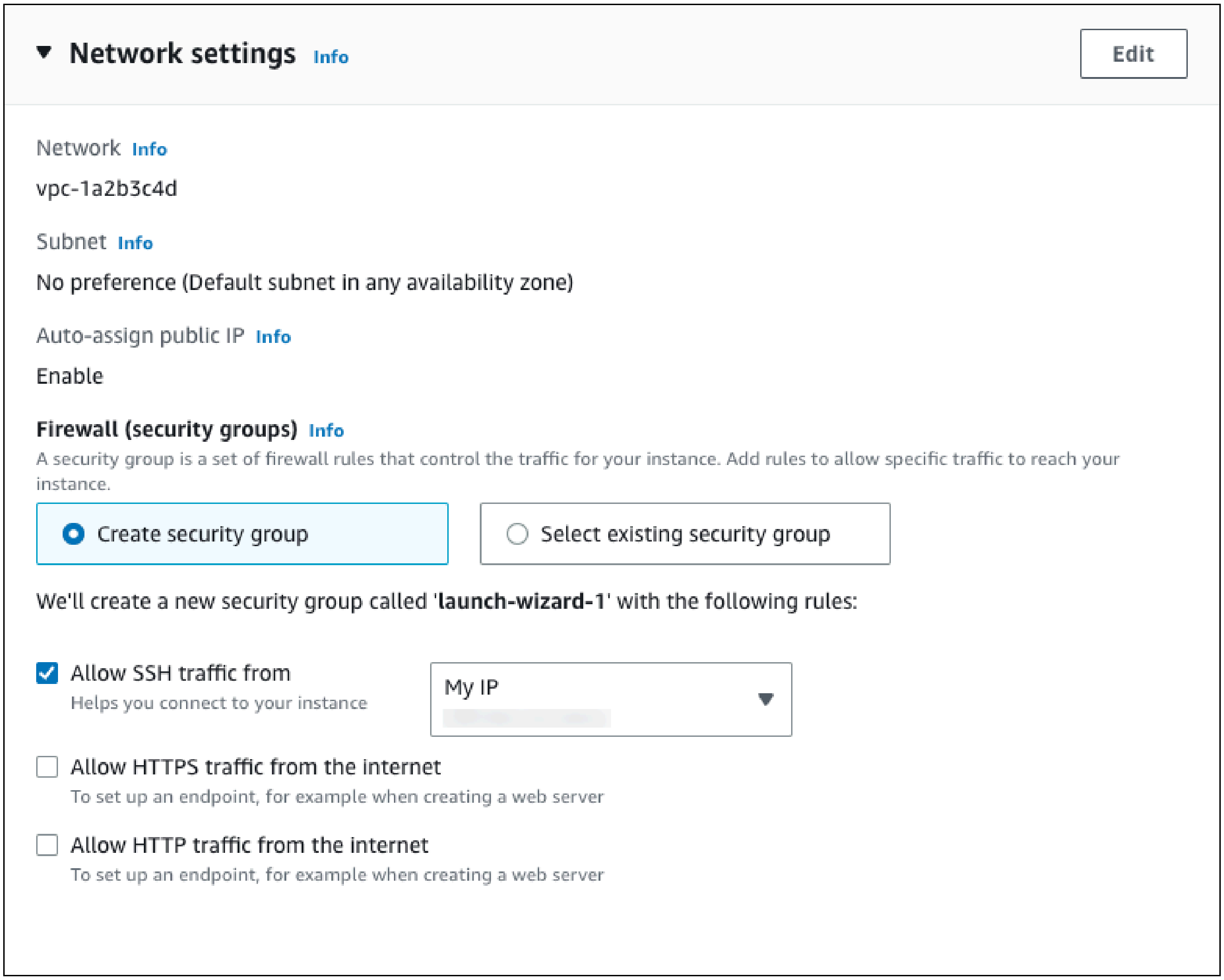
   표시된 IP 주소가 SSH 연결에 대해 올바른 경우 내 IP를 선택할 수 있습니다. 그렇지 않으면 SSH(Secure Shell)를 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를

결정할 수 있습니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다. IP 주소의 예는 192.0.2.1/32입니다.

대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 결정합니다.

> **Warning**
>
> SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 EC2 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 EC2 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

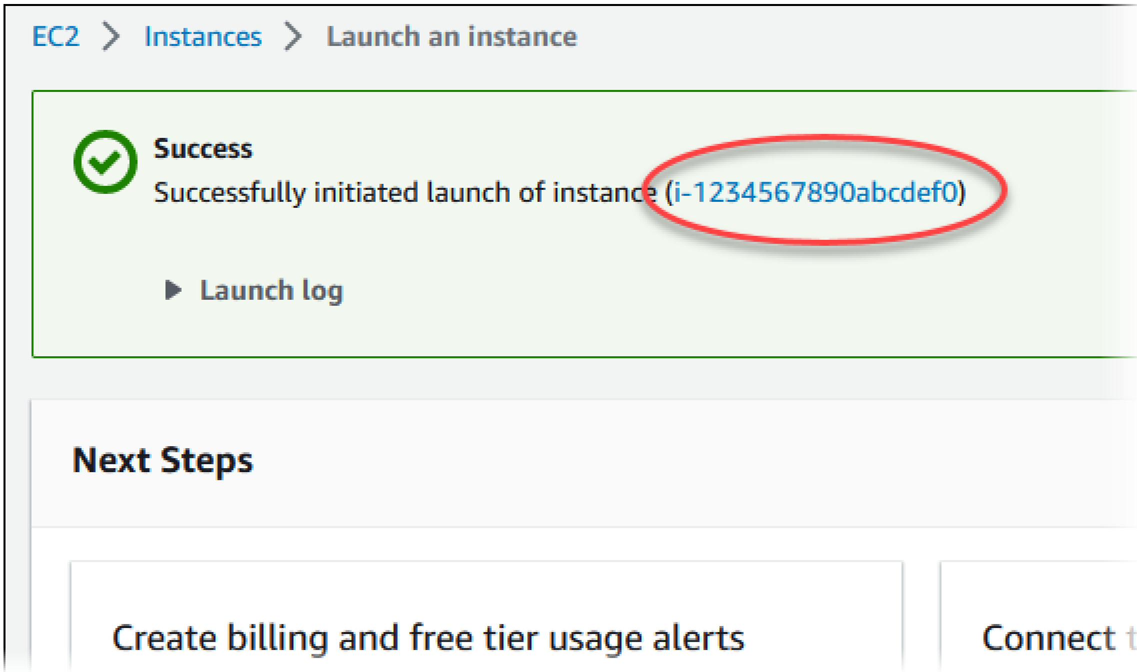
다음 이미지는 네트워크 설정 섹션의 예를 보여 줍니다.



f. 나머지 섹션에서 기본값은 그대로 둡니다.

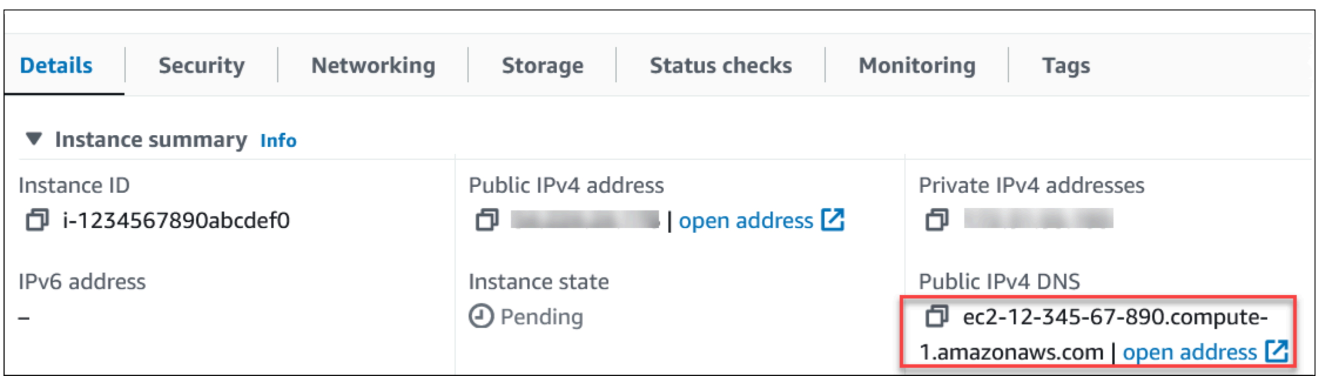
g. 요약 패널에서 EC2 인스턴스 구성 요약을 검토하고 준비가 되면 인스턴스 시작을 선택합니다.

5. 시작 상태 페이지에서, 새 EC2 인스턴스의 식별자(예: `i-1234567890abcdef0`)를 기록해 둡니다.

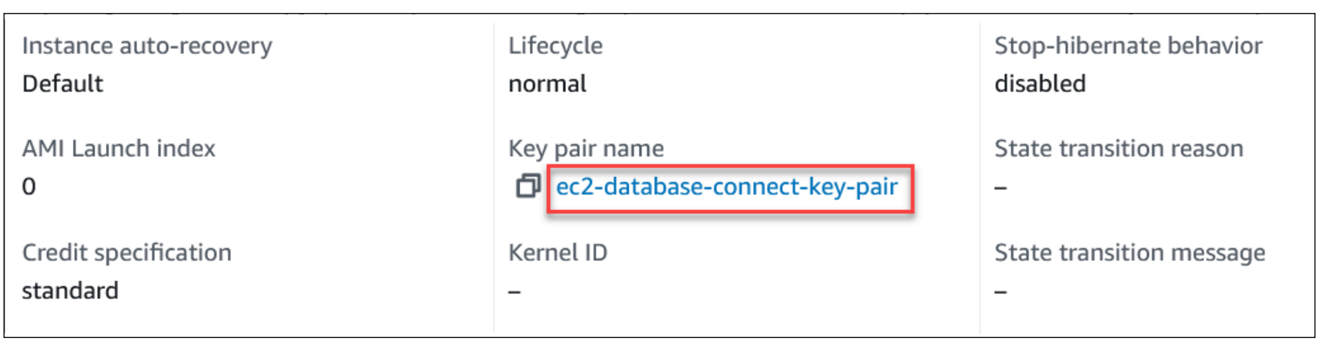


6. EC2 인스턴스 식별자를 선택하여 EC2 인스턴스 목록을 열고 EC2 인스턴스를 선택합니다.
7. 세부 정보 탭에서 SSH를 사용하여 연결할 때 필요한 다음 값을 기록해 둡니다.

    a. 인스턴스 요약에서 퍼블릭 IPv4 DNS의 값을 기록해 둡니다.



    b. 인스턴스 세부 정보에서 키 페어 이름의 값을 기록해 둡니다.



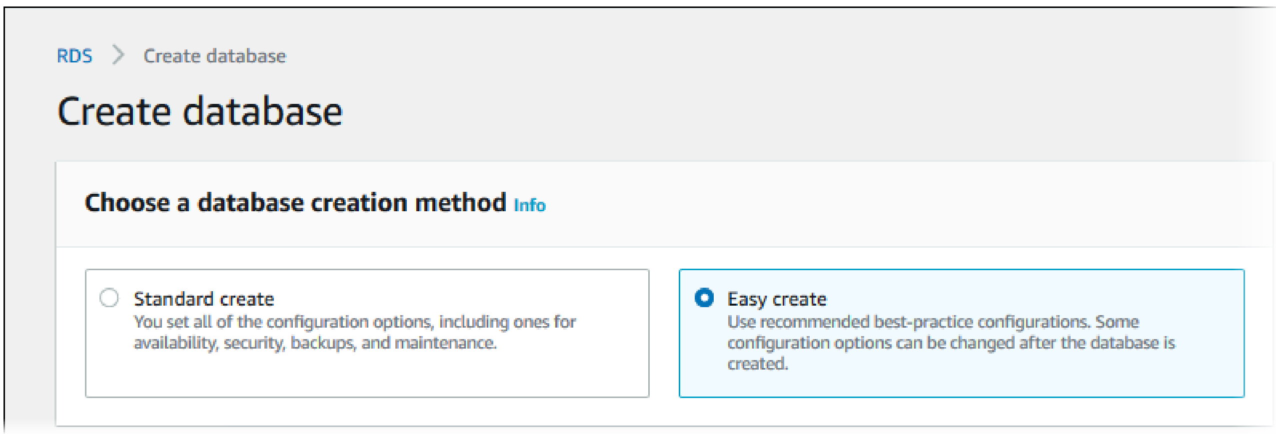
8. 계속하기 전에 EC2 인스턴스의 인스턴스 상태가 실행 중이 될 때까지 기다립니다.

2단계: Aurora PostgreSQL DB 클러스터 생성

이 예시에서는 간편 생성을 사용하여 db.t4g.large DB 인스턴스 클래스로 Aurora PostgreSQL DB 클러스터를 생성합니다.

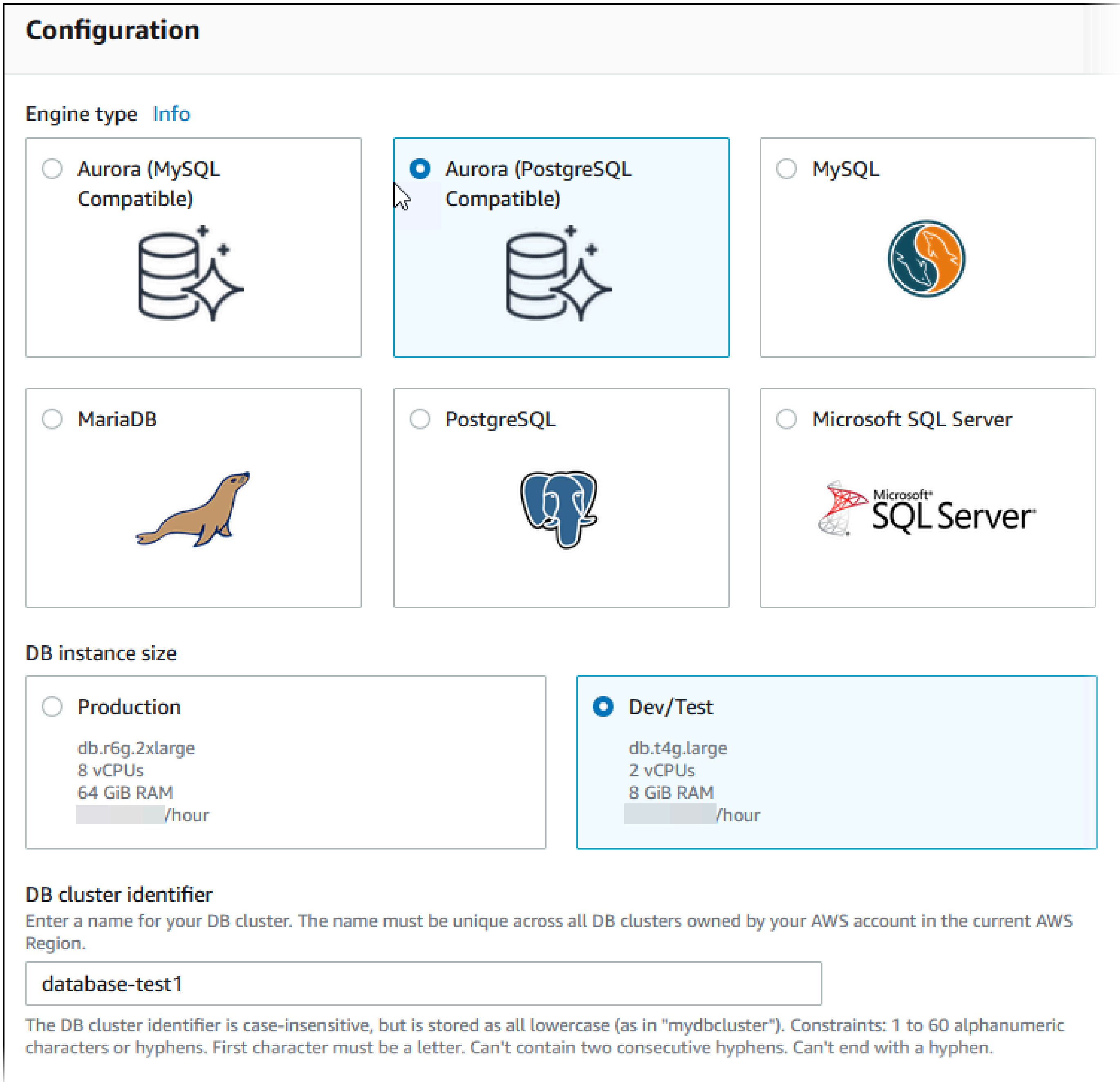
간편 생성을 사용하여 Aurora PostgreSQL DB 클러스터를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 상단에서 DB 클러스터를 생성하려는 AWS 리전을 선택합니다.
3. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
4. 데이터베이스 생성을 선택하고 간편 생성이 선택되어 있는지 확인합니다.



5. 구성의 엔진 유형에서 Aurora (PostgreSQL Compatible)(Aurora(PostgreSQL 호환))을 선택합니다.
6. DB instance size(DB 인스턴스 크기)에서 개발/테스트를 선택합니다.
7. DB 클러스터 식별자에서 **`database-test1`**을 입력합니다.

   데이터베이스 생성 페이지는 다음 이미지와 비슷해야 합니다.



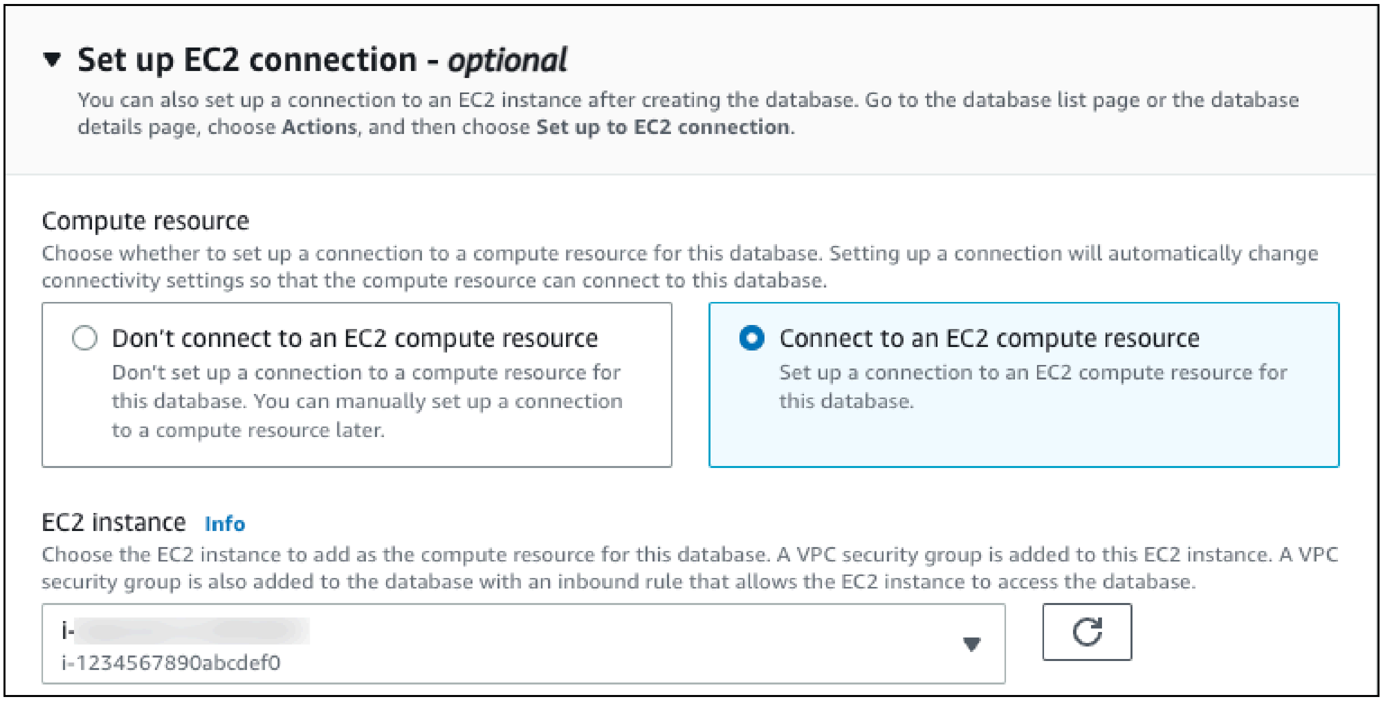
8. 마스터 사용자 이름에 사용자의 이름을 입력하거나 기본 이름(**`postgres`**)을 그대로 사용합니다.

9. DB 클러스터에 자동 생성된 마스터 암호를 사용하려면 암호 자동 생성을 선택합니다.

   마스터 암호를 입력하려면 암호 자동 생성 선택을 해제한 다음, 마스터 암호와 암호 확인에 동일한 암호를 입력합니다.

10. 이전에 생성한 EC2 인스턴스와의 연결을 설정하려면 EC2 연결 설정 - 선택 사항을 엽니다.

Connect to an EC2 compute resource(EC2 컴퓨팅 리소스에 연결)를 선택합니다. 이전에 생성한 EC2 인스턴스를 선택합니다.



11. 간편 생성 기본 설정 보기를 엽니다.

▼ View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

[간편 생성(Easy Create)]과 함께 사용되는 기본 설정을 검토할 수 있습니다. 데이터베이스 생성 후 편집 가능 열에는 데이터베이스 생성 후 어떤 옵션을 변경할 수 있는지 나와 있습니다.

- 설정의 해당 열에 아니요라고 되어 있지만 다른 설정을 원하는 경우 표준 생성을 사용하여 DB 클러스터를 만들 수 있습니다.
- 설정의 해당 열에 예라고 되어 있으며 다른 설정을 원하는 경우 표준 생성을 사용하여 DB 클러스터를 만들거나, DB 클러스터를 생성한 후 수정하여 설정을 변경할 수 있습니다.

12. 데이터베이스 생성를 선택합니다.

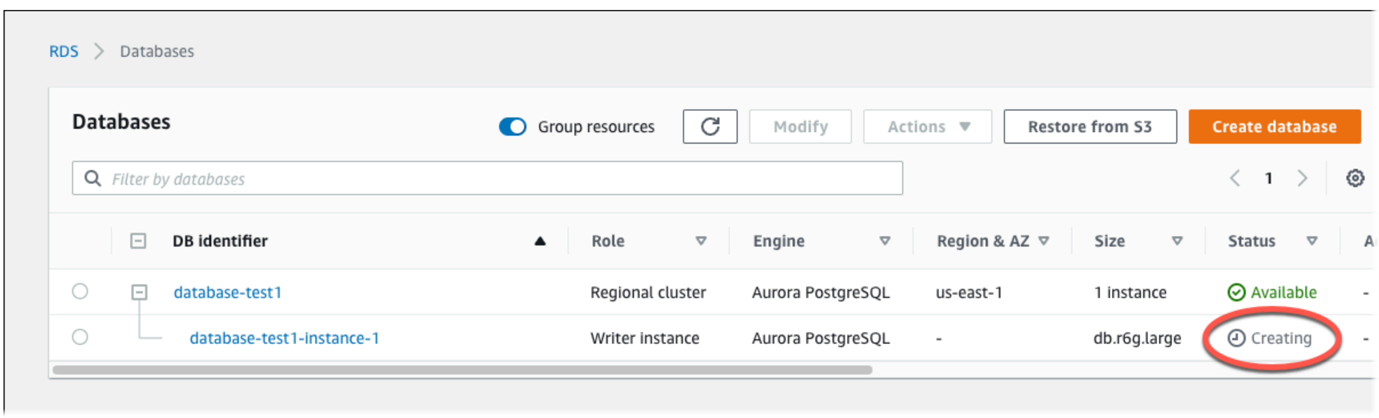
    DB 클러스터의 마스터 사용자 이름 및 암호를 보려면 자격 증명 세부 정보 보기를 선택합니다.

    DB 클러스터를 마스터 사용자로 연결하려면 화면에 나타난 사용자 이름과 암호를 사용합니다.

    > **Important**
    >
    > 마스터 사용자 암호를 다시 볼 수는 없습니다. 따라서 기록을 해두지 않으면 이를 변경해야 합니다.
    > DB 클러스터가 사용 가능한 상태가 되고 난 후에 마스터 사용자 암호를 변경해야 하는 경우에는 다음과 같은 방법으로 DB 클러스터를 수정할 수 있습니다. DB 클러스터 수정에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 수정](#) 단원을 참조하십시오.

13. 데이터베이스 목록에서 새 Aurora PostgreSQL DB 클러스터의 이름을 선택하면 세부 정보가 표시됩니다.

    DB 클러스터를 사용할 준비가 될 때까지 라이터 인스턴스의 상태는 생성 중입니다.



    라이터 인스턴스의 상태가 사용 가능으로 변경되면 DB 클러스터에 연결할 수 있습니다. DB 인스턴스 클래스와 스토리지의 용량에 따라 새 DB 인스턴스를 사용할 수 있을 때까지 최대 20분이 걸릴 수 있습니다.

(선택 사항) AWS CloudFormation을 사용하여 VPC, EC2 인스턴스 및 Aurora PostgreSQL 클러스터 생성

콘솔을 사용하여 VPC, EC2 인스턴스 및 Aurora PostgreSQL DB 클러스터를 생성하는 대신 AWS CloudFormation을 통해 코드형 인프라로 처리하여 AWS 리소스를 프로비저닝할 수 있습니다. AWS 리소스를 더 작고 관리하기 쉬운 단위로 구성하는 데 도움이 되도록 AWS CloudFormation 중첩 스택 기능을 사용할 수 있습니다. 자세한 내용은 [AWS CloudFormation 콘솔에서 스택 생성](#) 및 [중첩된 스택 작업](#)을 참조하세요.

> **Important**
>
> AWS CloudFormation은 무료이지만, CloudFormation에서 생성하는 리소스는 라이브입니다. 이러한 리소스를 종료하지 않으면 해당 리소스에 대한 표준 사용 요금이 발생합니다. 자세한 내용은 [Amazon Aurora 요금](#)을 참조하세요.

AWS CloudFormation 콘솔을 사용하여 리소스를 생성하려면 다음 단계를 완료합니다.

- 1단계: CloudFormation 템플릿 다운로드
- 2단계: CloudFormation을 사용하여 리소스 구성

CloudFormation 템플릿 파일을 다운로드하십시오.

CloudFormation 템플릿은 JSON 또는 YAML 텍스트 파일로, 스택에서 생성하려는 리소스에 대한 구성 정보가 들어 있습니다. 또한 이 템플릿은 Aurora 클러스터와 함께 VPC와 Bastion Host를 생성합니다.

템플릿 파일을 다운로드하려면 다음 링크인 [Aurora PostgreSQL CloudFormation 템플릿](#)을 엽니다.

Github 페이지에서 원시 파일 다운로드 버튼을 클릭하여 템플릿 YAML 파일을 저장합니다.

CloudFormation을 사용하여 리소스 구성

> **Note**
>
> 이 프로세스를 시작하기 전에 AWS 계정에 EC2 인스턴스용 키 페어가 있는지 확인합니다. 자세한 내용은 [Amazon EC2 키 페어 및 Linux 인스턴스](#)를 참조하세요.

AWS CloudFormation 템플릿을 사용할 때는 리소스가 제대로 생성되도록 올바른 파라미터를 선택해야 합니다. 다음 단계를 따릅니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 생성을 선택합니다.
3. 템플릿 지정 섹션에서 컴퓨터에서 템플릿 파일 업로드를 선택하고 다음을 선택합니다.
4. 스택 세부 정보 지정 페이지에서 다음 파라미터를 설정합니다.
   a. 스택 이름을 AurPostgreSQLTestStack으로 설정합니다.
   b. 파라미터에서 가용 영역 2개를 선택하여 가용 영역을 설정합니다.
   c. Linux Bastion Host 구성에서 키 이름에 대해 EC2 인스턴스에 로그인할 키 페어를 선택합니다.
   d. Linux Bastion Host 구성 설정에서 허용된 IP 범위를 IP 주소로 설정합니다. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하려면 https://checkip.amazonaws.com의 서비스를 사용하여 퍼블릭 IP 주소를 지정합니다. IP 주소의 예는 192.0.2.1/32입니다.

   > **Warning**
   >
   > SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 EC2 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 EC2 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

   e. 데이터베이스 일반 구성에서 데이터베이스 인스턴스 클래스를 db.t4g.large로 설정합니다.
   f. 데이터베이스 이름을 **`database-test1`**으로 설정합니다.
   g. 데이터베이스 마스터 사용자 이름에 마스터 사용자 이름을 입력합니다.
   h. 이 자습서에서는 Secrets Manager를 사용하여 DB 마스터 사용자 암호 관리를 `false`로 설정합니다.
   i. 데이터베이스 암호의 경우 원하는 암호를 설정합니다. 자습서의 향후 단계에 사용할 수 있도록 암호를 기억해 둡니다.
   j. 다중 AZ 배포를 `false`로 설정합니다.
   k. 다른 모든 설정은 기본값으로 둡니다. 계속하려면 다음을 클릭합니다.
5. 스택 옵션 구성 페이지에서는 모든 기본 옵션을 그대로 둡니다. 계속하려면 다음을 클릭합니다.
6. 스택 검토 페이지에서 데이터베이스 및 Linux Bastion Host 옵션을 확인한 후 제출을 선택합니다.

스택 생성 프로세스가 완료되면 BastionStack 및 APGNS라는 이름의 스택을 보고 데이터베이스에 연결하는 데 필요한 정보를 기록해 둡니다. 자세한 내용은 [AWS Management Console에서 AWS CloudFormation 스택 데이터 및 리소스 보기](#)를 참조하세요.

3단계: Aurora PostgreSQL DB 클러스터에 연결

표준 PostgreSQL 클라이언트 애플리케이션을 사용해 DB 클러스터에 연결할 수 있습니다. 이 예시에서는 psql 명령줄 클라이언트를 사용하여 Aurora PostgreSQL DB 클러스터에 연결합니다.

Aurora PostgreSQL DB 클러스터에 연결하려면

1. DB 클러스터에 대한 라이트 인스턴스의 엔드포인트(DNS 이름)와 포트 번호를 찾습니다.

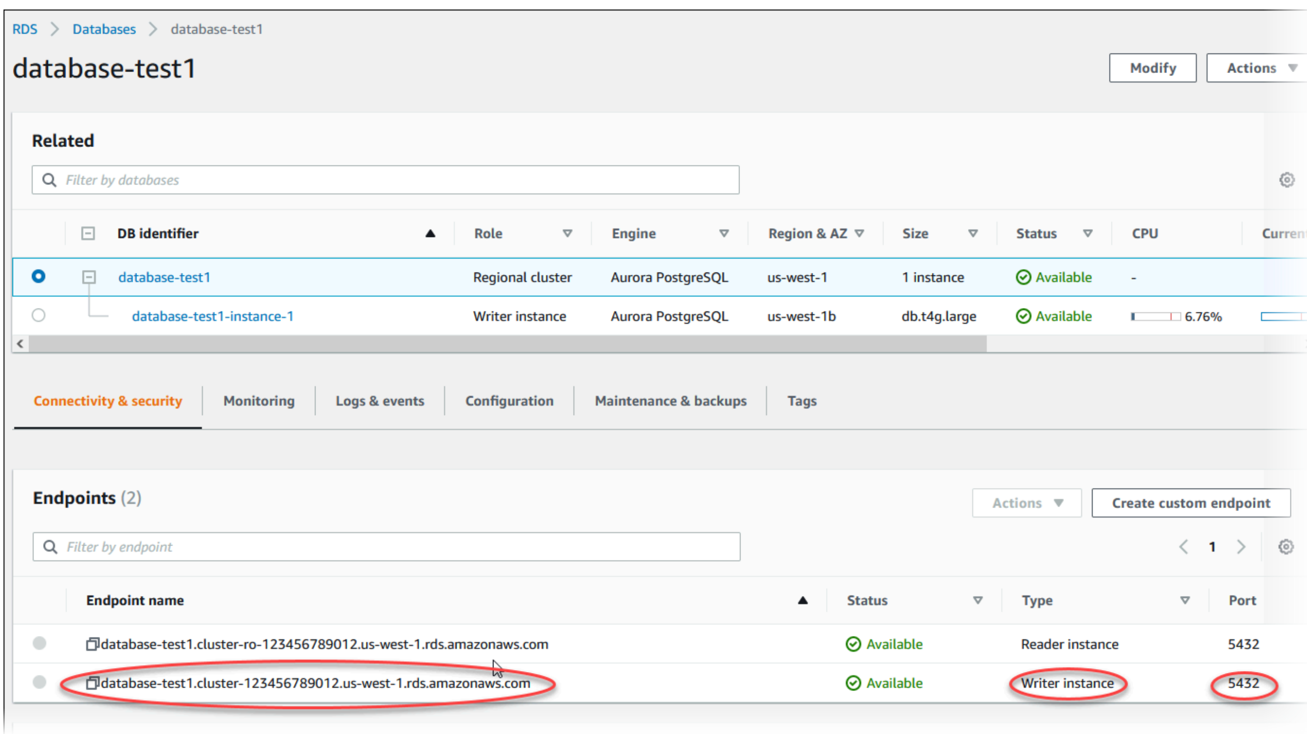
   a. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

   b. Amazon RDS 콘솔의 오른쪽 상단에서 DB 클러스터의 AWS 리전을 선택합니다.

   c. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

   d. Aurora PostgreSQL DB 클러스터 이름을 선택하여 세부 정보를 표시합니다.

   e. 연결 및 보안 탭에서 라이터 인스턴스의 엔드포인트를 복사합니다. 또한 포트 번호를 적어 둡니다. DB 클러스터에 연결하려면 엔드포인트와 포트 번호가 모두 필요합니다.



2. Amazon EC2 사용 설명서에 있는 [Linux 인스턴스에 연결]의 단계를 따라 앞에서 만든 EC2 인스턴스에 연결합니다.

   SSH를 사용하여 EC2 인스턴스에 연결하는 것이 좋습니다. Windows, Linux 또는 Mac에 SSH 클라이언트 유틸리티가 설치된 경우 다음 명령 형식을 사용하여 인스턴스에 연결할 수 있습니다.

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   예를 들어 `ec2-database-connect-key-pair.pem`이 Linux의 `/dir1`에 저장되어 있고, EC2 인스턴스의 퍼블릭 IPv4 DNS가 `ec2-12-345-678-90.compute-1.amazonaws.com`이라고 가정해 보겠습니다. SSH 명령은 다음과 같이 표시됩니다.

   ```
   ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
   user@ec2-12-345-678-90.compute-1.amazonaws.com
   ```

3. EC2 인스턴스에서 소프트웨어를 업데이트하여 최신 버그 수정 및 보안 업데이트를 받습니다. 이렇게 하려면 다음 명령을 사용합니다.

   > **Note**
   >
   > `-y` 옵션을 사용하면 확인 여부를 묻지 않고 업데이트를 설치합니다. 설치 전에 업데이트 정보를 확인하려면 이 옵션을 생략합니다.

   ```
   sudo dnf update -y
   ```

4. 다음 명령을 사용하여 Amazon Linux 2023의 PostgreSQL에서 psql 명령줄 클라이언트를 설치합니다.

   ```
   sudo dnf install postgresql15
   ```

5. Aurora PostgreSQL DB 클러스터에 연결합니다. 예를 들어, 다음 명령을 입력합니다. 이 작업을 수행하면 psql 클라이언트를 사용하여 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

   *`endpoint`*는 라이터 인스턴스의 엔드포인트로 대체하고, *`postgres`*는 연결하려는 데이터베이스 이름 `--dbname`으로 대체하고, *`postgres`*는 사용된 마스터 사용자 이름으로 대체합니다. 암호를 묻는 메시지가 표시되면 사용한 마스터 암호를 제공합니다.

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

사용자에 대한 암호를 입력하면 다음과 유사한 출력이 나타납니다.

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

Aurora PostgreSQL DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora PostgreSQL DB 클러스터에 연결 섹션을 참조하세요. DB 클러스터에 연결할 수 없는 경우 Amazon RDS DB 인스턴스에 연결할 수 없음 섹션을 참조하세요.

보안을 위해서는 암호화된 연결을 사용하는 것이 가장 좋습니다. 클라이언트와 서버가 동일한 VPC에 있고 네트워크를 신뢰할 수 있는 경우에만 암호화되지 않은 PostgreSQL 연결을 사용합니다. 암호화된 연결 사용에 대한 자세한 내용은 SSL/TLS를 이용한 Aurora PostgreSQL 데이터 보안 섹션을 참조하세요.

6. SQL 명령을 실행합니다.

   예를 들어, 다음 SQL 명령은 현재 날짜 및 시간을 보여줍니다.

```
SELECT CURRENT_TIMESTAMP;
```

4단계: EC2 인스턴스 및 DB 클러스터 삭제

생성한 샘플 EC2 인스턴스 및 DB 클러스터에 연결하고 탐색한 후에는 요금이 더 이상 부과되지 않도록 삭제합니다.

AWS CloudFormation을 사용하여 리소스를 생성했다면 이 단계를 건너뛰고 다음 단계로 이동합니다.

EC2 인스턴스를 삭제하는 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.

3. EC2 인스턴스를 선택하고 인스턴스 상태, 인스턴스 종료를 차례로 선택합니다.
4. 확인 메시지가 나타나면 종료를 선택합니다.

EC2 인스턴스 삭제에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료를 참조하세요.

DB 클러스터를 삭제하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 DB 클러스터에 연결된 DB 인스턴스를 선택합니다.
3. [Actions]에 대해 [Delete]를 선택합니다.
4. 삭제를 선택합니다.

   DB 클러스터에 연결된 모든 DB 인스턴스가 삭제되고 나면 DB 클러스터가 자동으로 삭제됩니다.

(선택 사항) CloudFormation으로 생성한 EC2 인스턴스 및 DB 클러스터 삭제

AWS CloudFormation을 사용하여 리소스를 생성한 경우 샘플 EC2 인스턴스 및 DB 클러스터에 연결하고 탐색한 후 CloudFormation 스택을 삭제하여 더 이상 비용이 청구되지 않도록 합니다.

CloudFormation 리소스를 삭제하려면

1. AWS CloudFormation 콘솔을 엽니다.
2. CloudFormation 콘솔의 스택 페이지에서 루트 스택(VPCStack, BastionStack 또는 APGNS라는 이름이 없는 스택)을 선택합니다.
3. 삭제를 선택합니다.
4. 확인 메시지가 나타나면 스택 삭제를 선택합니다.

CloudFormation에서 스택을 삭제하는 방법에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation 콘솔에서 스택 삭제를 참조하세요.

(선택 사항) DB 클러스터를 Lambda 함수에 연결

Aurora PostgreSQL DB 클러스터를 Lambda 서버리스 컴퓨팅 리소스에 연결할 수도 있습니다. Lambda 함수를 사용하면 인프라를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있습니다. 또

한 하루 12개에서 초당 수백 개에 이르는 모든 규모의 코드 실행 요청에 자동으로 응답할 수 있습니다. 자세한 내용은 Lambda 함수와 Aurora DB 클러스터 자동 연결 섹션을 참조하세요.

자습서: 웹 서버 및 Amazon Aurora DB 클러스터 생성

이 자습서에서는 PHP가 있는 Apache 웹 서버를 설치하고 MariaDB, MySQL 또는 PostgreSQL 데이터베이스를 생성하는 방법을 보여줍니다. 이 웹 서버는 Amazon Linux 2023을 사용하여 Amazon EC2 인스턴스에서 실행되며, Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터 중에서 선택할 수 있습니다. Amazon EC2 인스턴스와 DB 클러스터 모두 Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)에서 실행됩니다.

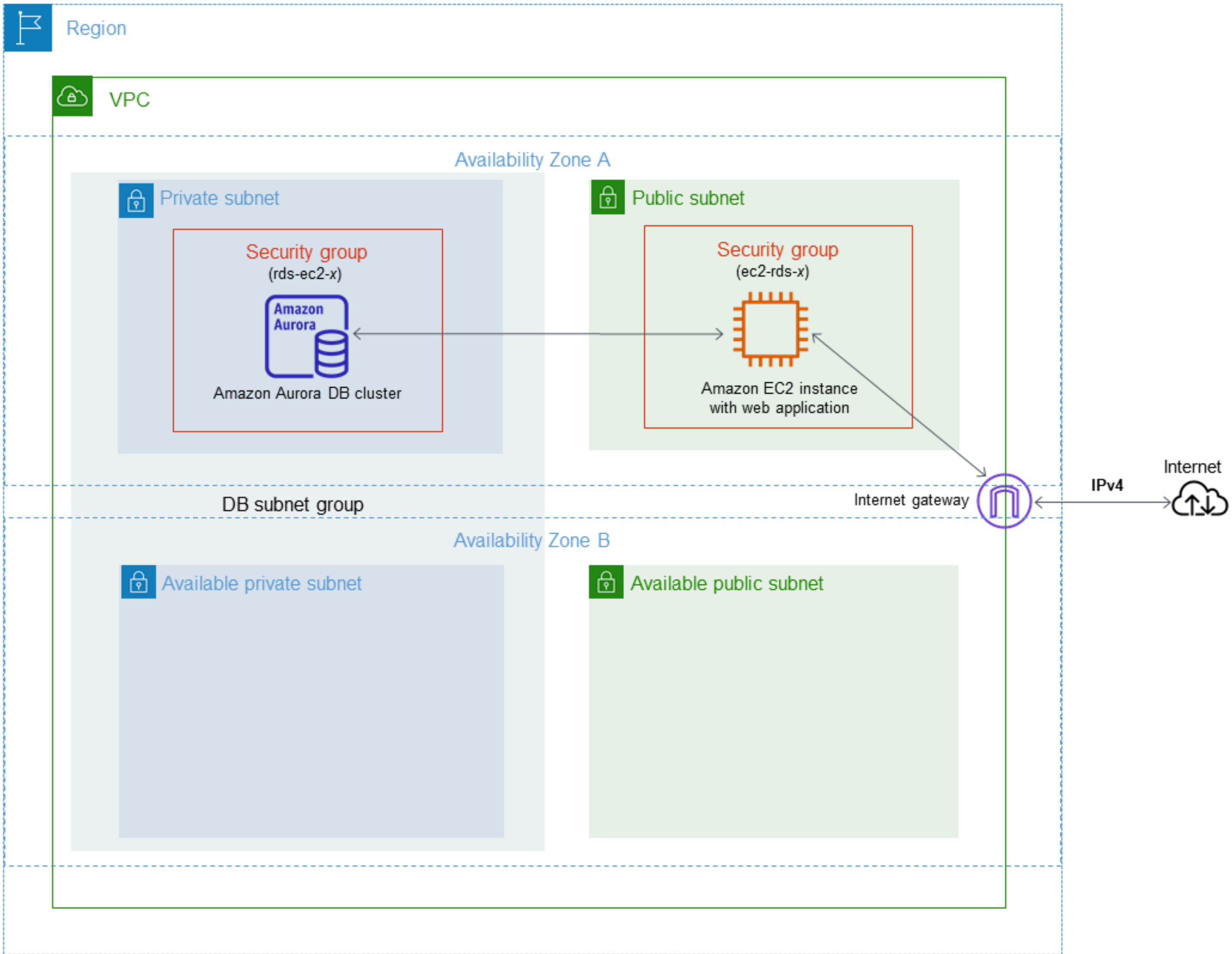
> **Important**
>
> AWS 계정 생성은 무료입니다. 그러나 이 자습서를 완료하면 사용하는 AWS 리소스에 대한 비용이 발생할 수 있습니다. 자습서가 더 이상 필요하지 않은 경우 자습서를 완료한 후에 이러한 리소스를 삭제할 수 있습니다.

> **Note**
>
> 이 자습서에서는 Amazon Linux 2023을 사용하여 작업하며 다른 버전의 Linux는 적용되지 않을 수 있습니다.

다음 자습서에서는 AWS 계정의 .NET용 기본 VPC, 서브넷 및 보안 그룹을 사용하는 EC2 인스턴스를 생성합니다. 이 자습서에서는 DB 클러스터터를 생성하고 생성한 EC2 인스턴스와의 연결을 자동으로 설정하는 방법을 보여줍니다. 그런 다음 자습서에서는 EC2 인스턴스에 웹 서버를 설치하는 방법을 보여줍니다. DB 클러스터 라이터 엔드포인트를 사용하여 VPC의 DB 클러스터에 웹 서버를 연결합니다.

1. EC2 인스턴스를 시작하여 DB 클러스터에 연결
2. Amazon Aurora DB 클러스터 생성
3. EC2 인스턴스에 웹 서버 설치

다음 다이어그램은 이 자습서를 완료했을 때 구성을 보여 줍니다.



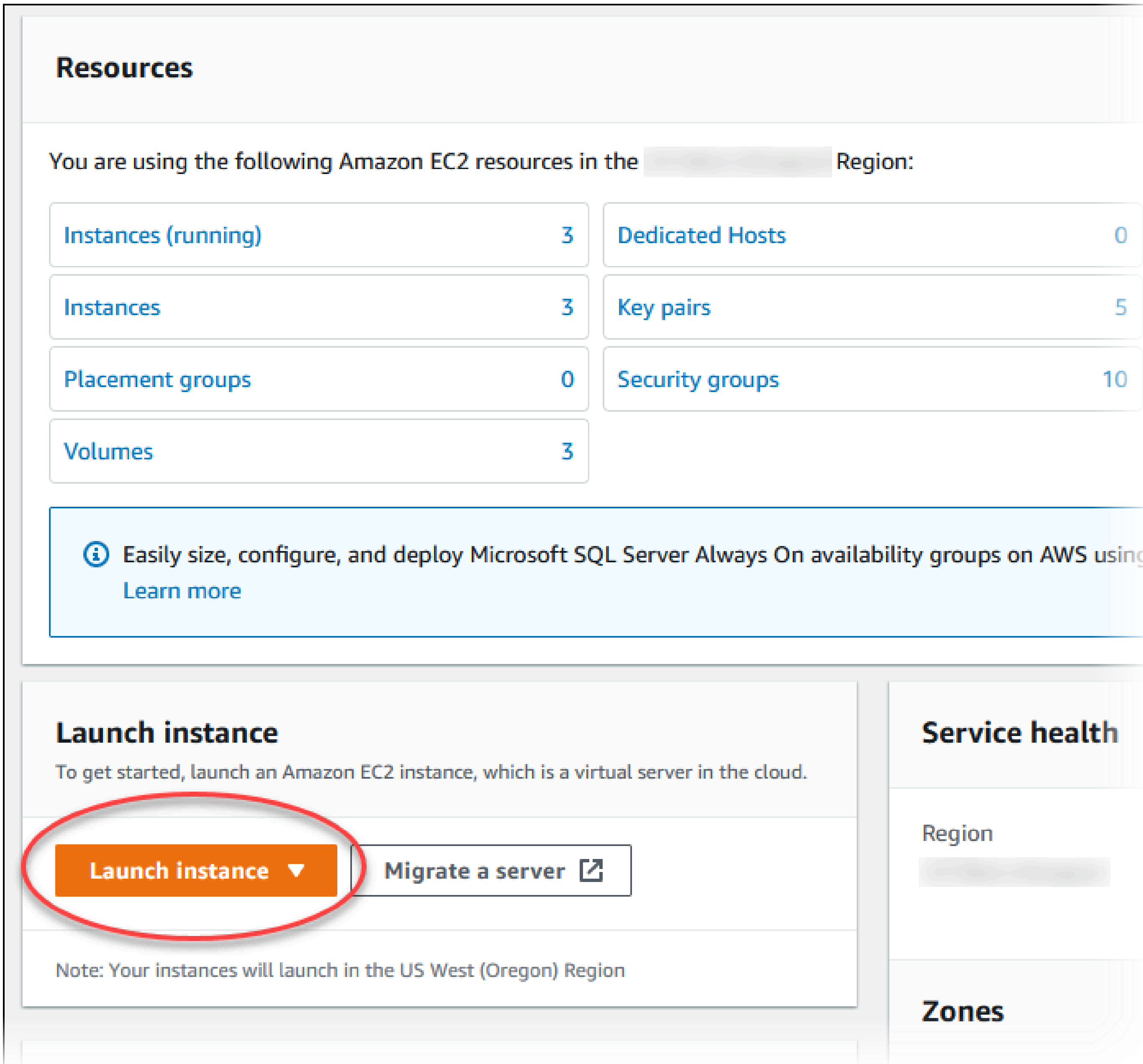
> **Note**
>
> 자습서를 완료하면 VPC의 각 가용 영역에 퍼블릭 서브넷과 프라이빗 서브넷이 있을 것입니다. 이 자습서에서는 AWS 계정에 대한 기본 VPC를 사용하고 EC2 인스턴스와 DB 클러스터 간의 연결을 자동으로 설정합니다. 대신 이 시나리오에 맞게 새 VPC 구성하려면 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)에서 작업을 완료하세요.

EC2 인스턴스를 시작하여 DB 클러스터에 연결

VPC의 퍼블릭 서브넷에서 Amazon EC2 인스턴스를 생성합니다.

EC2 인스턴스 시작

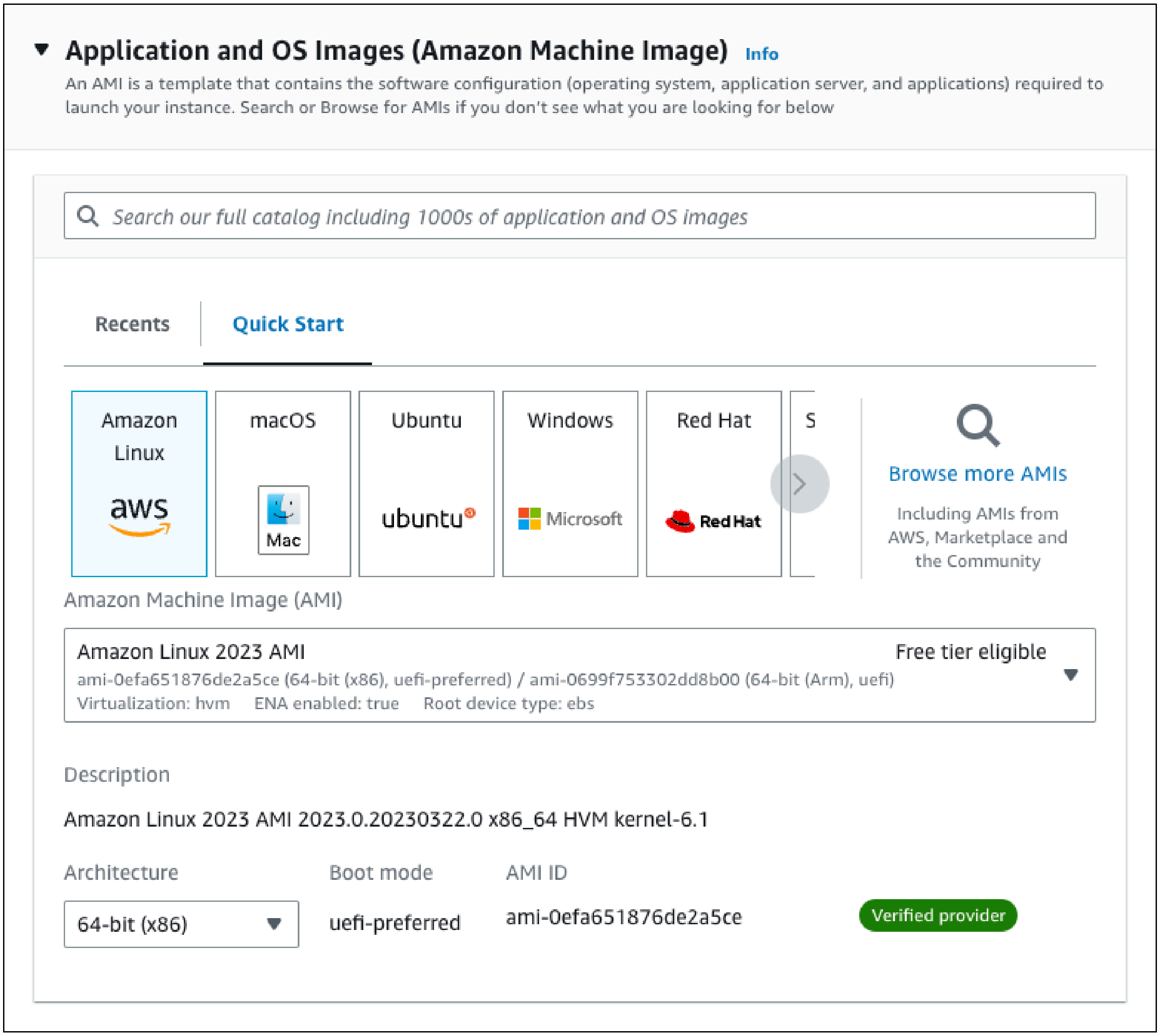
1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. AWS Management Console 콘솔의 오른쪽 상단에서 EC2 인스턴스를 생성하려는 AWS 리전을 선택합니다.
3. 다음과 같이 EC2 대시보드를 선택한 다음, 인스턴스 시작을 선택합니다.



4. 인스턴스 시작 페이지에서 다음 설정을 선택합니다.

a. Name and tags(이름 및 태그) 아래의 Name(이름)에 **`tutorial-ec2-instance-web-server`**을 입력하세요.

b. Application and OS Images (Amazon Machine Image)(애플리케이션 및 OS 이미지(Amazon Machine Image))에서 Amazon Linux를 선택한 다음 Amazon Linux 2023 AMI를 선택합니다. 다른 선택 항목에 대해서는 기본값을 그대로 유지합니다.



c. 인스턴스 유형에서 t2.micro를 선택합니다.

d. 키 페어(로그인)에서 기존 키 페어를 사용할 키 페어 이름을 선택합니다. Amazon EC2 인스턴스에 대한 새 키 페어를 생성하려면 새 키 페어 생성을 선택한 다음 키 페어 생성 창을 사용하여 생성합니다.

새 키 페어 생성에 대한 자세한 내용은 Amazon EC2 Linux 인스턴스용 사용 설명서의 키 페어 생성을 참조하세요.

e. 네트워크 설정에서 다음과 같이 이러한 값을 설정하고 다른 값은 기본값으로 유지합니다.

- SSH 트래픽 허용에서 EC2 인스턴스에 대한 SSH 연결 소스를 선택합니다.

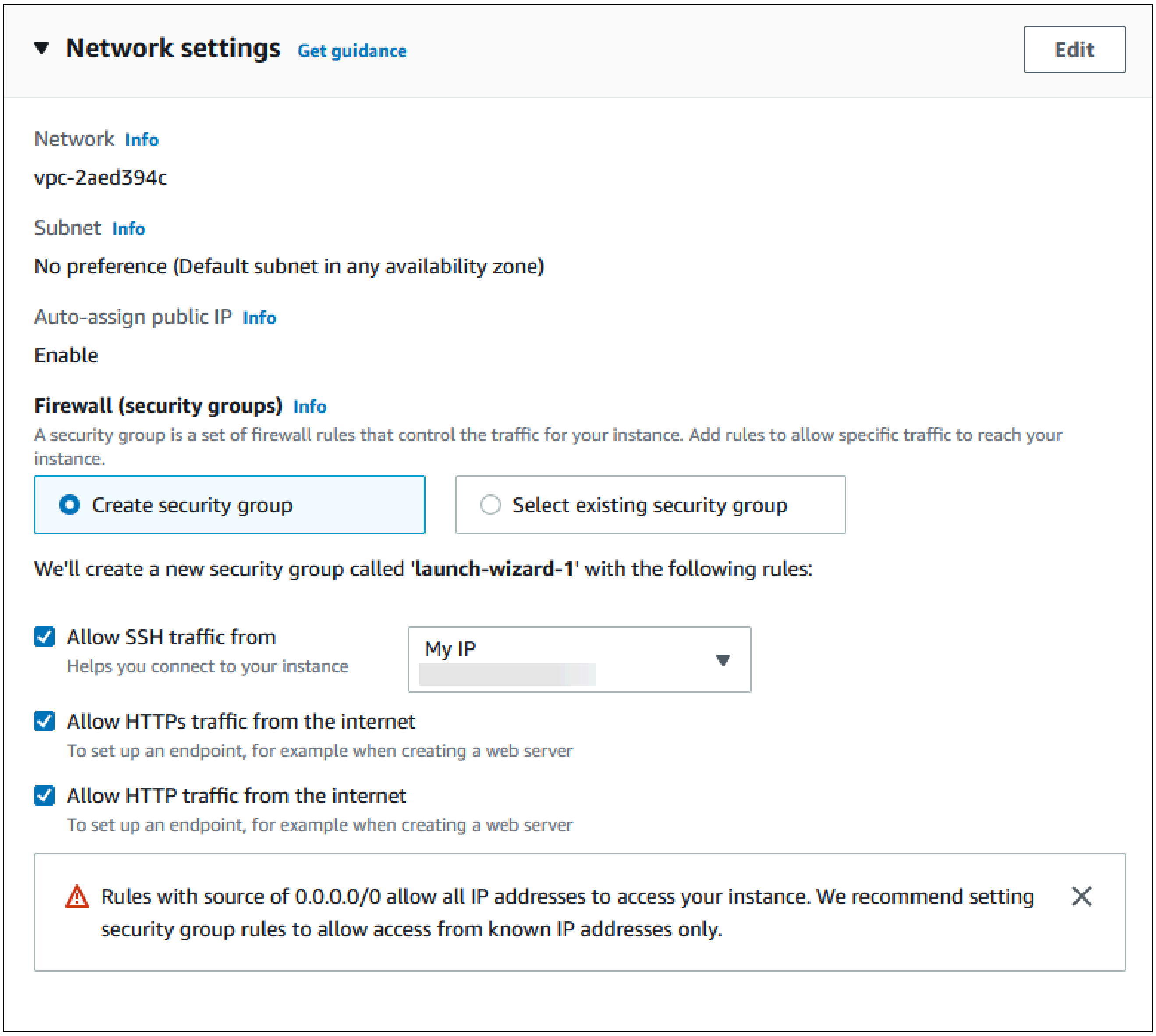
  표시된 IP 주소가 SSH 연결에 대해 올바른 경우 내 IP를 선택할 수 있습니다.

  그렇지 않으면 SSH(Secure Shell)를 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 결정할 수 있습니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다. IP 주소의 예는 `203.0.113.25/32`입니다.

  대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 결정합니다.

  > **Warning**
  > SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

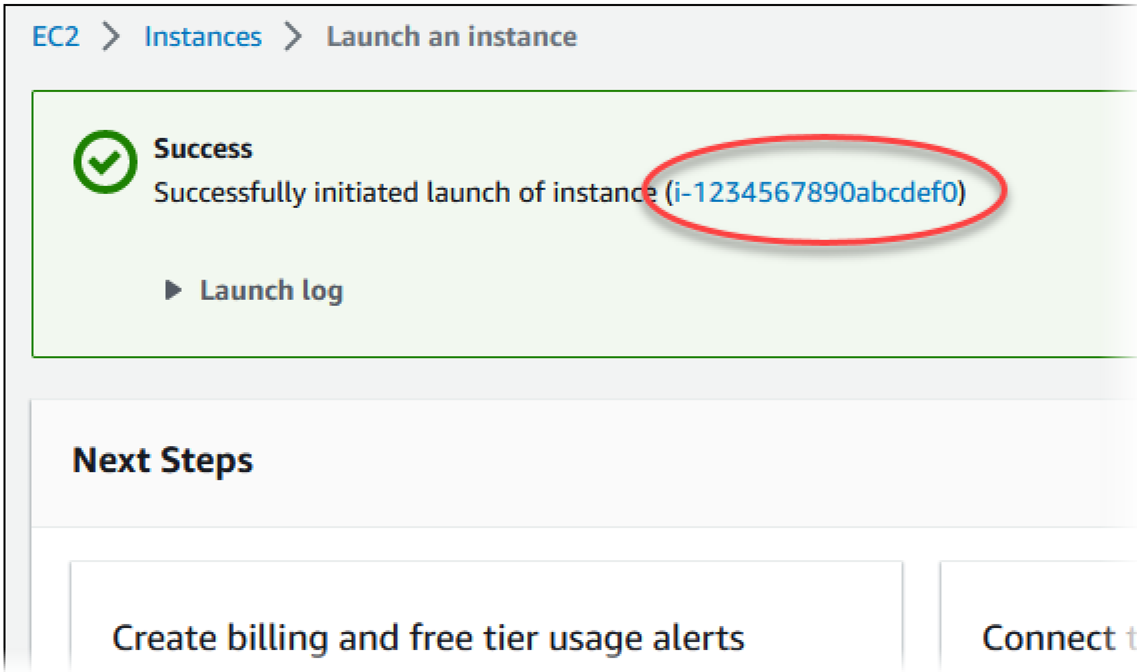
- 인터넷에서 오는 HTTPS 트래픽 허용을 켭니다.
- 인터넷에서 오는 HTTP 트래픽 허용을 켭니다.



f. 나머지 섹션에서 기본값은 그대로 둡니다.

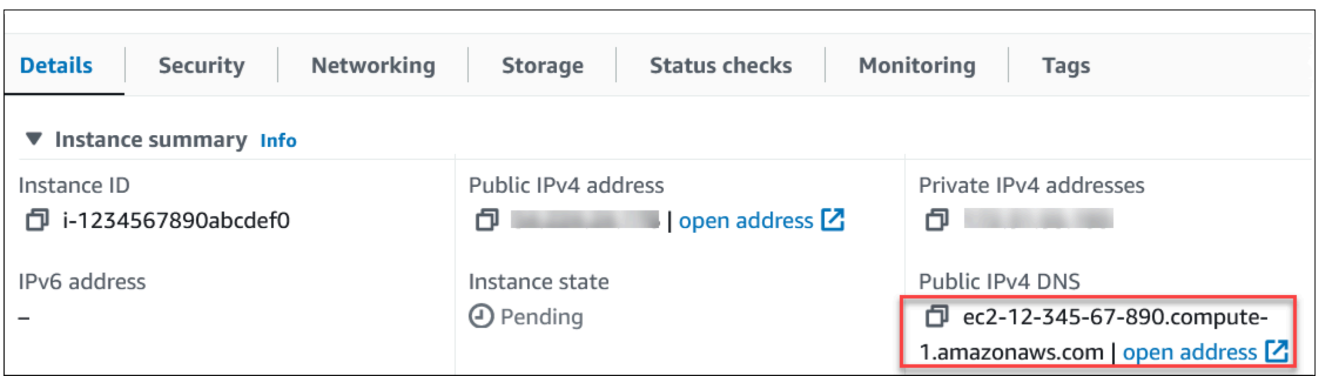
g. 요약 패널에서 인스턴스 구성 요약을 검토하고 준비가 되면 인스턴스 시작을 선택합니다.

5. 시작 상태 페이지에서, 새 EC2 인스턴스의 식별자(예: `i-1234567890abcdef0`)를 기록해 둡니다.

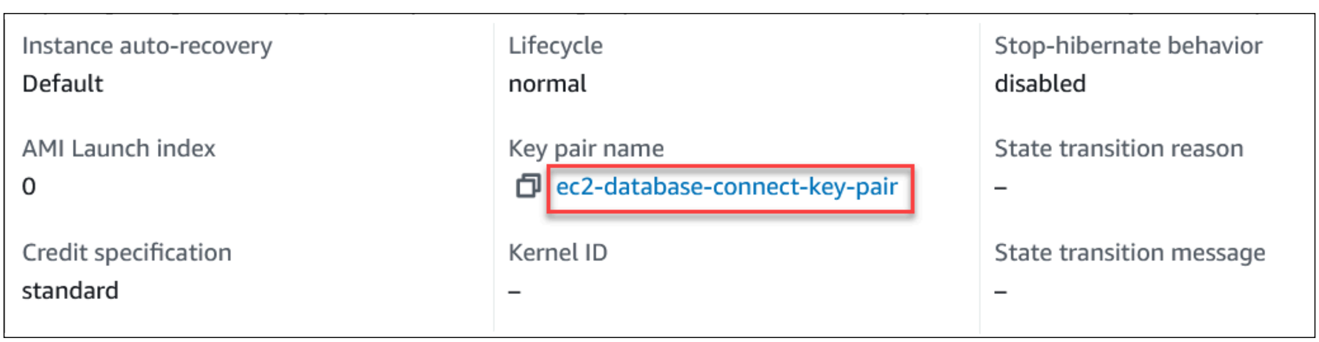


6. EC2 인스턴스 식별자를 선택하여 EC2 인스턴스 목록을 열고 EC2 인스턴스를 선택합니다.
7. 세부 정보 탭에서 SSH를 사용하여 연결할 때 필요한 다음 값을 기록해 둡니다.

    a. 인스턴스 요약에서 퍼블릭 IPv4 DNS의 값을 기록해 둡니다.



    b. 인스턴스 세부 정보에서 키 페어 이름의 값을 기록해 둡니다.



8. 인스턴스의 인스턴스 상태가 실행 중으로 읽힐 때까지 기다린 다음 계속합니다.

9. [Amazon Aurora DB 클러스터 생성](#)을 완료합니다.

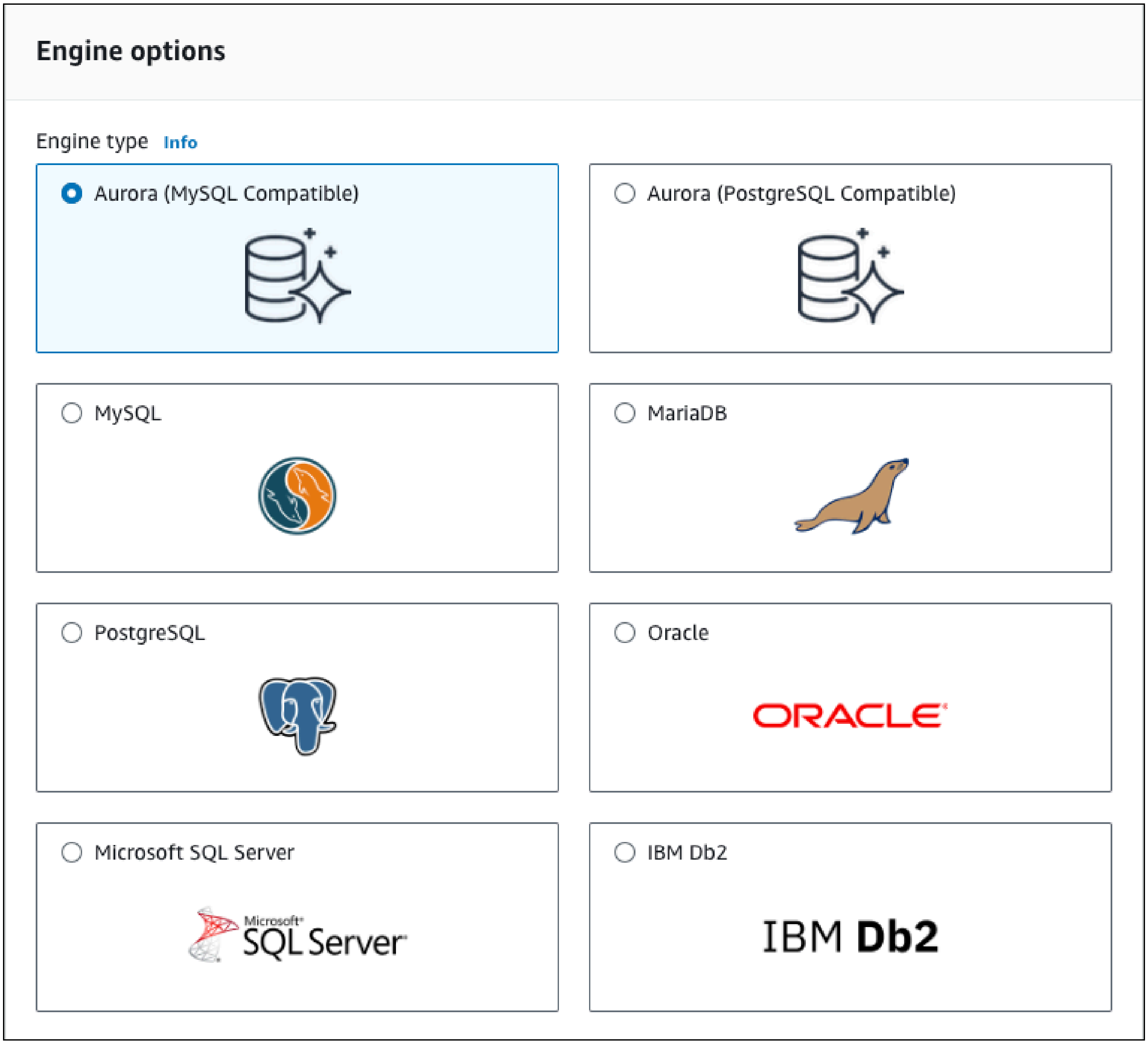
Amazon Aurora DB 클러스터 생성

웹 애플리케이션에서 사용되는 데이터를 유지 관리하는 Amazon Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터를 생성합니다.

Aurora MySQL

Aurora MySQL DB 클러스터를 생성하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단 모서리에서 AWS 리전이 EC2 인스턴스를 생성한 것과 동일한지 확인합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택합니다.
5. 데이터베이스 생성 페이지에서 표준 생성을 선택합니다.
6. 엔진 옵션에서 Aurora(MySQL 호환)를 선택합니다.

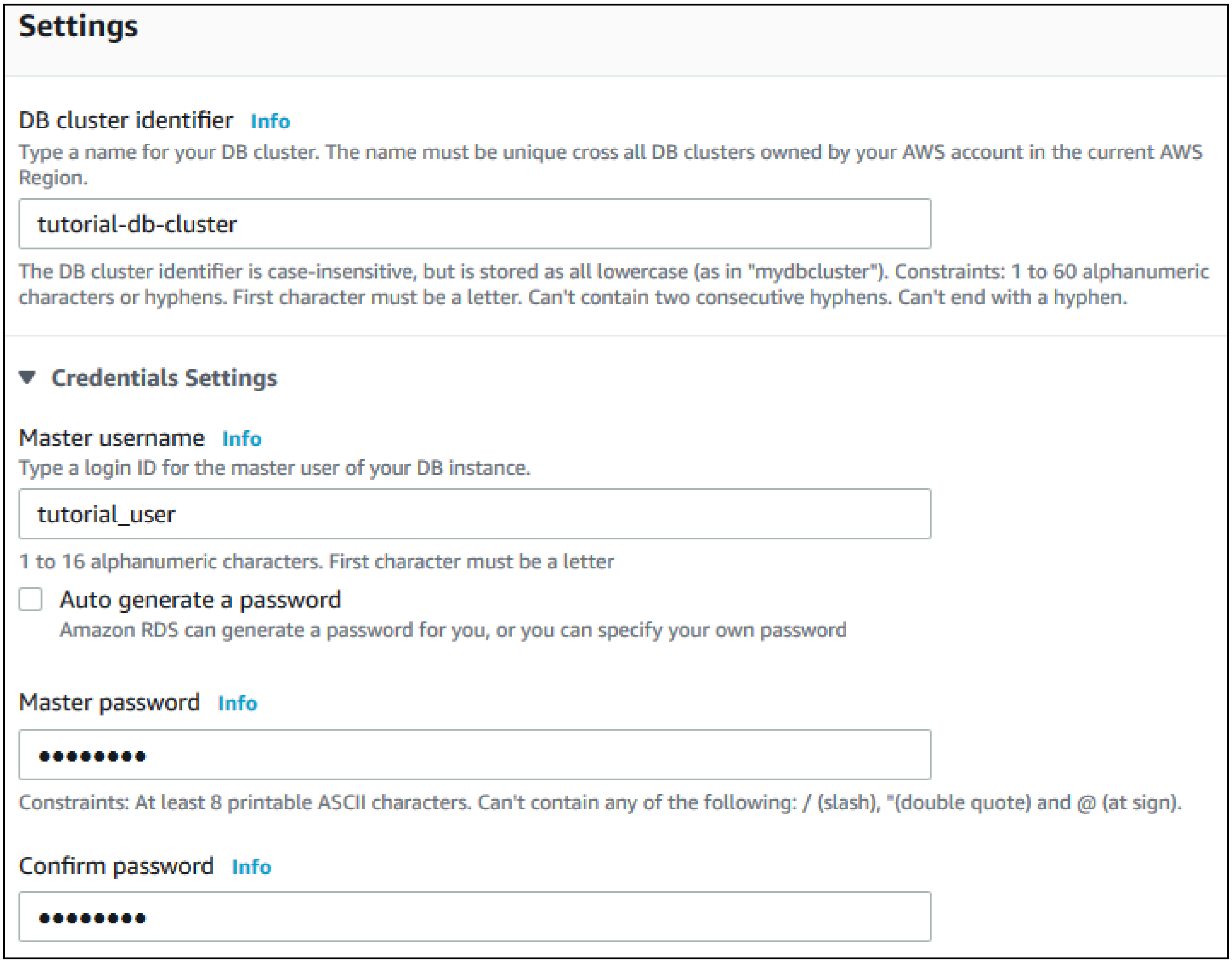


[버전(Version)] 및 기타 엔진 옵션의 기본값을 유지합니다.

7. 템플릿 섹션에서 개발/테스트를 선택합니다.

8. 설정 섹션에서 이러한 값들을 설정합니다.

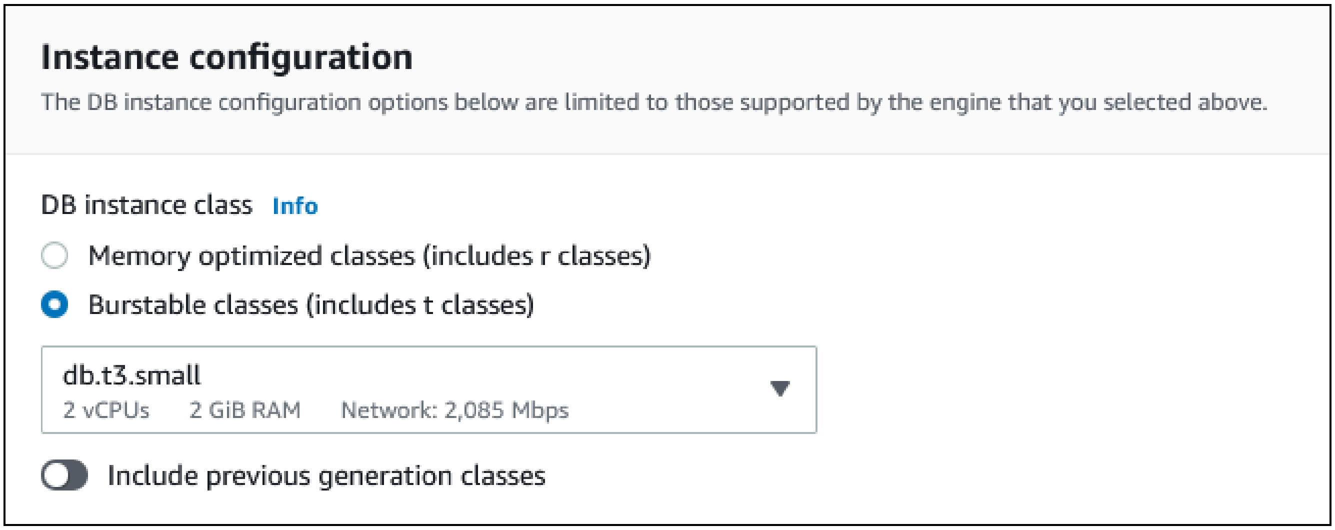
   - DB 클러스터 식별자 – **`tutorial-db-cluster`**를 입력합니다.
   - 마스터 사용자 이름 - **`tutorial_user`**를 입력합니다.
   - Auto generate a password(암호 자동 생성) - 옵션을 끈 상태로 둡니다.
   - 마스터 암호 – 암호를 입력합니다.
   - 암호 확인 – 암호를 다시 입력합니다.



9. 인스턴스 구성] 섹션에서 다음 값을 설정합니다.

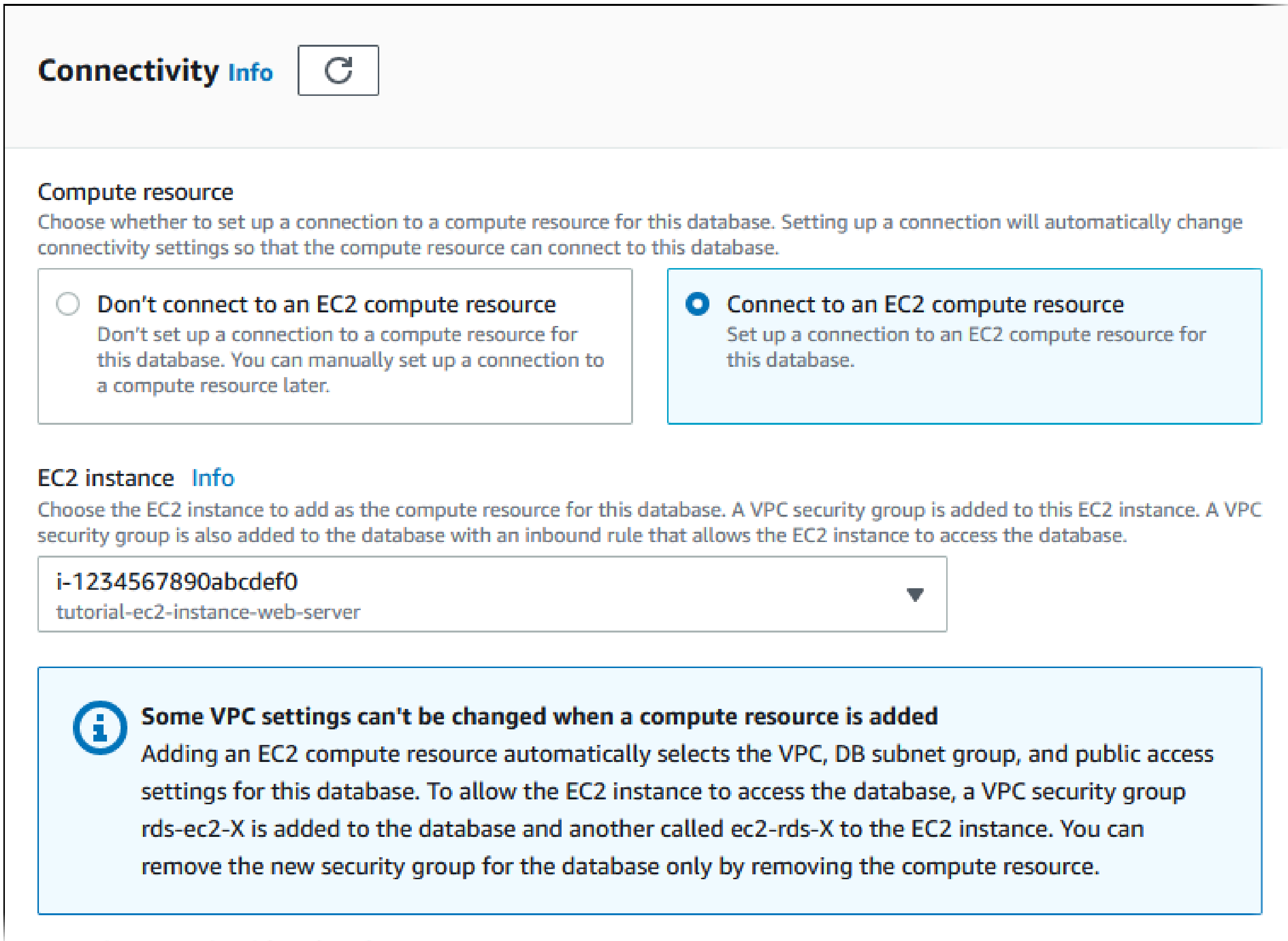
   - 버스트 가능 클래스(t 클래스 포함)
   - db.t3.small 또는 db.t3.medium

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.



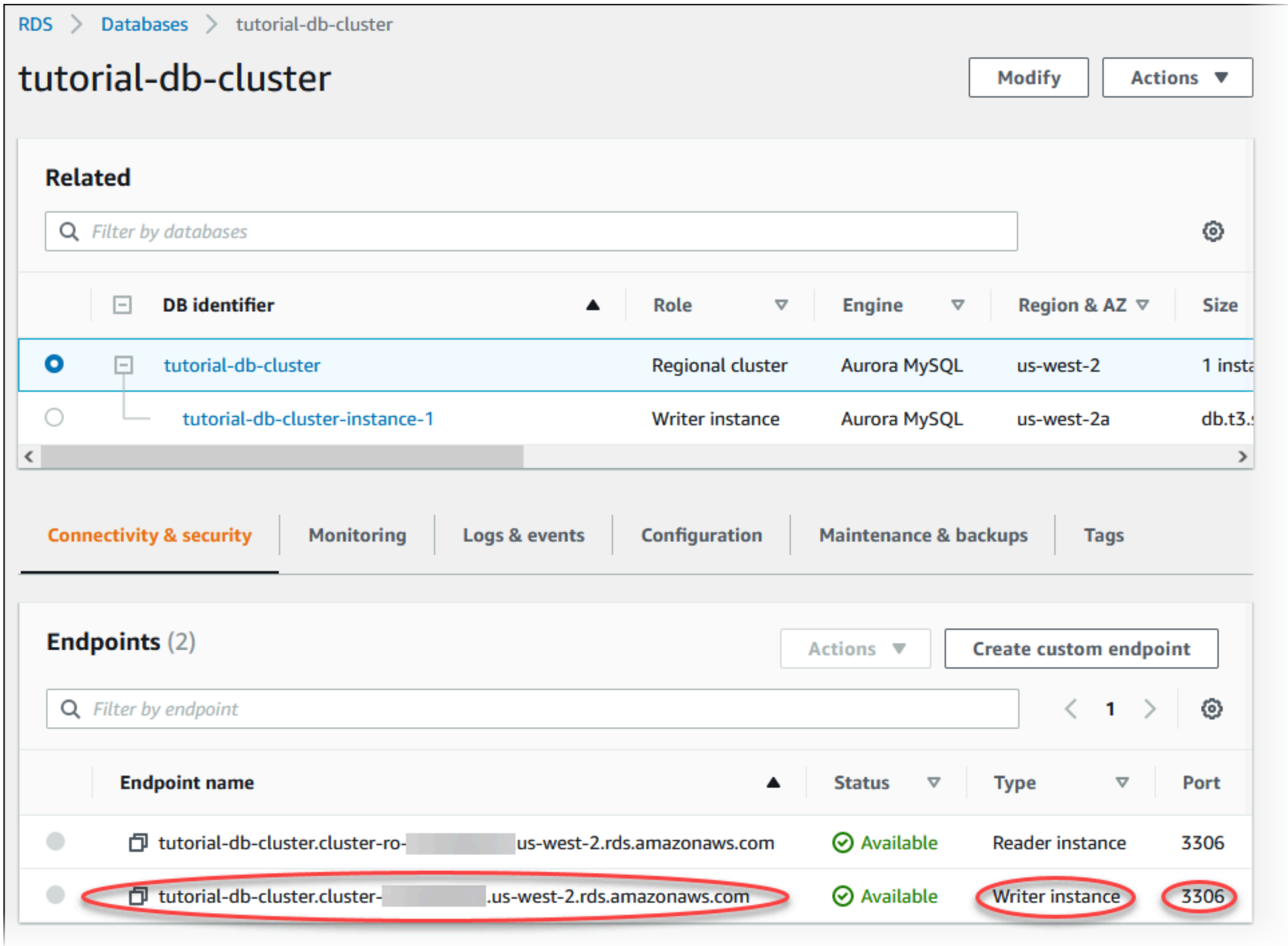
10. 가용성 및 내구성 섹션에서 기본값을 사용합니다.
11. 연결 섹션에서 다음 값을 설정하고 다른 값을 기본값으로 유지합니다.

    - 컴퓨팅 리소스에서 EC2 컴퓨팅 리소스에 연결을 선택합니다.
    - EC2 인스턴스의 경우 tutorial-ec2-instance-web-server와 같이 이전에 생성한 EC2 인스턴스를 선택합니다.



12. Additional configuration(추가 구성) 섹션을 열고 Initial database name(초기 데이터베이스 이름)에 **`sample`**를 입력합니다. 다른 옵션은 기본 설정을 유지합니다.
13. Aurora MySQL DB 클러스터를 생성하려면 Create database(데이터베이스 생성)를 선택합니다.

    새 DB 클러스터가 데이터베이스목록에 생성 중의 상태로 나타납니다.
14. 새 DB 클러스터의 상태가 사용 가능으로 나타날 때까지 기다립니다. 그런 다음, 세부 정보를 표시할 DB 클러스터 이름을 선택합니다.
15. Connectivity & security(연결 및 보안) 섹션에서 라이터 DB 인스턴스의 엔드포인트 및 포트를 확인합니다.



라이터 DB 인스턴스의 엔드포인트와 포트를 적어 둡니다. 이 정보를 사용하여 웹 서버를 DB 클러스터에 연결하게 됩니다.

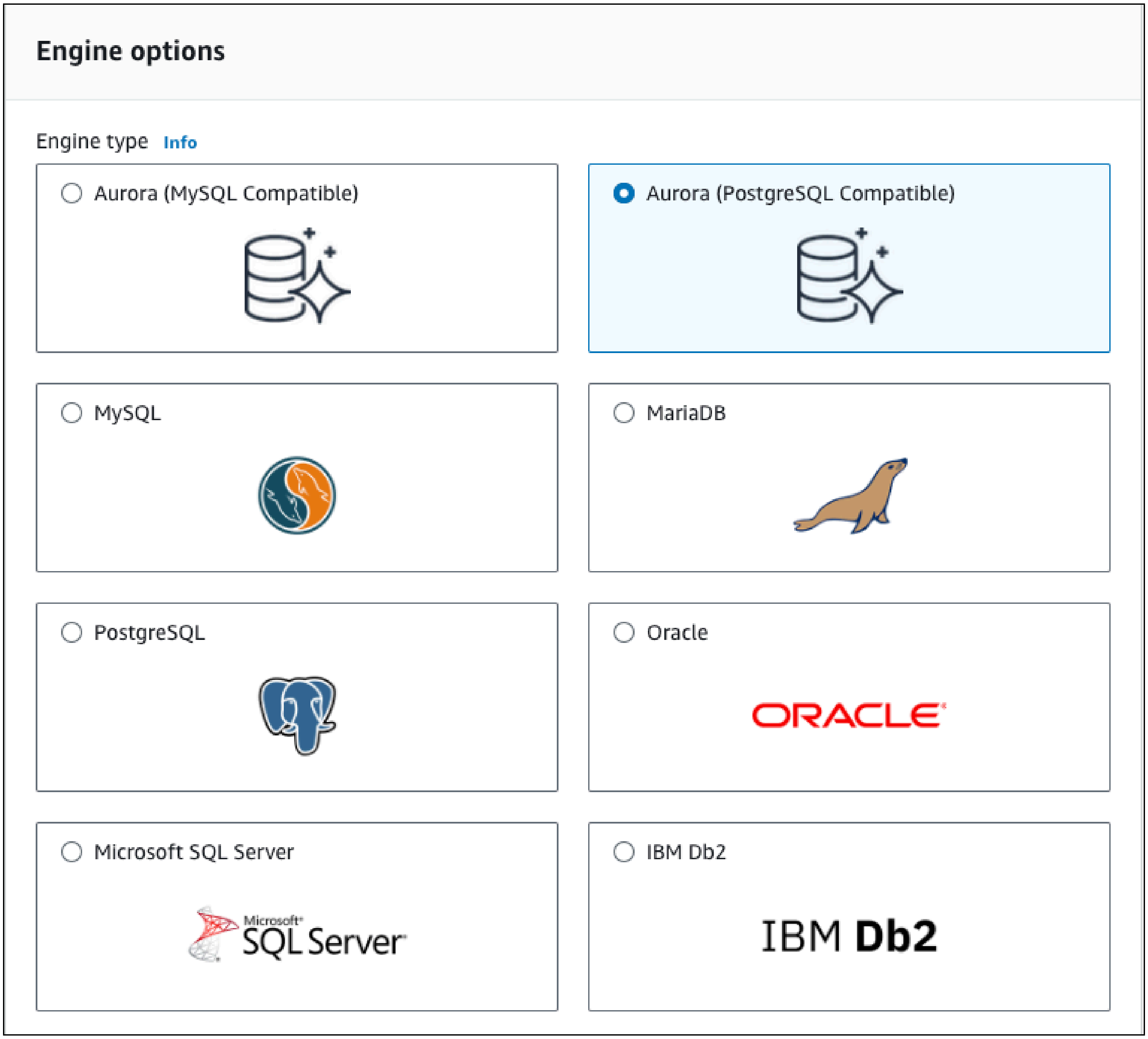
16. EC2 인스턴스에 웹 서버 설치를 완료합니다.

Aurora PostgreSQL

Aurora PostgreSQL DB 클러스터를 생성하려면

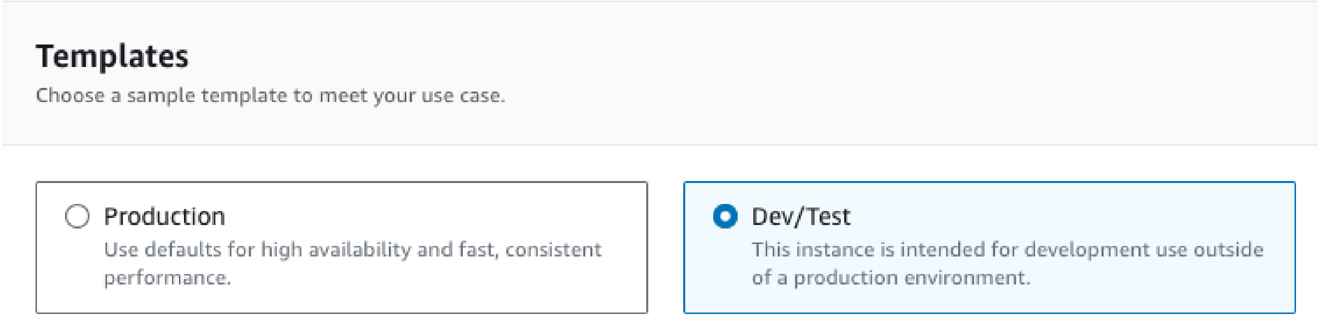
1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단 모서리에서 AWS 리전이 EC2 인스턴스를 생성한 것과 동일한지 확인합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택합니다.

5. 데이터베이스 생성 페이지에서 표준 생성을 선택합니다.
6. 엔진 옵션에서 Aurora(PostgreSQL 호환)를 선택합니다.



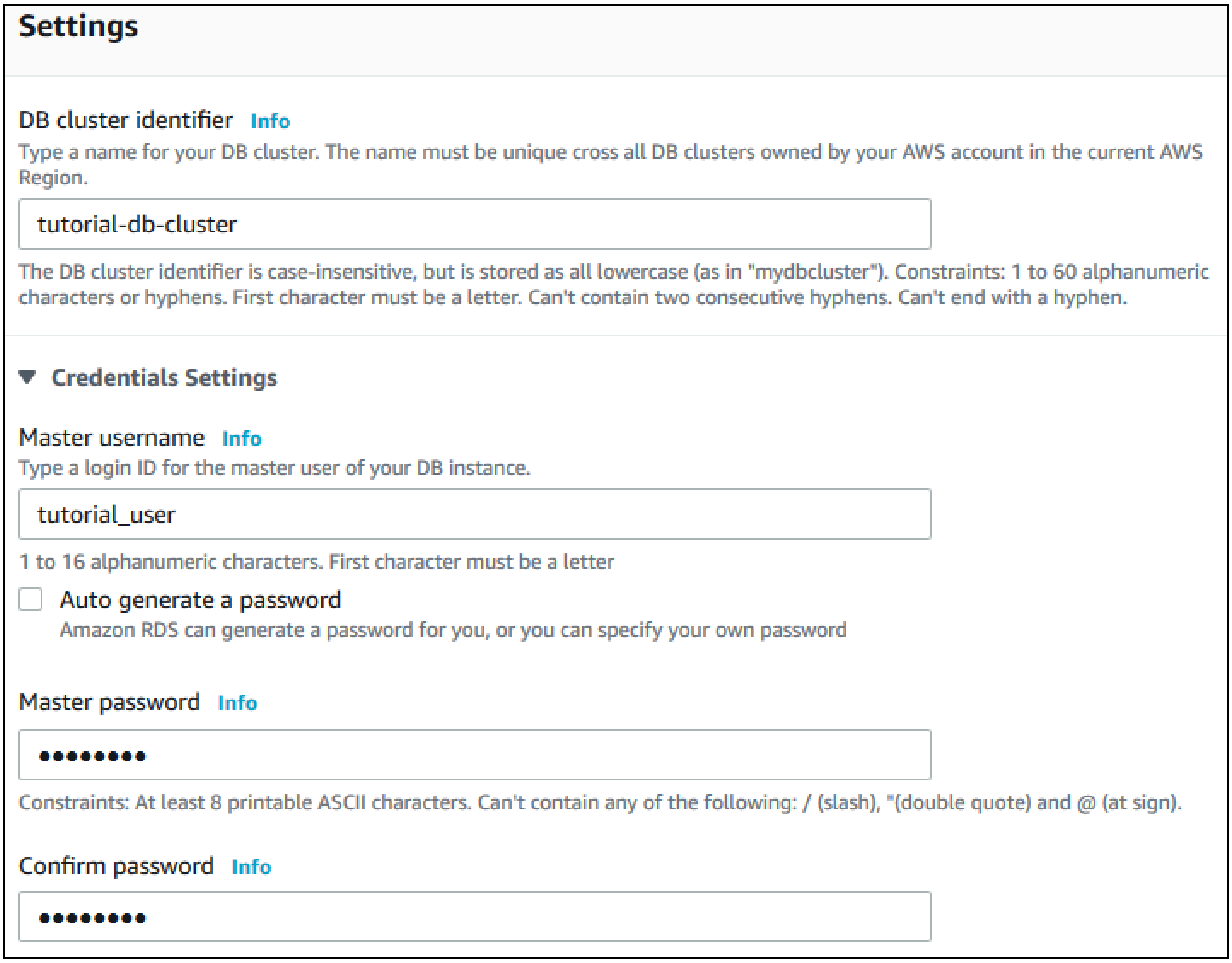
[버전(Version)] 및 기타 엔진 옵션의 기본값을 유지합니다.

7. 템플릿 섹션에서 개발/테스트를 선택합니다.



8. 설정 섹션에서 이러한 값들을 설정합니다.

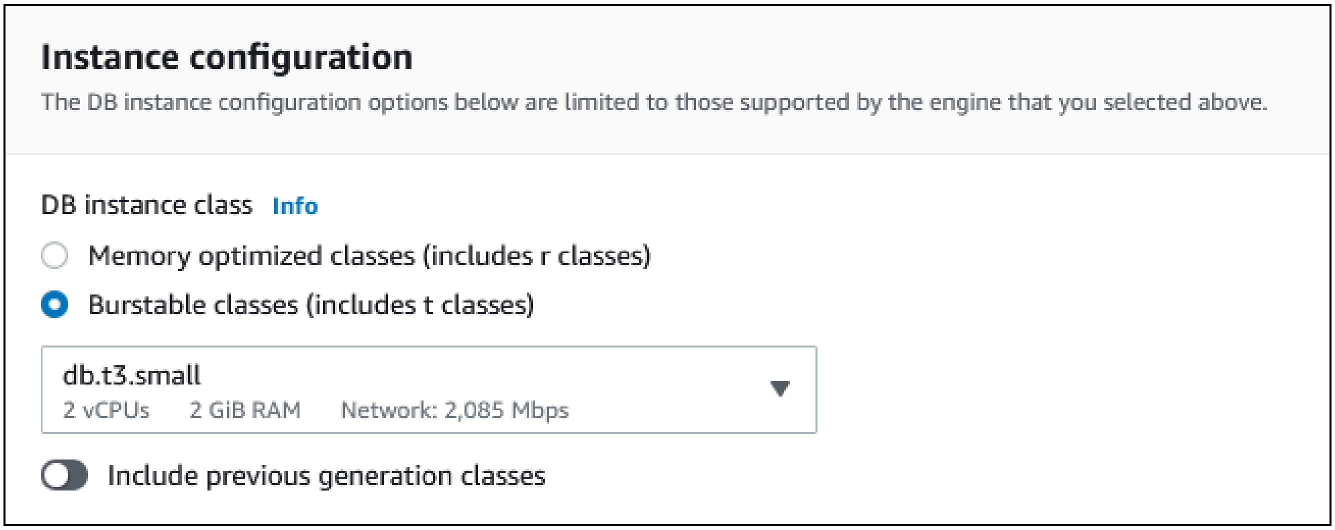
   - DB 클러스터 식별자 – **`tutorial-db-cluster`**를 입력합니다.
   - 마스터 사용자 이름 - **`tutorial_user`**를 입력합니다.
   - Auto generate a password(암호 자동 생성) - 옵션을 끈 상태로 둡니다.
   - 마스터 암호 – 암호를 입력합니다.
   - 암호 확인 – 암호를 다시 입력합니다.



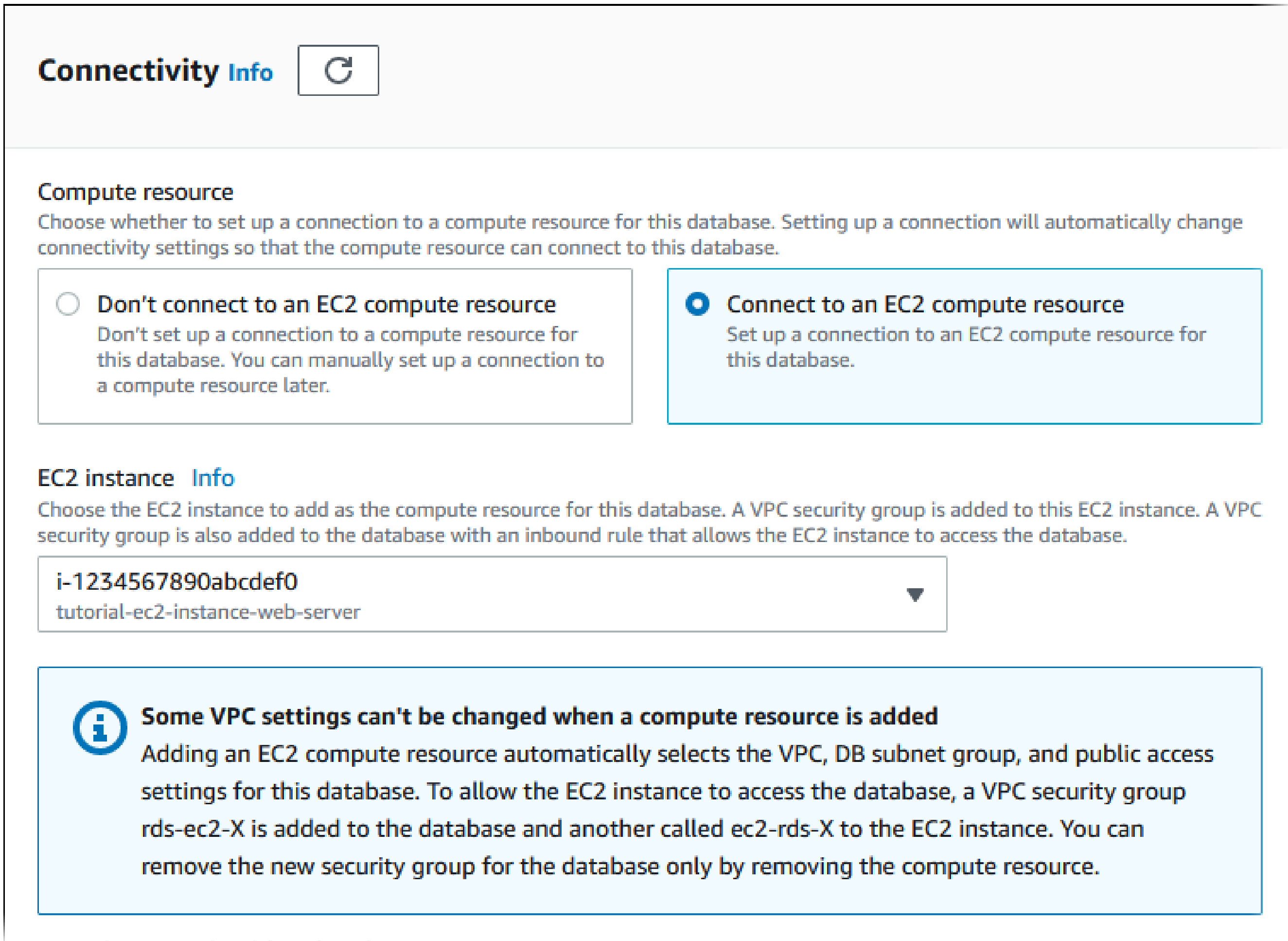
9. 인스턴스 구성] 섹션에서 다음 값을 설정합니다.

- 버스트 가능 클래스(t 클래스 포함)
- db.t3.small 또는 db.t3.medium

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

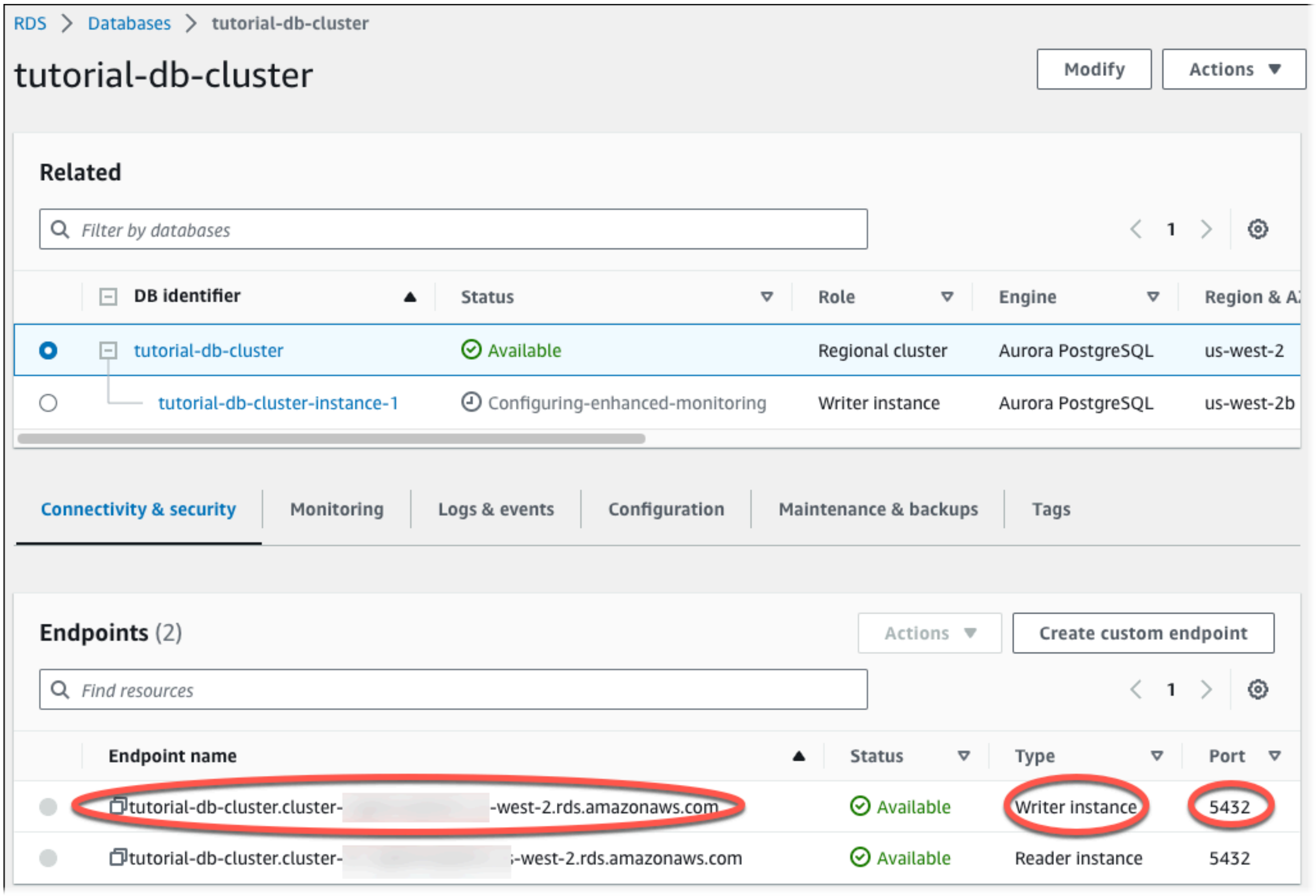


10. 가용성 및 내구성 섹션에서 기본값을 사용합니다.
11. 연결 섹션에서 다음 값을 설정하고 다른 값을 기본값으로 유지합니다.
    - 컴퓨팅 리소스에서 EC2 컴퓨팅 리소스에 연결을 선택합니다.
    - EC2 인스턴스의 경우 tutorial-ec2-instance-web-server와 같이 이전에 생성한 EC2 인스턴스를 선택합니다.



12. Additional configuration(추가 구성) 섹션을 열고 Initial database name(초기 데이터베이스 이름)에 **`sample`**를 입력합니다. 다른 옵션은 기본 설정을 유지합니다.
13. Aurora PostgreSQL DB 클러스터를 생성하려면 데이터베이스 생성을 선택합니다.

    새 DB 클러스터가 데이터베이스목록에 생성 중의 상태로 나타납니다.
14. 새 DB 클러스터의 상태가 사용 가능으로 나타날 때까지 기다립니다. 그런 다음, 세부 정보를 표시할 DB 클러스터 이름을 선택합니다.
15. Connectivity & security(연결 및 보안) 섹션에서 라이터 DB 인스턴스의 엔드포인트 및 포트를 확인합니다.



라이터 DB 인스턴스의 엔드포인트와 포트를 적어 둡니다. 이 정보를 사용하여 웹 서버를 DB 클러스터에 연결하게 됩니다.

16. EC2 인스턴스에 웹 서버 설치를 완료합니다.

EC2 인스턴스에 웹 서버 설치

EC2 인스턴스를 시작하여 DB 클러스터에 연결에서 생성한 EC2 인스턴스에 웹 서버를 설치합니다. 웹 서버는 Amazon Aurora DB 클러스터 생성에서 생성한 Amazon Aurora DB 클러스터에 연결됩니다.

PHP 및 MariaDB와 함께 Apache 웹 서버 설치

EC2 인스턴스에 연결하고 서버를 설치합니다.

EC2 인스턴스에 연결하고 PHP가 포함된 Apache 웹 서버를 설치하는 방법

1. Amazon EC2 사용 설명서에 있는 Linux 인스턴스에 연결의 단계를 따라 앞에서 만든 EC2 인스턴스에 연결합니다.

SSH를 사용하여 EC2 인스턴스에 연결하는 것이 좋습니다. Windows, Linux 또는 Mac에 SSH 클라이언트 유틸리티가 설치된 경우 다음 명령 형식을 사용하여 인스턴스에 연결할 수 있습니다.

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

예를 들어 `ec2-database-connect-key-pair.pem`이 Linux의 `/dir1`에 저장되어 있고, EC2 인스턴스의 퍼블릭 IPv4 DNS가 `ec2-12-345-678-90.compute-1.amazonaws.com`이라고 가정해 보겠습니다. SSH 명령은 다음과 같이 표시됩니다.

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. EC2 인스턴스에서 소프트웨어를 업데이트하여 최신 버그 수정 및 보안 업데이트를 받습니다. 이렇게 하려면 다음 명령을 사용하십시오.

> **Note**
>
> -y 옵션을 사용하면 확인 여부를 묻지 않고 업데이트를 설치합니다. 설치 전에 업데이트 정보를 확인하려면 이 옵션을 생략합니다.

```
sudo dnf update -y
```

3. 업데이트가 완료되면 다음 명령을 사용하여 Apache 웹 서버, PHP, MariaDB 또는 PostgreSQL 소프트웨어를 설치합니다. 이 명령은 여러 소프트웨어 패키지와 관련 종속 프로그램을 동시에 설치합니다.

MariaDB & MySQL

```
sudo dnf install -y httpd php php-mysqli mariadb105
```

PostgreSQL

```
sudo dnf install -y httpd php php-pgsql postgresql15
```

오류가 발생하면 인스턴스가 Amazon Linux 2023 AMI로 실행되지 않은 것입니다. Amazon Linux 2 AMI를 사용하고 있는 것일 수 있습니다. 다음 명령을 사용하여 Amazon Linux 버전을 볼 수 있습니다.

```
cat /etc/system-release
```

자세한 내용은 인스턴스 소프트웨어 업데이트 단원을 참조하십시오.

4. 다음 명령을 사용하여 웹 서버를 시작합니다.

```
sudo systemctl start httpd
```

웹 서버가 제대로 설치되고 시작되었는지 테스트할 수 있습니다. 이렇게 하려면 웹 브라우저의 주소 표시줄에 EC2 인스턴스의 퍼블릭 Domain Name System(DNS) 이름을 입력합니다(예: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`). 웹 서버가 실행되고 있으면 Apache 테스트 페이지가 표시됩니다.

Apache 테스트 페이지가 표시되지 않으면 자습서: DB 클러스터에 사용할 Amazon VPC 생성 (IPv4 전용)에서 생성한 VPC 보안 그룹에 대한 인바운드 규칙을 확인합니다. 인바운드 규칙에 웹 서버에 연결할 IP 주소에 대한 HTTP(포트 80) 액세스를 허용하는 규칙이 포함되어 있는지 확인합니다.

> **Note**
>
> Apache 테스트 페이지는 문서의 루트 디렉터리 `/var/www/html`에 콘텐츠가 없는 경우에만 표시됩니다. 문서의 루트 디렉터리에 콘텐츠를 추가한 후에는 콘텐츠가 EC2 인스턴스의 퍼블릭 DNS 주소에 나타납니다. 이 시점 이전에는 Apache 테스트 페이지에 나타납니다.

5. `systemctl` 명령을 사용하여 웹 서버가 시스템 부팅 때마다 시작되도록 구성합니다.

```
sudo systemctl enable httpd
```

`ec2-user`가 Apache 웹 서버의 기본 루트 디렉터리에 있는 파일을 관리할 수 있도록 하려면 `/var/www` 디렉터리의 소유권 및 권한을 변경합니다. 이 작업을 수행하는 방법에는 여러 가지가 있습니다. 본

자습서에서는 ec2-user를 apache 그룹에 추가하여 apache 그룹에 /var/www 디렉터리의 소유권을 부여하고 쓰기 권한을 할당합니다.

Apache 웹 서버에 대한 파일 권한 설정 방법

1. ec2-user 사용자를 apache 그룹에 추가합니다.

```
sudo usermod -a -G apache ec2-user
```

2. 권한을 새로 고치고 새 apache 그룹을 포함하려면 로그아웃합니다.

```
exit
```

3. 다시 로그인한 다음, apache 명령을 사용하여 groups 그룹이 있는지 확인합니다.

```
groups
```

출력 결과는 다음과 비슷합니다.

```
ec2-user adm wheel apache systemd-journal
```

4. /var/www 디렉터리 및 해당 콘텐츠의 그룹 소유권을 apache 그룹으로 변경합니다.

```
sudo chown -R ec2-user:apache /var/www
```

5. /var/www 및 그 하위 디렉터리의 디렉터리 권한을 변경해서 그룹 쓰기 권한을 추가하고 나중에 생성될 하위 디렉터리에서 그룹 ID를 설정합니다.

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. /var/www 디렉터리 및 하위 디렉터리의 파일 권한을 계속 변경해서 그룹 쓰기 권한을 추가합니다.

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

이제 ec2-user(및 apache 그룹의 향후 멤버)는 Apache 문서 루트에서 파일을 추가, 삭제, 편집할 수 있습니다. 따라서 정적 웹 사이트 또는 PHP 애플리케이션과 같은 콘텐츠를 추가할 수 있습니다.

> **Note**
>
> HTTP 프로토콜을 실행하는 웹 서버는 송신하거나 수신하는 데이터에 대해 아무런 전송 보안 기능도 제공하지 않습니다. 웹 브라우저를 사용하여 HTTP 서버에 연결하면 네트워크 경로를 따라 어디서든 엿보려는 사람들이 많은 정보를 볼 수 있습니다. 이 정보에는 방문하는 URL, 수신하는 웹 페이지의 내용, HTML 양식의 내용(암호 포함)이 포함됩니다.
> 웹 서버를 안전하게 보호하기 위한 최선의 방법은 HTTPS(HTTP Secure) 지원 기능을 설치하는 것입니다. 이 프로토콜은 SSL/TLS 암호화로 데이터를 보호합니다. 자세한 내용은 Amazon EC2 사용 설명서에서 [자습서: Amazon Linux AMI를 사용하여 SSL/TLS 구성]을 참조하세요.

DB 클러스터에 Apache 웹 서버 연결

이제 Amazon Aurora DB 클러스터에 연결되는 Apache 웹 서버에 콘텐츠를 추가합니다.

DB 클러스터에 연결되는 Apache 웹 서버에 콘텐츠를 추가하는 방법

1. EC2 인스턴스에 계속 연결되어 있을 때 디렉터리를 `/var/www`로 변경하고 `inc`라는 새로운 하위 디렉터리를 생성합니다.

```
cd /var/www
mkdir inc
cd inc
```

2. `inc`라는 `dbinfo.inc` 디렉터리에서 새 파일을 생성한 다음 nano 또는 선택한 편집기를 호출하여 파일을 편집합니다.

```
>dbinfo.inc
nano dbinfo.inc
```

3. 다음 내용을 `dbinfo.inc` 파일에 추가합니다. 여기서 *`db_instance_endpoint`*는 DB 클러스터에 대해 포트가 없는 DB 클러스터 라이터 엔드포인트입니다.

> **Note**
>
> 웹 서버의 문서 루트에 속하지 않은 폴더에 사용자 이름과 암호 정보를 두는 것이 좋습니다. 이렇게 하면 보안 정보가 노출될 가능성이 줄어듭니다.
> 애플리케이션에서 적절한 암호로 `master password`를 변경해야 합니다.

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. `dbinfo.inc` 파일을 저장하고 닫습니다. nano를 사용하는 경우 Ctrl+S 및 Ctrl+X를 사용하여 파일을 저장하고 닫습니다.

5. 디렉터리를 `/var/www/html`로 변경합니다.

```
cd /var/www/html
```

6. `html`라는 `SamplePage.php` 디렉터리에서 새 파일을 생성한 다음 nano 또는 선택한 편집기를 호출하여 파일을 편집합니다.

```
>SamplePage.php
nano SamplePage.php
```

7. 다음 콘텐츠를 `SamplePage.php` 파일에 추가합니다.

MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);
```

```
  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
```

```
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";
```

```
    if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);
```

```
/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
```

```
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
```

```
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. `SamplePage.php` 파일을 저장하고 닫습니다.
9. 웹 브라우저를 열고 `http://EC2 instance endpoint/SamplePage.php`(예: `http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php`)를 검색하여 웹 서버에서 DB 클러스터에 제대로 연결되는지 확인합니다.

`SamplePage.php`를 사용하여 DB 클러스터에 데이터를 추가할 수 있습니다. 그러면 추가한 데이터가 페이지에 표시됩니다. 데이터가 테이블에 삽입되었는지 확인하려면 Amazon EC2 인스턴스에 MySQL을 설치합니다. 그런 다음 DB 클러스터에 연결하여 테이블을 쿼리합니다.

DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 섹션을 참조하세요.

DB 클러스터를 최대한 보호하려면 VPC 외부의 소스가 DB 클러스터에 연결할 수 없는지 확인합니다.

웹 서버 및 데이터베이스 테스트를 완료한 후에는 DB 클러스터와 Amazon EC2 인스턴스를 삭제해야 합니다.

- DB 클러스터를 생성하려면 Aurora DB 클러스터 및 DB 인스턴스 삭제의 지침을 따릅니다. 최종 스냅샷을 생성할 필요가 없습니다.
- Amazon EC2 인스턴스를 종료하려면 Amazon EC2 사용 설명서에서 인스턴스 종료 섹션의 지침을 따르세요.

Amazon Aurora 자습서 및 샘플 코드

AWS 문서에는 일반적인 Amazon Aurora 사용 사례를 안내하는 몇 가지 자습서가 포함되어 있습니다. 이들 자습서 중에는 Amazon Aurora를 다른 AWS 서비스와 함께 사용하는 방법을 설명하는 자습서가 많습니다. 또한 GitHub에서 샘플 코드에 액세스할 수 있습니다.

> **Note**
> AWS 데이터베이스 블로그에서 더 많은 자습서를 찾을 수 있습니다. 교육에 대한 자세한 내용은 AWS Training and Certification을 참조하세요.

주제

이 안내서의 자습서

이 안내서에 포함된 다음 자습서는 Amazon Aurora의 일반적인 작업을 수행하는 방법을 설명합니다.

- 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)

  Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)에 DB 클러스터를 포함하는 방법을 알아봅니다. 이 경우 VPC는 동일한 VPC의 Amazon EC2에서 실행 중인 웹 서버와 데이터를 공유합니다.

- 자습서: DB 클러스터(듀얼 스택 모드)에 사용할 VPC 생성

  Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)에 DB 클러스터를 포함하는 방법을 알아봅니다. 이 경우 VPC는 동일한 VPC의 Amazon EC2와 데이터를 공유합니다. 이 자습서에서는 이중 스택 모드에서 실행되는 데이터베이스와 함께 작동하는 이 시나리오의 VPC를 생성합니다.

- 자습서: 웹 서버 및 Amazon Aurora DB 클러스터 생성

PHP가 있는 Apache 웹 서버를 설치하고 MySQL 데이터베이스를 생성하는 방법을 알아봅니다. 이 웹 서버는 Amazon Linux를 사용하여 Amazon EC2 인스턴스에서 실행되며, MySQL 데이터베이스는 Aurora MySQL DB 클러스터입니다. Amazon EC2 인스턴스 및 DB 클러스터가 모두 Amazon VPC에서 실행됩니다.

- 자습서: DB 클러스터 스냅샷에서 Amazon Aurora DB 클러스터 복원

  DB 클러스터를 DB 클러스터 스냅샷에서 복원하는 방법을 알아봅니다.

- 자습서: 태그를 사용하여 중지할 Aurora DB 클러스터 지정

  태그를 사용하여 중지할 Aurora DB 클러스터를 지정하는 방법을 알아봅니다.

- 자습서: Amazon EventBridge를 사용하여 DB 인스턴스의 상태 변경 로깅

  Amazon EventBridge 및 AWS Lambda를 사용하여 DB 인스턴스 상태 변경을 로깅하는 방법을 알아봅니다.

다른 AWS 안내서의 자습서

다른 AWS 안내서에 포함된 다음 자습서는 Amazon Aurora의 태스크를 수행하는 방법을 설명합니다.

> **Note**
>
> 일부 자습서에서는 Amazon RDS DB 인스턴스를 사용하지만 Aurora DB 클러스터를 사용하는 데에도 적용할 수 있습니다.

- AWS AppSync 개발자 안내서의 자습서: Aurora Serverless

  Data API가 활성화된 AWS AppSync DB 클러스터에 대해 기존 SQL 명령을 실행하기 위해 Aurora Serverless를 사용하여 데이터 원본을 제공하는 방법을 알아봅니다. AWS AppSync 해석기를 사용하면 GraphQL 쿼리, 변형 및 구독과 관련된 데이터 API에 대해 SQL 문을 실행할 수 있습니다.

- AWS Secrets Manager 사용 설명서의 자습서: AWS 데이터베이스에 대한 암호 교체

  AWS 데이터베이스에 대한 보안 암호를 생성하여 일정에 따라 교체하도록 구성하는 방법을 알아봅니다. 교체를 수동으로 한 번 트리거한 후 보안 암호의 새 버전으로 계속해서 액세스할 수 있는지 확인합니다.

- AWS Elastic Beanstalk 개발자 안내서의 자습서 및 샘플

AWS Elastic Beanstalk와 함께 Amazon RDS 데이터베이스를 사용하는 애플리케이션을 배포하는 방법을 알아봅니다.

- Amazon Machine Learning Developer Guide의 Amazon RDS 데이터베이스의 데이터를 사용하여 Amazon ML 데이터 원본 생성

  MySQL DB 인스턴스에 저장된 데이터로 Amazon Machine Learning(Amazon ML) 데이터 원본 객체를 생성하는 방법을 알아봅니다.

- Amazon QuickSight 사용 설명서의 수동으로 VPC의 Amazon RDS 인스턴스에 대한 액세스 허용

  VPC의 Amazon RDS DB 인스턴스에 대한 QuickSight 액세스를 활성화하는 방법을 알아봅니다.

Amazon Aurora PostgreSQL에 대한 AWS 워크숍 및 랩 콘텐츠 포털

다음의 워크숍 및 기타 실습 콘텐츠 모음은 Amazon Aurora PostgreSQL의 특성과 기능을 이해하는 데 도움이 됩니다.

- 수동으로 새 Aurora 클러스터 생성

  Amazon Aurora PostgreSQL 클러스터를 수동으로 생성하는 방법을 알아봅니다.

- Cloud9 구성 및 데이터베이스 초기화

  Cloud9을 구성하고 PostgreSQL 데이터베이스를 초기화하는 방법을 알아봅니다.

- 고속 클로닝

  Aurora 고속 클론을 생성하는 방법을 알아봅니다.

- 쿼리 계획 관리

  쿼리 계획 관리를 사용하여 문 집합에 대한 실행 계획을 제어하는 방법을 알아봅니다.

- 클러스터 캐시 관리

  Aurora PostgreSQL의 클러스터 캐시 관리 기능에 대해 알아봅니다.

- 데이터베이스 활동 스트리밍

  이 기능을 사용하여 데이터베이스 활동을 모니터링 및 감사하는 방법을 알아봅니다.

- 성능 개선 도우미

  성능 개선 도우미를 사용하여 DB 인스턴스를 모니터링하고 튜닝하는 방법을 알아봅니다.

- [RDS 도구를 사용한 성능 모니터링]

  AWS 및 Postgres 도구(Cloudwatch, 고급 모니터링, 느린 쿼리 로그, 성능 개선 도우미, PostgreSQL 카탈로그 보기)를 사용하여 성능 문제를 이해하고, 데이터베이스의 성능을 개선할 수 있는 방법을 알아봅니다.
- [Auto Scaling 읽기 전용 복제본]

  로드 생성기 스크립트를 사용하여 Aurora 읽기 전용 복제본 Auto Scaling이 실제로 어떻게 작동하는지 알아봅니다.
- [내결함성 테스트]

  DB 클러스터가 결함을 어떤 방식으로 견딜 수 있는지 알아봅니다.
- [Aurora Global Database]

  Aurora Global Database에 대해 알아봅니다.
- [기계 학습 사용]

  Aurora 기계 학습에 대해 알아봅니다.
- [Aurora Serverless v2]

  Aurora Serverless v2에 대해 알아봅니다.
- [Aurora PostgreSQL용 신뢰할 수 있는 언어 확장]

  Aurora PostgreSQL에서 안전하게 실행되는 고성능 확장 프로그램을 구축하는 방법을 알아봅니다.

Amazon Aurora MySQL에 대한 AWS 워크숍 및 랩 콘텐츠 포털

다음의 워크숍 및 기타 실습 콘텐츠 모음은 Amazon Aurora MySQL의 특성과 기능을 이해하는 데 도움이 됩니다.

- [Aurora 클러스터 생성]

  Amazon Aurora MySQL 클러스터를 수동으로 생성하는 방법을 알아봅니다.
- [데이터베이스에 연결하기 위한 Cloud9 클라우드 기반 IDE 환경 만들기]

  Cloud9을 구성하고 MySQL 데이터베이스를 초기화하는 방법을 알아봅니다.
- [고속 클로닝]

Aurora 고속 클론을 생성하는 방법을 알아봅니다.

- DB 클러스터 역추적

  DB 클러스터를 역추적하는 방법을 알아봅니다.

- 성능 개선 도우미

  성능 개선 도우미를 사용하여 DB 인스턴스를 모니터링하고 튜닝하는 방법을 알아봅니다.

- RDS 도구를 사용한 성능 모니터링

  AWS 및 SQL 도구를 사용하여 성능 문제를 이해하고 데이터베이스의 성능을 개선할 수 있는 방법을 알아봅니다.

- 쿼리 성능 분석

  다양한 도구를 사용하여 SQL 성능 관련 문제를 해결하는 방법을 알아봅니다.

- Auto Scaling 읽기 전용 복제본

  Auto Scaling 읽기 전용 복제본의 작동 방식을 알아봅니다.

- 내결함성 테스트

  Aurora MySQL의 고가용성 및 내결함성 기능에 대해 알아봅니다.

- Aurora Global Database

  Aurora Global Database에 대해 알아봅니다.

- Aurora Serverless v2

  Aurora Serverless v2에 대해 알아봅니다.

- 기계 학습 사용

  Aurora 기계 학습에 대해 알아봅니다.

GitHub의 자습서 및 샘플 코드

GitHub의 다음 자습서와 샘플 코드는 Amazon Aurora에서 일반적인 태스크를 수행하는 방법을 설명합니다.

- Aurora Serverless v2 대여 도서관 생성

고객이 책을 빌리고 반납할 수 있는 대여 도서관 애플리케이션을 생성하는 방법을 알아봅니다. 이 예에서는 Aurora Serverless v2 및 AWS SDK for Python (Boto3)을 사용합니다.

- SDK for Java 2.x를 사용하여 Aurora Serverless v2 데이터를 쿼리하는 Spring REST API를 사용하여 Amazon Aurora 항목 추적기 애플리케이션 생성

  Aurora Serverless v2 데이터를 쿼리하는 Spring REST API를 만드는 방법을 알아봅니다. SDK for Java 2.x를 사용하는 React 애플리케이션에서 사용하기 위한 것입니다.

- AWS SDK for PHP를 사용하여 Aurora Serverless v2 데이터를 쿼리하는 Amazon Aurora 항목 추적기 애플리케이션 생성

  데이터 API 및 `RdsDataClient`의 Aurora Serverless v2를 사용하는 애플리케이션을 만들어 작업 항목을 추적하고 보고하는 방법을 알아봅니다. 이 예제에서는 AWS SDK for PHP를 사용합니다.

- AWS SDK for Python (Boto3)를 사용하여 Aurora Serverless v2 데이터를 쿼리하는 Amazon Aurora 항목 추적기 애플리케이션 생성

  데이터 API 및 `RdsDataClient`의 Aurora Serverless v2를 사용하는 애플리케이션을 만들어 작업 항목을 추적하고 보고하는 방법을 알아봅니다. 이 예제에서는 AWS SDK for Python (Boto3)를 사용합니다.

AWS SDK와 함께 이 서비스 사용

다양한 프로그래밍 언어에 대해 AWS 소프트웨어 개발 키트(SDK)을 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예시 및 설명서를 제공합니다.

SDK 설명서	코드 예시
AWS SDK for C++	AWS SDK for C++ 코드 예시
AWS CLI	AWS CLI 코드 예시
AWS SDK for Go	AWS SDK for Go 코드 예시
AWS SDK for Java	AWS SDK for Java 코드 예시
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예시

SDK 설명서	코드 예시
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예시
AWS SDK for .NET	AWS SDK for .NET 코드 예시
AWS SDK for PHP	AWS SDK for PHP 코드 예시
AWS Tools for PowerShell	AWS Tools for PowerShell 코드 예시
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예시
AWS SDK for Ruby	AWS SDK for Ruby 코드 예시
AWS SDK for Rust	AWS SDK for Rust 코드 예시
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예시
AWS SDK for Swift	AWS SDK for Swift 코드 예시

이 서비스 관련 예시는 AWS SDK를 사용한 Aurora용 코드 예제를 참조하세요.

> **예제 사용 가능 여부**
>
> 필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

Amazon Aurora에 대한 프로그래밍 방식 액세스

Amazon RDS는 프로그래밍 방식으로 Amazon Aurora 리소스를 관리할 수 있는 다음 도구를 제공합니다.

AWS Command Line Interface (AWS CLI)

명령줄 쉘에서 AWS CLI를 사용하여 Amazon Aurora 리소스를 생성하고 관리할 수 있습니다. AWS CLI는 Amazon RDS와 같은 AWS 서비스의 API에 대한 직접 액세스를 제공합니다. Amazon RDS에 대한 명령의 구문 및 예시는 AWS CLI 명령 참조에서 rds 섹션을 참조하세요.

AWS CloudFormation

이 AWS 코드형 인프라(IaC) 도구를 사용하면 원하는 모든 Amazon Aurora 리소스를 설명하는 템플릿을 생성하고 AWS CloudFormation이 해당 리소스를 프로비저닝하고 구성해 줍니다. 자세한 내용은 the section called “을 사용하여 리소스 생성AWS CloudFormation” 단원을 참조하십시오.

AWS 소프트웨어 개발 키트(SDK)

AWS는 널리 사용되는 많은 기술 및 프로그래밍 언어를 위한 SDK를 제공합니다. 이를 통해 애플리케이션 내에서 해당 언어 또는 기술로 AWS 서비스를 더 쉽게 직접적으로 호출할 수 있습니다. 이러한 SDK에 대한 자세한 내용은 AWS에서 애플리케이션 개발 및 관리를 위한 도구를 참조하세요.

Amazon RDS API

이 API는 Amazon RDS의 프로토콜 수준 인터페이스입니다. 이 API를 사용할 때는 모든 HTTPS 요청을 올바르게 포맷하고 모든 요청에 유효한 디지털 서명을 추가해야 합니다. 자세한 내용은 Amazon RDS API 참조 단원을 참조하십시오.

콘솔 투 코드

이 도구를 사용하면 Amazon RDS 콘솔에서 수행하는 작업에 대한 코드를 생성하고 AWS CloudFormation과 같은 다른 도구에서 해당 코드를 사용할 수 있습니다. 자세한 내용은 the section called “콘솔 투 코드” 단원을 참조하십시오.

RDS Data API

RDS 데이터 API(데이터 API)는 Amazon Aurora Serverless 데이터베이스의 데이터에 프로그래밍 방식으로 액세스할 수 있는 권한을 제공합니다. 데이터 API는 연결을 관리하지 않고 SQL 문을 실행할 수 있는 안전한 HTTP 엔드포인트를 제공합니다. 자세한 내용은 Amazon RDS Data API 사용 단원을 참조하십시오.

콘솔 투 코드를 사용하여 Amazon Aurora 콘솔 작업을 위한 코드를 생성합니다.

콘솔은 리소스를 생성하고 프로토타입을 테스트하기 위한 안내 경로를 제공합니다. 대규모로 동일한 리소스를 생성하려면 자동화 코드가 필요합니다. 콘솔 투 코드는 자동화 코드를 시작하는 데 도움이 되는 Amazon Q Developer의 기능입니다. 콘솔 투 코드는 기본값과 사용자가 제공한 파라미터 값을 포함하여 콘솔 작업을 기록합니다. 그런 다음 생성형 AI를 사용하여 선택한 작업에 대해 원하는 언어와 형식으로 코드를 제안합니다. 콘솔 워크플로는 사용자가 지정한 파라미터 값이 모두 유효한지 확인하기 때문에 콘솔 투 코드를 사용하여 생성하는 코드는 호환되는 파라미터 값을 가집니다. 코드를 시작점으로 사용하여 특정 사용 사례에서 프로덕션에 바로 사용할 수 있도록 사용자 지정할 수 있습니다.

예를 들어 콘솔 투 코드를 사용하면 Aurora DB 클러스터 생성을 기록하고 AWS CloudFormation JSON 형식으로 코드를 생성하도록 선택할 수 있습니다. 그런 다음 해당 코드를 복사하고 AWS CloudFormation 템플릿에서 사용할 수 있도록 사용자 지정할 수 있습니다.

콘솔 투 코드는 현재 다음과 같은 언어 및 형식으로 코드형 인프라(IAC)를 생성할 수 있습니다.

- CDK Java
- CDK Python
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

콘솔 투 코드를 사용하는 방법에 대한 자세한 내용 및 지침은 Amazon Q Developer 사용 설명서의 Amazon Q Developer 콘솔 투 코드를 사용하여 AWS 서비스 자동화를 참조하세요.

Amazon Aurora DB 클러스터 설정

이 섹션에서는 Aurora DB 클러스터를 설정하는 방법을 보여줍니다. Aurora DB 클러스터를 생성하기 전에 DB 클러스터를 실행하는 DB 인스턴스 클래스를 결정하십시오. 또한 AWS 리전을 선택하여 DB 클러스터 실행 위치를 결정하십시오. 그런 다음, DB 클러스터를 생성하십시오. Aurora 이외의 데이터가 있는 경우 이 데이터를 Aurora DB 클러스터로 마이그레이션할 수 있습니다.

주제

Amazon Aurora DB 클러스터 생성

Amazon Aurora DB 클러스터는 MySQL 또는 PostgreSQL과 호환되는 DB 인스턴스와 3개의 가용 영역에 걸쳐 복사되는 DB 클러스터의 데이터를 단일 가상 볼륨으로 보유하는 클러스터 볼륨으로 구성됩니다. 기본적으로 Aurora DB 클러스터에는 읽기와 쓰기를 수행하는 기본 DB 인스턴스와 옵션으로 최대 15개의 Aurora 복제본(리더 DB 인스턴스)이 포함됩니다. Aurora DB 클러스터에 대한 자세한 내용은 Amazon Aurora DB 클러스터 단원을 참조하세요.

Aurora에는 두 가지 주요 유형의 DB 클러스터가 있습니다.

- Aurora 프로비저닝 - 예상 워크로드에 따라 라이터 및 리더 인스턴스용 DB 인스턴스 클래스를 선택합니다. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요. Aurora 프로비저닝에는 Aurora 글로벌 데이터베이스를 비롯한 여러 옵션이 있습니다. 자세한 내용은 Amazon Aurora Global Database 사용 섹션을 참조하세요.
- Aurora Serverless - Aurora Serverless v2는 Aurora에 대한 온디맨드 방식의 자동 조정 구성입니다. 용량은 애플리케이션 수요에 따라 자동으로 조정됩니다. DB 클러스터가 사용하는 리소스에 대해서만 청구됩니다. 이 자동화는 매우 가변적이고 예측할 수 없는 워크로드가 있는 환경에 특히 유용합니다. 자세한 내용은 Aurora Serverless v2 사용하기 섹션을 참조하세요.

다음에서는 Aurora DB 클러스터를 생성하는 방법을 확인할 수 있습니다. 시작하려면 먼저 DB 클러스터 사전 조건 단원을 참조하세요.

Aurora DB 클러스터에 연결하는 지침은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

목차

DB 클러스터 사전 조건

> **Important**
>
> Aurora DB 클러스터를 생성할 수 있으려면 먼저 Amazon Aurora 환경 설정 단원의 작업들을 완료해야 합니다.

다음은 DB 클러스터를 생성하기 전에 완료해야 하는 전제 조건입니다.

주제

- DB 클러스터의 네트워크 구성
- 추가 사전 조건

DB 클러스터의 네트워크 구성

Amazon Aurora DB 클러스터는 가용 영역이 최소 2개 이상인 AWS 리전의 Amazon VPC 서비스 기반 Virtual Private Cloud(VPC)에서만 생성할 수 있습니다. DB 클러스터에 대해 선택한 DB 서브넷 그룹은 2개 이상의 가용 영역을 포함해야 합니다. 드물게 가용 영역에 장애가 발생할 경우 이 구성을 사용하면 장애 조치에 사용 가능한 DB 인스턴스가 DB 클러스터에 항상 하나 이상 있어야 합니다.

새 DB 클러스터와 동일한 VPC의 EC2 인스턴스 간에 연결을 설정하려는 경우 DB 클러스터 생성 중에 설정할 수 있습니다. 동일한 VPC의 EC2 인스턴스 이외의 리소스에서 DB 클러스터에 연결하려는 경우 네트워크 연결을 수동으로 구성할 수 있습니다.

주제

- EC2 인스턴스와의 자동 네트워크 연결 구성
- 네트워크 수동 구성

EC2 인스턴스와의 자동 네트워크 연결 구성

Aurora DB 클러스터를 생성할 때 AWS Management Console을 사용하여 Amazon EC2 인스턴스와 새 DB 클러스터 간의 연결을 설정할 수 있습니다. 이렇게 하면 RDS가 VPC 및 네트워크 설정을 자동으로 구성합니다. DB 클러스터는 EC2 인스턴스가 DB 클러스터에 액세스할 수 있도록 EC2 인스턴스와 동일한 VPC에 생성됩니다.

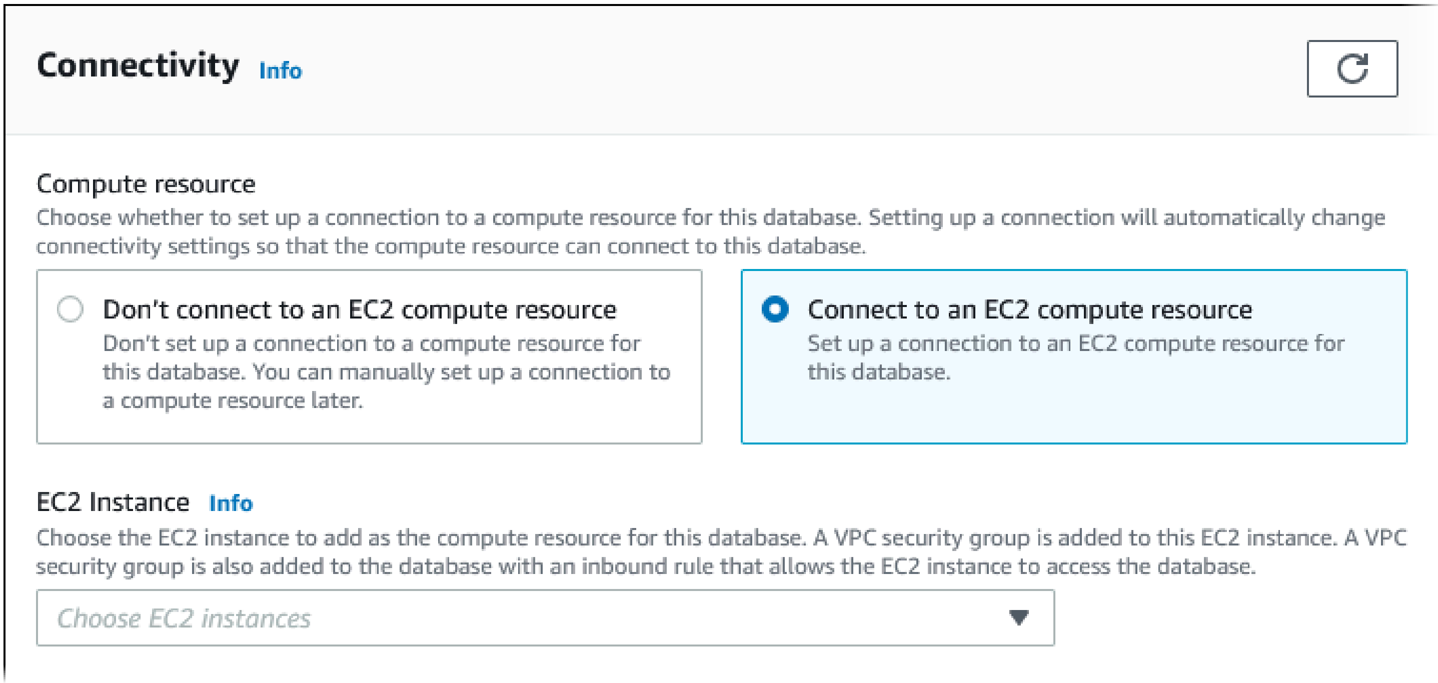
다음은 EC2 인스턴스를 DB 클러스터와 연결하기 위한 요구 사항입니다.

- DB 클러스터를 생성하려면 먼저 EC2 인스턴스가 AWS 리전에 있어야 합니다.

  AWS 리전에 EC2 인스턴스가 없는 경우 콘솔은 인스턴스를 생성할 수 있는 링크를 제공합니다.
- 현재 DB 클러스터는 Aurora Serverless DB 클러스터일 수 없으며 Aurora Global Database의 일부일 수도 없습니다.
- DB 인스턴스를 만드는 사용자는 다음 작업을 수행할 수 있는 권한이 있어야 합니다.
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

이 옵션을 사용하면 프라이빗 DB 클러스터가 생성됩니다. DB 클러스터는 프라이빗 서브넷만 있는 DB 서브넷 그룹을 사용하여 VPC 내 리소스에 대한 액세스를 제한합니다.

EC2 인스턴스를 DB 클러스터에 연결하려면 Create database(데이터베이스 생성) 페이지의 Connectivity(연결) 섹션에서 Connect to an EC2 compute resource(EC2 컴퓨팅 리소스에 연결)을 선택합니다.



Connect to an EC2 compute resource(EC2 컴퓨팅 리소스에 연결)를 선택하면 RDS가 다음 옵션을 자동으로 설정합니다. Don't connect to an EC2 compute resource(EC2 컴퓨팅 리소스에 연결하지 않음)를 선택하여 EC2 인스턴스와의 연결을 설정하지 않도록 선택하지 않는 한 이러한 설정을 변경할 수 없습니다.

콘솔 옵션	자동 설정
네트워크 유형	RDS는 네트워크 유형을 IPv4로 설정합니다. 현재 듀얼 스택 모드는 EC2 인스턴스와 DB 클러스터 간의 연결을 설정할 때 지원되지 않습니다.
Virtual Private Cloud(VPC)	RDS는 VPC를 EC2 인스턴스와 연결된 VPC로 설정합니다.
DB 서브넷 그룹	RDS에는 EC2 인스턴스와 동일한 가용 영역에 프라이빗 서브넷이 있는 DB 서브넷 그룹이 필요합니다. 이 요구 사항을 충족하는 DB 서브넷 그룹이 있다면 RDS는 기존 DB 서브넷 그룹을 사용합니다. 기본적으로 이 옵션은 Automatic setup(자동 설정)으로 설정되어 있습니다. Automatic setup(자동 설정)을 선택하고 이 요구 사항을 충족하는 DB 서브넷 그룹이 없으면 다음 작업이 수행됩니다. RDS는 가용 영역 중 하나가 EC2 인스턴스와 동일한 3개의 가용 영역에서 3개

콘솔 옵션	자동 설정
	의 사용 가능한 프라이빗 서브넷을 사용합니다. 가용 영역에서 프라이빗 서브넷을 사용할 수 없는 경우 RDS는 가용 영역에 프라이빗 서브넷을 생성합니다. 그런 다음 RDS는 DB 서브넷 그룹을 생성합니다. 프라이빗 서브넷을 사용할 수 있는 경우 RDS는 서브넷과 연결된 라우팅 테이블을 사용하고 생성된 모든 서브넷을 이 라우팅 테이블에 추가합니다. 프라이빗 서브넷을 사용할 수 없는 경우 RDS는 인터넷 게이트웨이 액세스가 없는 라우팅 테이블을 생성하고 생성한 서브넷을 라우팅 테이블에 추가합니다. RDS를 사용하면 기존 DB 서브넷 그룹도 사용할 수 있습니다. 선택한 기존 DB 서브넷 그룹을 사용하려면 Choose existing(기존 항목 선택)을 선택합니다.
공개 액세스(Public access)	RDS는 DB 클러스터에 공개적으로 액세스할 수 없도록 아니요를 선택합니다. 보안을 위해 데이터베이스를 비공개로 유지하고 인터넷에서 액세스할 수 없도록 하는 것이 가장 좋습니다.

콘솔 옵션	자동 설정
VPC 보안 그룹(방화벽)	RDS는 DB 클러스터와 연결된 새 보안 그룹을 생성합니다. 보안 그룹의 이름은 `rds-ec2-n`이며, 여기서 *n*은 숫자입니다. 이 보안 그룹에는 EC2 VPC 보안 그룹(방화벽)이 소스인 인바운드 규칙이 포함됩니다. DB 클러스터와 연결된 이 보안 그룹을 통해 EC2 인스턴스가 DB 클러스터에 액세스할 수 있습니다. RDS는 DB 인스턴스와 연결된 새 보안 그룹도 생성합니다. 보안 그룹의 이름은 `ec2-rds-n`이며, 여기서 *n*은 숫자입니다. 이 보안 그룹에는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙이 포함됩니다. 이 보안 그룹을 사용하면 EC2 인스턴스가 DB 클러스터에 트래픽을 보내도록 허용할 수 있습니다. Create new(새로 생성)를 선택하고 새 보안 그룹의 이름을 입력하여 다른 새 보안 그룹을 추가할 수 있습니다. Choose existing(기존 항목 선택)을 선택하고 추가할 보안 그룹을 선택하여 기존 보안 그룹을 추가할 수 있습니다.
가용 영역	DB 클러스터 생성(단일 AZ 배포) 중에 Availability & durability(가용성 및 내구성) 측면에서 Aurora 복제본을 생성하지 않으면 RDS가 EC2 인스턴스의 가용 영역을 선택합니다. DB 클러스터 생성(다중 AZ 배포) 중에 Aurora 복제본을 생성하면 RDS는 DB 클러스터에 있는 하나의 DB 인스턴스에 대해 EC2 인스턴스의 가용 영역을 선택합니다. RDS는 DB 클러스터의 다른 DB 인스턴스에 대해 다른 가용 영역을 임의로 선택합니다. 기본 DB 인스턴스 또는 Aurora 복제본은 EC2 인스턴스와 동일한 가용 영역에서 생성됩니다. 기본 DB 인스턴스와 EC2 인스턴스가 서로 다른 가용 영역에 있는 경우 교차 가용 영역 비용이 발생할 가능성이 있습니다.

이러한 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

DB 클러스터를 생성한 후에 이러한 설정을 변경할 경우 변경 사항이 EC2 인스턴스와 DB 클러스터 간의 연결에 영향을 미칠 수 있습니다.

네트워크 수동 구성

동일한 VPC의 EC2 인스턴스 이외의 리소스에서 DB 클러스터에 연결하려는 경우 네트워크 연결을 수동으로 구성할 수 있습니다. AWS Management Console을 사용하여 DB 클러스터를 생성하는 경우에는 Amazon RDS에서 VPC를 자동으로 생성할 수 있습니다. 또는 Aurora DB 클러스터에서 기존 VPC를 사용하거나 새 VPC를 생성할 수 있습니다. 어떤 접근 방식이든 Amazon Aurora DB 클러스터에서 VPC를 사용하려면 2개 이상의 가용 영역마다 VPC에 서브넷이 1개 이상 있어야 합니다.

기본적으로 Amazon RDS는 기본 DB 인스턴스와 Aurora 복제본을 자동으로 가용 영역에 생성합니다. 특정 가용 영역을 선택하려면 Availability & durability(가용성 및 내구성) 다중 AZ 배포 설정을 Don't create an Aurora Replica(Aurora 복제본 생성 안 함)로 변경합니다. 이렇게 하면 VPC의 가용 영역 중에서 선택할 수 있는 Availability Zone(가용 영역) 설정이 노출됩니다. 그러나 기본 설정을 유지하고 Amazon RDS가 다중 AZ 배포를 생성하고 가용 영역을 선택하도록 하는 것을 권장합니다. 이렇게 하면 Aurora DB 클러스터는 Aurora의 두 가지 주요 이점 중 하나인 빠른 장애 조치 및 고가용성 기능을 사용하여 생성됩니다.

기본 VPC가 없거나 VPC를 생성하지 않았다면 콘솔을 사용해 DB 클러스터를 생성할 때 Amazon RDS에서 자동으로 VPC를 생성할 수 있습니다. 그 밖에는 다음과 같은 방법이 있습니다.

- DB 클러스터를 배포하려는 AWS 리전에서 두 개 이상의 가용 영역에 각각 한 개 이상의 서브넷을 갖는 VPC를 생성합니다. 자세한 내용은 VPC에서 DB 클러스터를 사용한 작업 및 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 단원을 참조하세요.
- DB 클러스터에 대한 연결 권한을 부여할 수 있도록 VPC 보안 그룹을 지정합니다. 자세한 내용은 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다. 및 보안 그룹을 통한 액세스 제어 단원을 참조하세요.
- DB 클러스터에서 VPC의 서브넷 2개 이상을 사용할 수 있도록 RDS DB 서브넷 그룹을 지정합니다. 자세한 내용은 DB 서브넷 그룹을 사용한 작업 섹션을 참조하세요.

VPC에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 단원을 참조하세요. 프라이빗 DB 클러스터에 대한 네트워크를 구성하는 자습서는 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 단원에서 확인하세요.

Aurora DB 클러스터와 동일한 VPC 있지 않은 리소스에 연결하려는 경우 VPC에서 DB 클러스터에 액세스하는 시나리오에서 적절한 시나리오를 참조하십시오.

추가 사전 조건

DB 클러스터를 만들려면 먼저 다음과 같은 추가 사전 조건을 고려하세요.

- AWS Identity and Access Management(IAM) 자격 증명을 사용하여 AWS에 연결할 경우 Amazon RDS 작업을 수행하는 데 필요한 사용 권한을 부여하는 IAM 정책이 AWS 계정에 필요합니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.

  IAM을 사용하여 Amazon RDS 콘솔에 액세스하려면, 먼저 사용자 자격 증명으로 AWS Management Console에 로그인해야 합니다. 그런 다음 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔로 이동합니다.
- DB 클러스터의 구성 파라미터를 사용자 지정하려면 DB 클러스터 파라미터 그룹과 DB 파라미터 그룹을 필요한 파라미터 설정으로 지정해야 합니다. DB 클러스터 파라미터 그룹 또는 DB 파라미터 그룹의 생성 또는 변경에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.
- DB 클러스터에 지정할 TCP/IP 포트 번호를 정합니다. 일부 기업에서는 방화벽이 Aurora 기본값 포트(MySQL일 때 3306, PostgreSQL일 때 5432)에 대한 연결을 차단합니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 DB 클러스터에 다른 포트를 선택해야 합니다. DB 클러스터의 인스턴스는 모두 동일한 포트를 사용합니다.
- 데이터베이스의 메이저 엔진 버전이 RDS 표준 지원 종료일에 다다른 경우 추가 지원 CLI 옵션 또는 RDS API 파라미터를 사용해야 합니다. 자세한 내용은 Aurora DB 클러스터 설정의 RDS 추가 지원을 참조하세요.

DB 클러스터 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora DB 클러스터를 생성할 수 있습니다.

콘솔

손쉬운 생성의 활성화 여부와 관계없이 AWS Management Console을 사용하여 DB 클러스터를 생성할 수 있습니다. Easy create(간편 생성)를 활성화한 경우에는 DB 엔진 유형, DB 인스턴스 크기 및 DB 인스턴스 식별자만 지정합니다. Easy create(간편 생성)는 다른 구성 옵션에서도 기본 설정을 사용합니다. Easy create(간편 생성)가 활성화되지 않은 경우에는 데이터베이스를 생성할 때 가용성, 보안, 백업 및 유지 관리에 대한 옵션을 포함하여 더 많은 구성 옵션을 지정합니다.

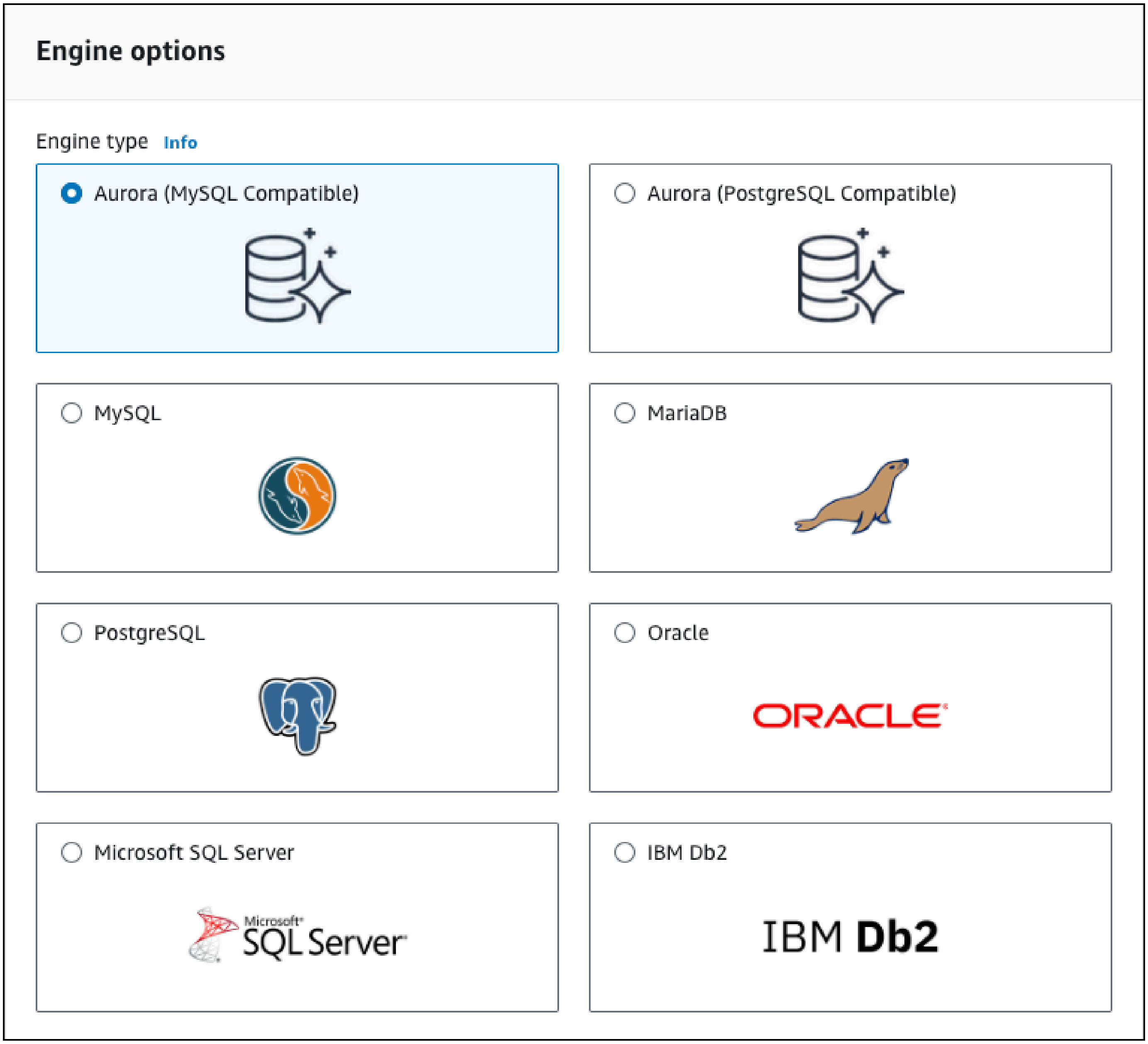
> **Note**
>
> 이 예에서는 [표준 생성(Standard Create)]이 활성화되고 [간편 생성(Easy Create)]은 활성화되지 않습니다. 손쉬운 생성을 활성화한 상태에서 DB 클러스터를 생성하는 방법은 Amazon Aurora 시작하기 섹션을 참조하세요.

콘솔을 사용하여 Aurora DB 클러스터를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단 모서리에서 DB 인스턴스를 생성하려는 AWS 리전을 선택합니다.

   일부 AWS 리전에서는 Aurora를 사용할 수 없습니다. Aurora를 사용할 수 있는 AWS 리전 목록은 리전 가용성 섹션을 참조하세요.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택합니다.
5. 데이터베이스 생성 방법 선택에서 표준 생성을 선택합니다.
6. 캐시 유형에서 다음 중 하나를 선택합니다.

   - Aurora(MySQL 호환)
   - Aurora(PostgreSQL 호환)



7. 엔진 버전을 선택합니다.

   자세한 내용은 Amazon Aurora 섹션을 참조하세요. 필터를 사용하여 원하는 기능(예: Aurora Serverless v2)과 호환되는 버전을 선택할 수 있습니다. 자세한 내용은 Aurora Serverless v2 사용하기 섹션을 참조하세요.

8. 템플릿에서 사용 사례에 맞는 템플릿을 선택합니다.
9. 마스터 암호를 입력하려면 다음과 같이 하세요.

   a. 설정 섹션에서 보안 인증 정보 설정을 엽니다.

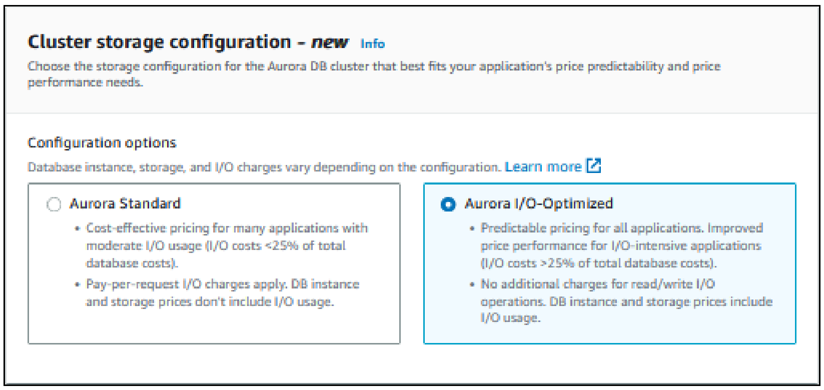
b. Auto generate a password(암호 자동 생성) 확인란의 선택을 취소합니다.

c. (선택 사항) 마스터 사용자 이름 값을 변경하고 마스터 암호 및 암호 확인에 동일한 암호를 입력합니다.

기본적으로 새 DB 인스턴스는 마스터 사용자를 위해 자동 생성된 암호를 사용합니다.

10. VPC 보안 그룹(방화벽) 아래의 연결 섹션에서 새로 만들기를 선택하면 로컬 컴퓨터의 IP 주소가 데이터베이스에 액세스할 수 있도록 허용하는 인바운드 규칙과 함께 VPC 보안 그룹이 생성됩니다.

11. 클러스터 스토리지 구성의 경우 Aurora I/O-Optimized 또는 Aurora Standard 중 하나를 선택합니다. 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 섹션을 참조하세요.



12. (선택 사항) 이 DB 클러스터의 컴퓨팅 리소스에 대한 연결을 설정합니다.

    DB 클러스터를 생성하는 동안 Amazon EC2 인스턴스와 새 DB 클러스터 간의 연결을 구성할 수 있습니다. 자세한 내용은 EC2 인스턴스와의 자동 네트워크 연결 구성 섹션을 참조하세요.

13. 나머지 섹션에서 DB 클러스터 설정을 지정합니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하십시오.

14. 데이터베이스 생성을 선택합니다.

    자동 생성된 암호를 사용하기로 한 경우에는 Databases(데이터베이스) 페이지에 View credential details(자격 증명 세부 정보 보기) 버튼이 나타납니다.

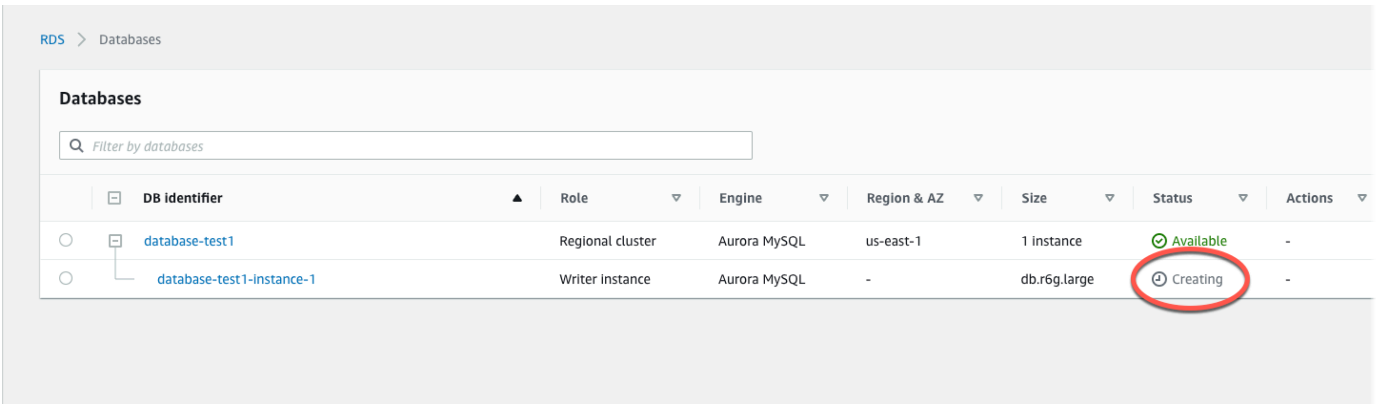
    DB 클러스터의 마스터 사용자 이름 및 암호를 보려면 자격 증명 세부 정보 보기를 선택합니다.

    DB 인스턴스를 마스터 사용자로 연결하려면 화면에 나타난 사용자 이름과 암호를 사용합니다.

> **Important**
>
> 마스터 사용자 암호를 다시 볼 수는 없습니다. 따라서 기록을 해두지 않으면 이를 변경해야 합니다. DB 인스턴스가 사용 가능한 상태가 되고 난 후에 마스터 사용자 암호를 변경해야 하는 경우에는 다음과 같은 방법으로 DB 인스턴스를 수정할 수 있습니다. DB 인스턴스 변경에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 단원을 참조하십시오.

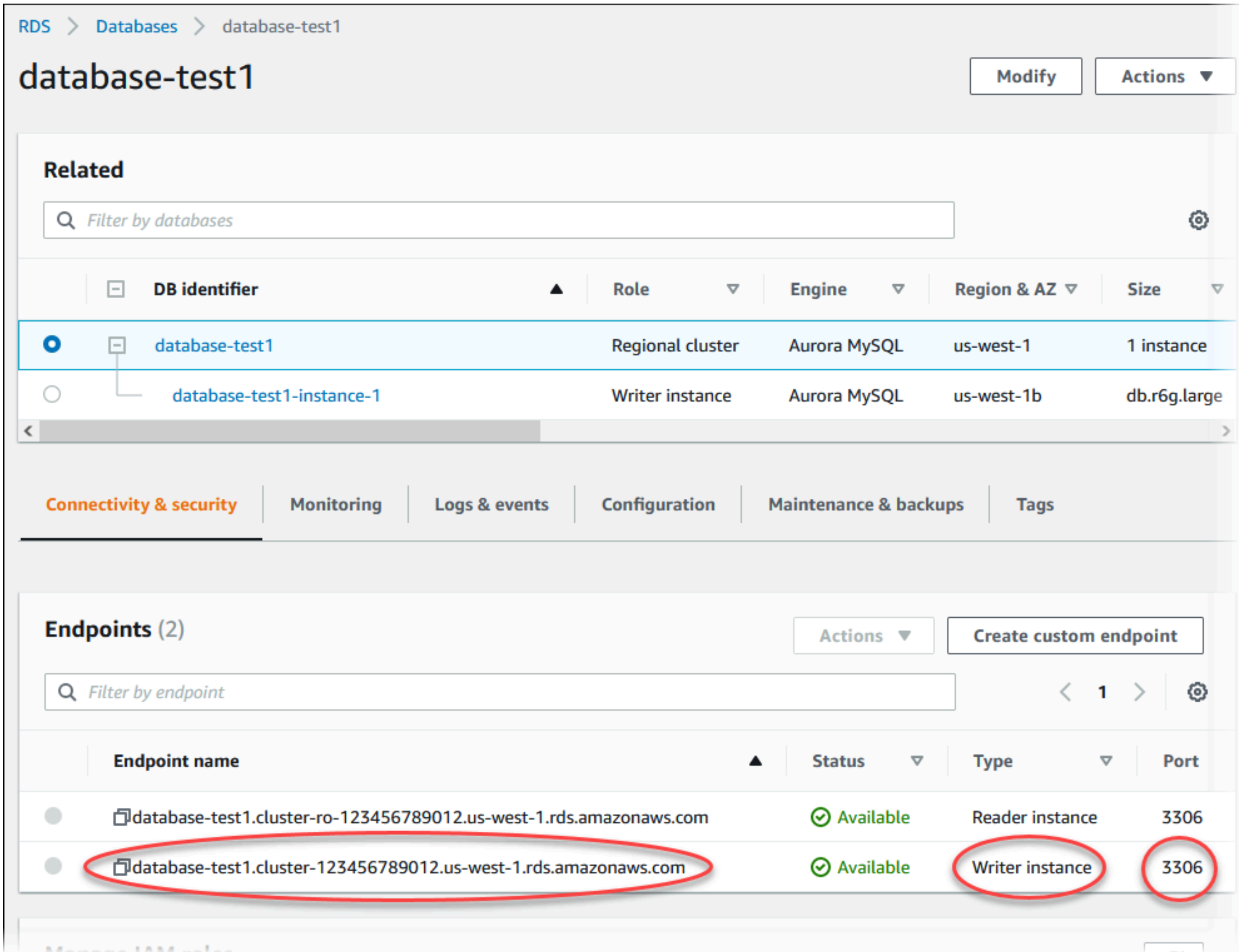
15. 데이터베이스에서 새 Aurora DB 클러스터의 이름을 선택합니다.

    RDS 콘솔에 새 DB 인스턴스의 세부 정보가 표시됩니다. DB 클러스터를 사용할 준비가 될 때까지 DB 클러스터와 그 DB 인스턴스의 상태는 생성 중입니다.



    둘 모두의 상태가 사용 가능으로 변경되면 DB 인스턴스에 연결할 수 있습니다. DB 인스턴스 클래스와 스토리지의 용량에 따라 새 DB 인스턴스를 사용할 수 있을 때까지 최대 20분이 걸릴 수 있습니다.

    새로 생성된 클러스터를 보려면 Amazon RDS 콘솔의 탐색 창에서 데이터베이스를 선택합니다. 그런 다음, DB 클러스터 세부 정보를 표시할 DB 클러스터를 선택합니다. 자세한 내용은 Amazon Aurora DB 클러스터 보기 섹션을 참조하세요.



Connectivity & security(연결 및 보안) 탭에서 라이터 DB 인스턴스의 포트 및 엔드포인트를 적어둡니다. 쓰기 또는 읽기 작업을 수행하는 애플리케이션은 모두 JDBC 및 ODBC 연결 문자열에 이 클러스터의 엔드포인트와 포트를 사용합니다.

AWS CLI

> **Note**
>
> AWS CLI를 사용해 Aurora DB 클러스터를 새로 만들기 전에 반드시 VPC 및 RDS DB 서브넷 그룹 생성 같은 필수 사전 조건을 충족해야 합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.

AWS CLI를 사용하여 Aurora MySQL DB 클러스터 또는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다.

AWS CLI를 사용하여 Aurora MySQL DB 클러스터를 생성하려면

Aurora MySQL 8.0 또는 5.7 호환 DB 클러스터 또는 DB 인스턴스를 생성할 때는 `--engine` 옵션에 `aurora-mysql`을 지정해야 합니다.

다음 단계를 완료합니다.

1. 새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC 보안 그룹 ID를 확인한 다음 [create-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/create-db-cluster.html) AWS CLI 명령을 호출하여 Aurora MySQL DB 클러스터를 생성하세요.

   예를 들어, 다음 명령을 사용하면 이름이 `sample-cluster`인 새 MySQL 8.0 호환 DB 클러스터가 생성됩니다. 클러스터는 기본 엔진 버전 및 Aurora I/O-Optimized 스토리지 유형을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 8.0 \
    --storage-type aurora-iopt1 \
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

   Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 8.0 ^
    --storage-type aurora-iopt1 ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

   다음 명령을 사용하면 이름이 `sample-cluster`인 새 MySQL 5.7 호환 DB 클러스터가 생성됩니다. 클러스터는 기본 엔진 버전 및 Aurora Standard 스토리지 유형을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
    --engine aurora-mysql --engine-version 5.7 \
    --storage-type aurora \
    --master-username user-name --manage-master-user-password \
```

```
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. 콘솔을 사용하여 DB 클러스터를 생성하면 Amazon RDS에서 자동으로 DB 클러스터를 위한 기본 인스턴스(라이터)를 생성합니다. AWS CLI를 사용하여 DB 클러스터를 생성할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들 때까지 DB 클러스터 엔드포인트는 `Creating` 상태를 유지합니다.

   create-db-instanceAWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하세요. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하세요.

   > **Note**
   >
   > DB 인스턴스의 `--storage-type` 옵션은 설정할 수 없습니다. DB 클러스터에만 설정할 수 있습니다.

   예를 들어, 다음 명령을 사용하면 이름이 `sample-instance`인 새 MySQL 5.7 또는 8.0 호환 DB 인스턴스가 생성됩니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

   Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

AWS CLI를 사용하여 Aurora PostgreSQL DB 클러스터를 생성하려면

1. 새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC 보안 그룹 ID를 확인한 다음 create-db-cluster AWS CLI 명령을 호출하여 Aurora PostgreSQL DB 클러스터를 생성하세요.

   예를 들어, 다음 명령을 사용하면 이름이 `sample-cluster`인 새 DB 클러스터가 생성됩니다. 클러스터는 기본 엔진 버전 및 Aurora I/O-Optimized 스토리지 유형을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Windows의 경우:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

2. 콘솔을 사용하여 DB 클러스터를 생성하면 Amazon RDS에서 자동으로 DB 클러스터를 위한 기본 인스턴스(라이터)를 생성합니다. AWS CLI를 사용하여 DB 클러스터를 생성할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들 때까지 DB 클러스터 엔드포인트는 `Creating` 상태를 유지합니다.

   create-db-instanceAWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하세요. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하세요.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
   instance-class db.r5.large
   ```

   Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

이 예제에서는 마스터 사용자 암호를 생성하고 이를 Secrets Manager에서 관리하는 `--manage-master-user-password` 옵션도 지정합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 섹션을 참조하세요. 또는 `--master-password` 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

RDS API

> **Note**
>
> AWS CLI를 사용해 Aurora DB 클러스터를 새로 만들기 전에 반드시 VPC 및 RDS DB 서브넷 그룹 생성 같은 필수 사전 조건을 충족해야 합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.

새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC 보안 그룹 ID를 확인한 다음 CreateDBCluster 작업을 호출하여 DB 클러스터를 생성하십시오.

Aurora MySQL 버전 2 또는 3 DB 클러스터 또는 DB 인스턴스를 생성할 때는 `Engine` 파라미터에 `aurora-mysql`을 지정해야 합니다.

Aurora PostgreSQL DB 클러스터 또는 DB 인스턴스를 생성할 때는 `aurora-postgresql` 파라미터에 `Engine`을 지정하세요.

콘솔을 사용하여 DB 클러스터를 생성하면 Amazon RDS에서 자동으로 DB 클러스터를 위한 기본 인스턴스(라이터)를 생성합니다. RDS API를 사용하여 DB 클러스터를 생성할 경우 반드시 CreateDBInstance를 사용하여 DB 클러스터를 위한 프라이머리 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들 때까지 DB 클러스터 엔드포인트는 `Creating` 상태를 유지합니다.

기본(라이터) DB 인스턴스 생성

AWS Management Console을 사용하여 DB 클러스터를 생성하면 Amazon RDS에서 자동으로 DB 클러스터를 위한 기본 인스턴스(라이터)를 생성합니다. AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 생성할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인

스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들 때까지 DB 클러스터 엔드포인트는 `Creating` 상태를 유지합니다.

자세한 내용은 [DB 클러스터 생성](#) 섹션을 참조하세요.

> **Note**
> 라이터 DB 인스턴스가 없는 DB 클러스터('헤드리스 클러스터'라고도 함)가 있는 경우 콘솔을 사용하여 라이터 인스턴스를 생성할 수 없습니다. AWS CLI 또는 RDS API를 사용해야 합니다.

다음 예제에서는 [create-db-instance](#) AWS CLI 명령을 사용하여 이름이 `headless-test`로 지정된 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 생성합니다.

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Aurora DB 클러스터 설정

다음 테이블에는 Aurora DB 인스턴스를 생성할 때 선택하는 설정에 대한 세부 정보가 나와 있습니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
자동 마이너 버전 업그레이드	DB 엔진에 대한 기본 마이너 버전 업그레이드가 있을 때 Aurora DB 클러스터에서 이를 자동으로 수신하도록 하려면 마이너 버전 자동 업그레이드 사용을 선택합니다. 마이너 버전 자동 업그레이드 설정은 Aurora PostgreSQL 및 Aurora MySQL DB 클러스터 모두에 적용됩니다. Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 [Amazon](#)	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. 클러스터의 DB 인스턴스에서 이 설정이 꺼져 있으면 클러스터가 자동으로 업그레이드되지 않습니다. AWS CLI를 사용하여 [`create-db-instance`](#) 를 실행하고 `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` 옵션을 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
	Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하세요. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하세요.	RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `AutoMinorVersionUpgrade` 파라미터를 설정합니다.
AWS KMS key	Encryption(암호화)를 Enable encryption(암호화 활성화)으로 설정한 경우에만 사용할 수 있습니다. 이 DB 클러스터를 암호화하는 데 사용할 AWS KMS key을(를) 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--kms-key-id` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `KmsKeyId` 파라미터를 설정합니다.
역추적	Aurora MySQL에만 적용됩니다. 역추적을 활성화하려면 Enable Backtrack(역추적 활성화)을 선택하고 역추적을 비활성화하려면 Disable Backtrack(역추적 비활성화)을 선택합니다. 역추적을 이용하면 새 DB 클러스터를 만들지 않고 특정 시점으로 DB 클러스터를 되감을 수 있습니다. 기본적으로는 비활성화되어 있습니다. 역추적을 활성화할 경우 DB 클러스터를 역추적할 수 있는 기간(대상 역추적 기간)도 함께 지정하십시오. 자세한 내용은 Aurora DB 클러스터 역추적 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--backtrack-window` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `BacktrackWindow` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
인증 기관	DB 클러스터의 DB 인스턴스에서 사용하는 서버 인증서의 CA(인증 기관)입니다. 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-instance` 를 실행하고 `--ca-certificate-identifier` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `CACertificateIdentifier` 파라미터를 설정합니다.
클러스터 스토리지 구성	DB 클러스터의 스토리지 유형(Aurora I/O-Optimized 또는 Aurora Standard)입니다. 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--storage-type` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `StorageType` 파라미터를 설정합니다.
스냅샷으로 태그 복사	스냅샷을 생성할 때 DB 인스턴스 태그를 DB 스냅샷에 복사하려면 이 옵션을 선택합니다. 자세한 내용은 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--copy-tags-to-snapshot | --no-copy-tags-to-snapshot` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `CopyTagsToSnapshot` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
데이터베이스 인증	사용하고자 하는 데이터베이스 인증입니다. MySQL의 경우: • 데이터베이스 암호로만 데이터베이스 사용자를 인증할 수 있도록 Password authentication(암호 인증)을 선택합니다. • IAM 사용자 및 역할을 통해 데이터베이스 암호 및 사용자 자격 증명으로 데이터베이스 사용자를 인증하려면 [암호 및 IAM 데이터베이스 인증(Password and IAM database authentication)]을 선택합니다. 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요. PostgreSQL의 경우: • IAM 사용자 및 역할을 통해 데이터베이스 암호 및 사용자 자격 증명으로 데이터베이스 사용자를 인증하려면 IAM database authentication(IAM 데이터베이스 인증)을 선택합니다. 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요. • Kerberos 인증을 사용하여 데이터베이스 암호 및 사용자 자격 증명을 인증하려면 [Kerberos authentication(Kerberos 인증)]을 선택합니다. 자세한 내용은 Aurora PostgreSQL과 함께	AWS CLI에서 IAM 데이터베이스 인증을 사용하려면 `create-db-cluster` 를 실행하고 `--enable-iam-database-authentication | --no-enable-iam-database-authentication` 옵션을 설정합니다. RDS API에서 IAM 데이터베이스 인증을 사용하려면 `CreateDBCluster` 를 호출하고 `EnableIAMDatabaseAuthentication` 파라미터를 설정합니다. AWS CLI에서 Kerberos 인증을 사용하려면 `create-db-cluster`를 실행하고 `--domain` 및 `--domain-iam-role-name` 옵션을 설정합니다. RDS API에서 Kerberos 인증을 사용하려면 `CreateDBCluster` 를 호출하고 `Domain` 및 `DomainIAMRoleName` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
	Kerberos 인증 사용 섹션을 참조하세요.	
데이터베이스 포트	애플리케이션과 유틸리티가 데이터베이스에 액세스할 때 사용할 포트를 지정합니다. Aurora MySQL DB 클러스터는 기본 MySQL 포트(3306)으로, 그리고 Aurora PostgreSQL DB 클러스터는 기본 PostgreSQL 포트(5432)로 기본 설정됩니다. 일부 기업에서는 방화벽이 이러한 기본 포트 연결을 차단하는 경우도 있습니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 새로운 DB 클러스터에 다른 포트를 선택해야 합니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--port` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `Port` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
DB 클러스터 식별자	선택한 AWS 리전에 속한 계정에 고유한 DB 클러스터 이름을 입력합니다. 이 식별자는 DB 클러스터에 대한 클러스터 엔드포인트 주소로 사용됩니다. 클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora 엔드포인트 연결 단원을 참조하십시오. DB 클러스터 식별자는 다음과 같은 제약 조건이 있습니다. • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • AWS 리전별로 모든 DB 클러스터에 대해 AWS 계정당 고유해야 합니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--db-cluster-identifier` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `DBClusterIdentifier` 파라미터를 설정합니다.
DB 클러스터 파라미터 그룹	DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본값으로 사용할 수 있는 DB 클러스터 파라미터 그룹을 제공하며, 자체 DB 클러스터 파라미터 그룹을 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--db-cluster-parameter-group-name` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `DBClusterParameterGroupName` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
DB 인스턴스 클래스	프로비저닝된 용량 유형에만 적용됩니다. DB 클러스터의 각 인스턴스 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스에 대한 자세한 내용은 [Amazon Aurora DB 인스턴스 클래스](#) 섹션을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 `create-db-instance` 를 실행하고 `--db-instance-class` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `DBInstanceClass` 파라미터를 설정합니다.
DB 파라미터 그룹	파라미터 그룹을 선택합니다. Aurora에 사용할 수 있는 기본 파라미터 그룹이 포함되어 있거나, 직접 파라미터 그룹을 생성할 수 있습니다. 파라미터 그룹에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 단원을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 `create-db-instance` 를 실행하고 `--db-parameter-group-name` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `DBParameterGroupName` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
DB 서브넷 그룹	DB 클러스터에 사용할 DB 서브넷 그룹입니다. 기존 DB 서브넷 그룹을 사용하려면 Choose existing(기존 항목 선택)을 선택합니다. 그런 다음 Existing DB subnet groups(기존 DB 서브넷 그룹) 드롭다운 목록에서 필요한 서브넷 그룹을 선택합니다. RDS가 호환되는 DB 서브넷 그룹을 선택하도록 하려면 Automatic setup(자동 설정)을 선택합니다. 해당 그룹이 없으면 RDS는 클러스터에 대한 새 서브넷 그룹을 생성합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--db-subnet-group-name` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `DBSubnetGroupName` 파라미터를 설정합니다.
삭제 방지 활성화	DB 클러스터가 삭제되는 것을 방지하려면, 삭제 방지 활성화를 선택합니다. 콘솔을 사용하여 프로덕션 DB 클러스터를 생성할 경우 기본값으로 삭제 방지가 활성화됩니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--deletion-protection | --no-deletion-protection` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `DeletionProtection` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
암호화 활성화	이 DB 클러스터의 저장 시 암호화를 활성화하려면 Enable encryption 를 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 --storage-encrypted | --no-storage-encrypted 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 StorageEncrypted 파라미터를 설정합니다.
고급 모니터링 활성화	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 create-db-instance 를 실행하고 --monitoring-interval 및 --monitoring-role-arn 옵션을 설정합니다. RDS API를 사용하여 CreateDBInstance 를 호출하고 MonitoringInterval 및 MonitoringRoleArn 파라미터를 설정합니다.
RDS Data API 활성화	RDS Data API(데이터 API)를 활성화하려면 RDS Data API 활성화를 선택합니다. 데이터 API는 연결을 관리하지 않고 SQL 문을 실행할 수 있는 안전한 HTTP 엔드포인트를 제공합니다. 자세한 내용은 Amazon RDS Data API 사용 섹션을 참조하세요.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 --enable-http-endpoint | --no-enable-http-endpoint 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 EnableHttpEndpoint 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
엔진 유형	이 DB 클러스터에 사용할 데이터베이스 엔진을 선택합니다.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 `--engine` 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 `Engine` 파라미터를 설정합니다.
엔진 버전	프로비저닝된 용량 유형에만 적용됩니다. DB 엔진의 버전 번호를 선택합니다.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 `--engine-version` 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 `EngineVersion` 파라미터를 설정합니다.
장애 조치 우선 순위	인스턴스의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스의 결함으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 create-db-instance 를 실행하고 `--promotion-tier` 옵션을 설정합니다. RDS API를 사용하여 CreateDBInstance 를 호출하고 `PromotionTier` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
초기 데이터베이스 이름	기본 데이터베이스의 이름을 입력합니다. Aurora MySQL DB 클러스터의 이름을 제공하지 않으면 Amazon RDS는 DB 클러스터에서 생성하려고 하는 데이터베이스를 생성하지 않습니다. Aurora PostgreSQL DB 클러스터의 이름을 제공하지 않으면 Amazon RDS는 `postgres`라는 데이터베이스를 생성합니다. Aurora MySQL의 경우, 기본 데이터베이스 이름에는 다음과 같은 제약 조건이 적용됩니다. • 1–64자의 영숫자로 구성되어야 합니다. • 데이터베이스 엔진에서 예약한 단어는 사용할 수 없습니다. Aurora PostgreSQL의 경우, 기본 데이터베이스 이름에는 다음과 같은 제약 조건이 적용됩니다. • 1~63자의 영숫자로 구성되어야 합니다. • 문자로 시작해야 합니다. 이후 문자는 글자, 밑줄 또는 숫자(0~9)가 될 수 있습니다. • 데이터베이스 엔진에서 예약한 단어는 사용할 수 없습니다. 추가 데이터베이스를 생성하려면, DB 클러스터에 연결한 다음 SQL	AWS CLI를 사용하여 [`create-db-cluster`]() 를 실행하고 `--database-name` 옵션을 설정합니다. RDS API를 사용하여 [`CreateDBCluster`]() 를 호출하고 `DatabaseName` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
	명령어 CREATE DATABASE를 사용하십시오. DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.	
로그 내보내기	로그 내보내기 섹션에서 Amazon CloudWatch Logs에 게시하기 시작할 로그를 선택합니다. Amazon CloudWatch Logs에 Aurora MySQL 로그를 게시하는 방법은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 섹션을 참조하세요. Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시하는 방법은 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--enable-cloudwatch-logs-exports` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `EnableCloudwatchLogsExports` 파라미터를 설정합니다.
유지보수 윈도우	기간 선택을 선택하고 시스템 유지 관리를 실행할 수 있는 주 단위 기간을 지정합니다. 또는 Amazon RDS가 임의로 기간을 지정하도록 하려면 기본 설정 없음을 선택합니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--preferred-maintenance-window` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `PreferredMaintenanceWindow` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
AWS Secrets Manager에서 마스터 자격 증명 관리	Secrets Manager에서 보안 암호의 마스터 사용자 암호를 관리하려면 AWS Secrets Manager에서 마스터 자격 증명 관리를 선택합니다. 원하는 경우 보안 암호를 보호하는 데 사용할 KMS 키를 선택합니다. 자신의 계정에서 KMS 키를 선택하거나 다른 계정의 키를 입력할 수 있습니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--manage-master-user-password | --no-manage-master-user-password` 및 `--master-user-secret-kms-key-id` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `ManageMasterUserPassword` 및 `MasterUserSecretKmsKeyId` 파라미터를 설정합니다.
마스터 암호	DB 클러스터에 로그인할 암호를 입력합니다. • Aurora MySQL의 경우, 암호는 8~41자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. • Aurora PostgreSQL의 경우, 암호는 8~99자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. • /, ", @ 또는 공백을 포함할 수 없습니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--master-user-password` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `MasterUserPassword` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
마스터 사용자 이름	DB 클러스터에 로그인할 때 마스터 사용자 이름으로 사용할 이름을 입력합니다. • Aurora MySQL의 경우, 이름은 1–16자의 영숫자로 구성되어야 합니다. • Aurora PostgreSQL의 경우에는 1–63자의 영숫자로 구성되어야 합니다. • 첫 번째 자리는 문자여야 합니다. • 이름은 데이터베이스 엔진에서 예약한 단어는 사용할 수 없습니다. DB 클러스터를 생성한 후에는 마스터 사용자 이름을 변경할 수 없습니다.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 `--master-username` 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 `MasterUsername` 파라미터를 설정합니다.
다중 AZ 배포	프로비저닝된 용량 유형에만 적용됩니다. 장애 조치 지원을 위해 다른 가용 영역에 Aurora 복제본을 생성할지 여부를 결정합니다. 다른 영역에 복제본 생성을 선택하면 Amazon RDS가 DB 클러스터를 위한 기본 인스턴스와 다른 가용 영역의 DB 클러스터에 Aurora 복제본을 생성합니다. 다중 가용 영역에 대한 자세한 내용은 [리전 및 가용 영역](리전 및 가용 영역) 단원을 참조하세요.	AWS CLI를 사용하여 create-db-cluster 를 실행하고 `--availability-zones` 옵션을 설정합니다. RDS API를 사용하여 CreateDBCluster 를 호출하고 `AvailabilityZones` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
네트워크 유형	DB 클러스터에서 지원하는 IP 주소 지정 프로토콜. 리소스가 IPv4 주소 지정 프로토콜을 통해서만 DB 클러스터와 통신할 수 있도록 지정하는 IPv4. 리소스가 IPv4, IPv6 또는 모두를 통해 DB 클러스터와 통신할 수 있도록 지정하는 듀얼 스택 모드. IPv6 주소 지정 프로토콜을 통해 DB 클러스터와 통신해야 하는 리소스가 있는 경우 이중 스택 모드를 사용합니다. 듀얼 스택 모드를 사용하려면 두 개의 가용 영역에 걸쳐 IPv4 및 IPv6 네트워크 프로토콜을 모두 지원하는 서브넷이 두 개 이상 있어야 합니다. 또한 IPv6 CIDR 블록을 지정한 DB 서브넷 그룹의 서브넷과 연결해야 합니다. 자세한 내용은 Amazon Aurora IP 주소 지정 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--network-type` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `NetworkType` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
공개 액세스(Public access)	공개적으로 액세스할 수 있음(Publicly accessible)을 선택하여 DB 클러스터에 퍼블릭 IP 주소를 부여하거나 공개적으로 액세스할 수 없음(Not publicly accessible)을 선택합니다. DB 클러스터의 인스턴스는 퍼블릭과 프라이빗 DB 인스턴스를 모두 혼합하여 사용할 수 있습니다. 모든 사용자의 액세스에서 인스턴스를 숨기는 방법에 대한 자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 단원을 참조하십시오. Amazon VPC 외부에서 DB 인스턴스에 연결하려면 DB 인스턴스에 공개적으로 액세스할 수 있어야 하고, DB 인스턴스 보안 그룹의 인바운드 규칙을 사용하여 액세스 권한을 부여해야 하며, 기타 요구 사항을 충족해야 합니다. 자세한 내용은 Amazon RDS DB 인스턴스에 연결할 수 없음 섹션을 참조하세요. DB 인스턴스에 공개적으로 액세스할 수 없는 경우 AWS Site-to-Site VPN 연결 또는 AWS Direct Connect 연결을 사용하여 프라이빗 네트워크에서 액세스할 수도 있습니다. 자세한 내용은 인터네트워크 트래픽 개인 정보 섹션을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 `create-db-instance` 를 실행하고 `--publicly-accessible | --no-publicly-accessible` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `PubliclyAccessible` 파라미터를 설정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
RDS 추가 지원	지원되는 메이저 엔진 버전이 Aurora 표준 지원 종료 날짜를 지나서도 계속 실행되도록 하려면 RDS 추가 지원 활성화를 선택합니다. DB 클러스터를 생성할 때 Amazon Aurora는 기본적으로 RDS 확장 지원을 사용합니다. Aurora 표준 지원 종료일 후에 새 DB 클러스터가 생성되지 않도록 하고 RDS 확장 지원에 대한 요금이 부과되지 않도록 하려면 이 설정을 비활성화하세요. 기존 DB 클러스터에는 RDS 확장 지원 요금 시작일이 되기 전까지는 요금이 부과되지 않습니다. 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--engine-lifecycle-support` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `EngineLifecycleSupport` 파라미터를 설정합니다.
RDS 프록시	Create an RDS Proxy(RDS 프록시 생성)를 선택하여 DB 클러스터용 프록시를 생성합니다. Amazon RDS는 프록시에 대한 IAM 역할과 Secrets Manager 암호를 자동으로 생성합니다. 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.	DB 클러스터를 생성할 때는 사용할 수 없습니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
보관 기간	Aurora가 데이터베이스 백업 사본을 보관하는 기간을 1~35일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--backup-retention-period` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `BackupRetentionPeriod` 파라미터를 설정합니다.
DevOps Guru 켜기	DevOps Guru 활성화(Turn on DevOps Guru)를 선택하여 Aurora 데이터베이스에 대해 Amazon DevOps Guru를 켭니다. DevOps Guru for RDS가 성능 이상에 대한 자세한 분석을 제공하려면 성능 개선 도우미를 활성화해야 합니다. 자세한 내용은 DevOps Guru for RDS 설정 섹션을 참조하세요.	RDS 콘솔 내에서 DevOps Guru for RDS를 활성할 수 있지만, RDS API 또는 CLI로는 해당 작업을 수행할 수 없습니다. DevOps Guru를 켜는 방법에 대한 자세한 내용은 Amazon DevOps Guru 사용 설명서를 참조하세요.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
성능 개선 도우미 활성화	성능 개선 도우미 활성화(Turn on Performance Insights)를 선택하여 Amazon RDS 성능 개선 도우미를 설정합니다. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.	Aurora 클러스터의 모든 DB 인스턴스에 대해 이 값을 설정합니다. AWS CLI를 사용하여 `create-db-instance` 를 실행하고 `--enable-performance-insights` | `--no-enable-performance-insights` , `--performance-insights-kms-key-id` 및 `--performance-insights-retention-period` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBInstance` 를 호출하고 `EnablePerformanceInsights` , `PerformanceInsightsKMSKeyId` 및 `PerformanceInsightsRetentionPeriod` 파라미터를 설정합니다.
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS에서 VPC를 생성하도록 하려면 새 VPC 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.	AWS CLI 및 API의 경우 VPC 보안 그룹 ID를 지정합니다.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터
VPC 보안 그룹(방화벽)	Amazon RDS에서 VPC 보안 그룹을 생성하게 하려면 새로 생성을 선택합니다. 또는 Choose existing(기존 그룹 선택)을 선택하고 VPC 보안 그룹을 하나 이상 지정하여 DB 클러스터에 대한 네트워크 액세스를 보호합니다. RDS 콘솔에서 새로 생성을 선택하는 경우 브라우저에서 검색된 IP 주소에서 DB 인스턴스에 액세스하도록 허용하는 발신 규칙을 사용하여 새 보안 그룹이 생성됩니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.	AWS CLI를 사용하여 `create-db-cluster` 를 실행하고 `--vpc-security-group-ids` 옵션을 설정합니다. RDS API를 사용하여 `CreateDBCluster` 를 호출하고 `VpcSecurityGroupIds` 파라미터를 설정합니다.

DB 클러스터용 Amazon Aurora에 적용되지 않는 설정

다음과 같은 AWS CLI 명령 `create-db-cluster` 및 RDS API 작업 `CreateDBCluster` 의 설정은 Amazon Aurora DB 클러스터에 적용되지 않습니다.

> **Note**
>
> AWS Management Console은 Aurora DB 클러스터에 대한 이러한 설정을 표시하지 않습니다.

AWS CLI 설정	RDS API 설정
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade` | `--no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`

AWS CLI 설정	RDS API 설정
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`
`--publicly-accessible | --no-publicly-accessible`	`PubliclyAccessible`

DB 인스턴스용 Amazon Aurora에 적용되지 않는 설정

다음과 같은 AWS CLI 명령 [`create-db-instance`](create-db-instance) 및 RDS API 작업 [`CreateDBInstance`](CreateDBInstance) 의 설정은 DB 인스턴스 Amazon Aurora DB 클러스터에 적용되지 않습니다.

> **Note**
>
> AWS Management Console은 Aurora DB 인스턴스에 대한 이러한 설정을 표시하지 않습니다.

AWS CLI 설정	RDS API 설정
`--allocated-storage`	`AllocatedStorage`
`--availability-zone`	`AvailabilityZone`

AWS CLI 설정	RDS API 설정
--backup-retention-period	BackupRetentionPeriod
--backup-target	BackupTarget
--character-set-name	CharacterSetName
--character-set-name	CharacterSetName
--custom-iam-instance-profile	CustomIamInstanceProfile
--db-security-groups	DBSecurityGroups
--deletion-protection | --no-deletion-protection	DeletionProtection
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-cloudwatch-logs-exports	EnableCloudwatchLogsExports
--enable-customer-owned-ip | --no-enable-customer-owned-ip	EnableCustomerOwnedIp
--enable-iam-database-authentication | --no-enable-iam-database-authentication	EnableIAMDatabaseAuthentication
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--master-username	MasterUsername
--master-user-password	MasterUserPassword
--max-allocated-storage	MaxAllocatedStorage

AWS CLI 설정	RDS API 설정
`--multi-az | --no-multi-az`	`MultiAZ`
`--nchar-character-set-name`	`NcharCharacterSetName`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--preferred-backup-window`	`PreferredBackupWindow`
`--processor-features`	`ProcessorFeatures`
`--storage-encrypted | --no-storage-encrypted`	`StorageEncrypted`
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--timezone`	`Timezone`
`--vpc-security-group-ids`	`VpcSecurityGroupIds`

AWS CloudFormation을 사용하여 Amazon Aurora 리소스 생성

Amazon Aurora는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 등)를 설명하는 템플릿을 생성하고 AWS CloudFormation은 해당 리소스를 자동으로 프로비저닝하고 구성합니다.

AWS CloudFormation을 사용할 때 템플릿을 재사용하여 Aurora 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번 설명한 후 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

Aurora과AWS CloudFormation템플릿

AWS CloudFormation 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer 이란 무엇입니까?를 참조하세요.

Aurora는 AWS CloudFormation에서 리소스 생성을 지원합니다. 이러한 리소스에 대한 JSON 및 YAML 템플릿의 예를 비롯한 자세한 내용은 AWS CloudFormation 사용 설명서에서 RDS 리소스 유형 참조를 참조하세요.

AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API 참조
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

Amazon Aurora DB 클러스터에 연결

MySQL 또는 PostgreSQL 데이터베이스에 연결할 때 사용한 것과 동일한 도구를 사용하여 Aurora DB 클러스터에 연결할 수 있습니다. MySQL 또는 PostgreSQL DB 인스턴스에 연결되는 모든 스크립트, 유틸리티 또는 애플리케이션에서 연결 문자열을 설정합니다. 그리고 동일한 퍼블릭 키를 사용하여 Secure Sockets Layer(SSL)를 연결합니다.

연결 문자열에서는 보통 DB 클러스터에 연결된 특별 엔드포인트에서 나온 호스트 및 포트 정보를 사용합니다. 이들 엔드포인트에서는 클러스터의 DB 인스턴스 수에 관계 없이 동일한 연결 파라미터를 사용할 수 있습니다. 문제 해결 같은 전문 작업의 경우, Aurora DB 클러스터의 특별 DB 인스턴스에 나온 호스트 및 포트 정보를 사용할 수 있습니다.

> **Note**
>
> Aurora Serverless DB 클러스터의 경우 DB 인스턴스가 아닌 데이터베이스 엔드포인트에 연결합니다. Aurora Serverless DB 클러스터의 데이터베이스 엔드포인트는 AWS Management Console의 연결 및 보안 탭에서 찾을 수 있습니다. 자세한 내용은 Amazon Aurora Serverless v1 사용 섹션을 참조하세요.

Aurora DB 엔진과 DB 클러스터 또는 인스턴스 작업에 사용하는 특정 도구에 관계없이 엔드포인트에 액세스할 수 있어야 합니다. Aurora DB 클러스터는 Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)에만 생성해야 합니다. 즉, 다음 방법 중 하나를 사용하여 VPC 내부 또는 VPC 외부에서 엔드포인트에 액세스합니다.

- VPC 내의 Aurora DB 클러스터에 액세스 – VPC를 통해 AuroraDB 클러스터에 액세스할 수 있습니다. 이렇게 하려면 특정 Aurora DB 클러스터에 대한 액세스를 허용하도록 VPC의 보안 그룹에서 인바운드 규칙을 편집합니다. 다양한 Aurora DB 클러스터 시나리오에 맞게 VPC를 구성하는 방법을 포함하여 자세한 내용은 Amazon Virtual Private Cloud(VPC) 및 Amazon Aurora를 참조하세요.
- VPC 외부의 Aurora DB 클러스터에 액세스 – VPC 외부에서 Aurora DB 클러스터에 액세스하려면 DB 클러스터의 퍼블릭 엔드포인트 주소를 사용합니다.

자세한 내용은 Aurora 연결 장애 문제 해결 섹션을 참조하세요.

목차

AWS 드라이버를 사용하여 Aurora DB 클러스터에 연결

더 빠른 전환 및 장애 조치 시간, AWS Secrets Manager, AWS Identity and Access Management(IAM) 및 페더레이션 ID를 사용한 인증을 지원하도록 설계된 AWS 드라이버 제품군입니다. AWS 드라이버는 DB 클러스터 상태 모니터링과 클러스터 토폴로지 파악을 통해 새 라이터를 결정합니다. 이 접근 방식은 전환 및 장애 조치 시간을 오픈 소스 드라이버의 경우 수십 초였던 것에 비해 10초 미만으로 단축합니다.

다음 표에는 각 드라이버에 지원되는 기능이 나와 있습니다. 새로운 서비스 기능이 도입됨에 따라 AWS 드라이버 제품군의 목표는 이러한 서비스 기능에 대한 지원을 기본 제공하는 것입니다.

Feature	AWS JDBC 드라이버	AWS Python 드라이버	MySQL용 AWS ODBC 드라이버	AWS Advanced NodeJS Wrapper
장애 조치 지원	예	예	예	예
향상된 장애 조치 모니터링	예	예	예	예
읽기/쓰기 분할	예	예	아니요	예

Feature	AWS JDBC 드라이버	AWS Python 드라이버	MySQL용 AWS ODBC 드라이버	AWS Advanced NodeJS Wrapper
Aurora 연결 추적기	예	예	아니요	예
드라이버 메타데이터 연결	예	해당 사항 없음	해당 사항 없음	N/A
원격 측정	예	예	아니요	예
Secrets Manager	예	예	예	예
IAM 인증.	예	예	예	예
페더레이션 ID(AD FS)	예	예	아니요	예
페더레이션 ID(Okta)	예	예	예	예
Aurora PostgreSQL Limitless Database	예(Aurora PostgreSQL만 해당)	아니요	아니요	예(Aurora PostgreSQL만 해당)

AWS 드라이버에 대한 자세한 내용은 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터의 해당 언어 드라이버를 참조하세요.

Amazon Aurora MySQL DB 클러스터에 연결

Aurora MySQL DB 클러스터에 대해 인증하려면, MySQL 사용자 이름 및 암호 인증 또는 AWS Identity and Access Management(IAM) 데이터베이스 인증을 사용할 수 있습니다. MySQL 사용자 이름 및 암호 인증 사용에 대한 자세한 정보는 MySQL 설명서의 액세스 제어 및 계정 관리를 참조하십시오. IAM 데이터베이스 인증 사용에 대한 자세한 내용은 IAM 데이터베이스 인증 단원을 참조하세요.

MySQL 8.0과 호환되는 Amazon Aurora DB 클러스터에 연결되어 있으면 MySQL 버전 8.0과 호환되는 SQL 명령을 실행할 수 있습니다. 최소 호환 버전은 MySQL 8.0.23입니다. MySQL 8.0 SQL 구문에 대

한 자세한 정보는 MySQL 8.0 참조 매뉴얼을 참조하세요. Aurora MySQL 3에 적용되는 제한 사항에 대한 자세한 정보는 Aurora MySQL 버전 3과 MySQL 8.0 커뮤니티 에디션 비교 섹션을 참조하세요.

MySQL 5.7과 호환되는 Amazon Aurora DB 클러스터에 연결되어 있으면 MySQL 버전 5.7과 호환되는 SQL 명령을 실행할 수 있습니다. MySQL 5.7 SQL 구문에 대한 자세한 정보는 MySQL 5.7 참조 매뉴얼을 참조하십시오. Aurora MySQL 5.7에 적용되는 제한 사항에 대한 자세한 정보는 Aurora MySQL 버전 2는 MySQL 5.7과 호환을 참조하십시오.

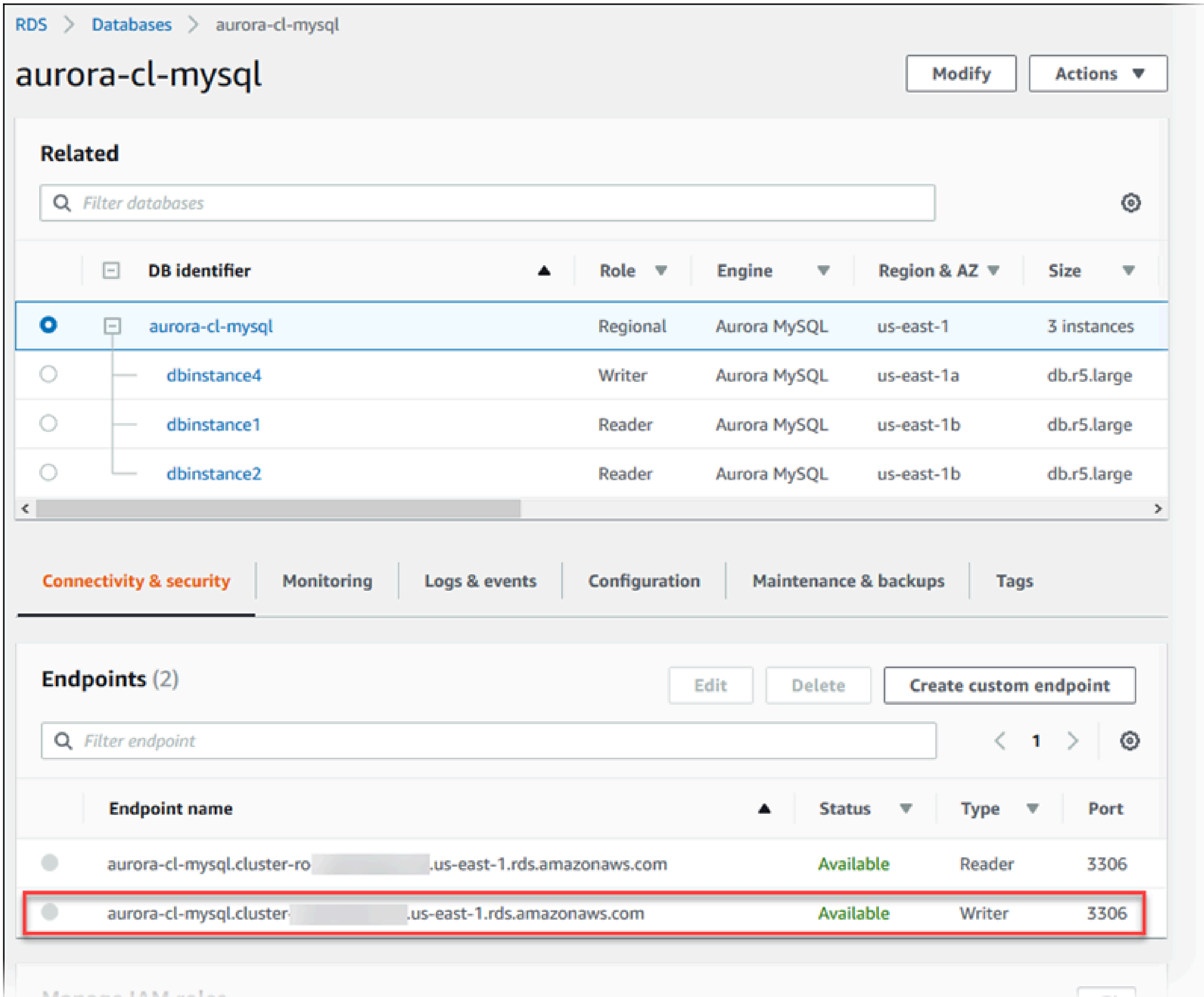
> **Note**
>
> Amazon Aurora MySQL DB 클러스터에 연결하는 데 도움이 되는 자세한 안내는 Aurora 연결 관리 핸드북에서 확인할 수 있습니다.

DB 클러스터의 세부 정보 보기에서 MySQL 연결 문자열에 사용할 수 있는 클러스터 엔드포인트를 확인할 수 있습니다. 엔드포인트는 DB 클러스터의 도메인 이름과 포트로 구성되어 있습니다. 예를 들어 엔드포인트 값이 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`이라면 MySQL 연결 문자열에 다음과 같이 값을 지정합니다.

- 호스트 또는 호스트 이름은 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`으로 지정합니다.
- 포트의 경우, DB 클러스터를 생성할 때 사용한 포트 값 또는 `3306`를 지정합니다

클러스터 엔드포인트는 DB 클러스터의 기본 인스턴스에 연결하는 데 사용되며, 이를 통해 읽기 및 쓰기 연산을 실행할 수 있습니다. 또한 DB 클러스터에 DB 클러스터의 데이터에 대한 읽기 전용 액세스를 지원하는 Aurora 복제본이 최대 15개 있을 수 있습니다. 기본 인스턴스와 각 Aurora 복제본에는 클러스터 엔드포인트와 독립적이고 클러스터의 특정 DB 인스턴스에 직접 연결할 수 있는 고유의 엔드포인트가 있습니다. 클러스터 엔드포인트는 항상 기본 인스턴스를 가리킵니다. 기본 인스턴스에 장애가 발생하여 교체되는 경우 클러스터 엔드포인트는 새로운 기본 인스턴스를 가리킵니다.

클러스터 엔드포인트(라이터 엔드포인트)를 보려면 Amazon RDS 콘솔에서 Databases(데이터베이스)를 선택하고 DB 클러스터의 이름을 선택하여 DB 클러스터 세부 정보를 표시하십시오.



주제

- [Aurora MySQL 연결 유틸리티]
- [MySQL 유틸리티를 사용하여 Aurora MySQL에 연결]
- [Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora MySQL에 연결]
- [Amazon Web Services(AWS) Python 드라이버를 사용하여 Aurora MySQL에 연결]
- [MySQL용 Amazon Web Services(AWS) ODBC 드라이버를 사용하여 Aurora MySQL에 연결]
- [Amazon Web Services(AWS) Advanced NodeJS Wrapper를 사용하여 Aurora MySQL에 연결]
- [SSL을 사용하여 Aurora MySQL에 연결]

Aurora MySQL 연결 유틸리티

다음과 같은 연결 유틸리티를 사용할 수 있습니다.

- 명령줄 – MySQL 명령줄 유틸리티 같은 도구를 사용하여 Amazon Aurora DB 클러스터에 연결할 수 있습니다. MySQL 유틸리티 사용에 대한 자세한 정보는 MySQL 설명서의 mysql - MySQL 명령줄 클라이언트를 참조하세요.
- GUI – MySQL Workbench 유틸리티를 통해 UI 인터페이스를 사용하여 연결할 수 있습니다. 자세한 내용은 MySQL Workbench 다운로드 페이지를 참조하세요.
- AWS 드라이버
  - Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora MySQL에 연결
  - Amazon Web Services(AWS) Python 드라이버를 사용하여 Aurora MySQL에 연결
  - MySQL용 Amazon Web Services(AWS) ODBC 드라이버를 사용하여 Aurora MySQL에 연결
  - Amazon Web Services(AWS) Advanced NodeJS Wrapper를 사용하여 Aurora MySQL에 연결

MySQL 유틸리티를 사용하여 Aurora MySQL에 연결

다음 절차를 따르세요. VPC의 프라이빗 서브넷에 DB 클러스터를 구성했다고 가정합니다. 자습서: 웹 서버 및 Amazon Aurora DB 클러스터 생성의 자습서에 따라 구성한 Amazon EC2 인스턴스를 사용하여 연결합니다.

> **Note**
> 이 절차에서는 자습서에서 웹 서버를 설치할 필요가 없지만, MariaDB 10.5는 설치해야 합니다.

MySQL 유틸리티를 사용하여 DB 클러스터를 연결하려면

1. DB 클러스터에 연결하는 데 사용하는 EC2 인스턴스에 로그인합니다.

   다음과 유사한 출력 화면이 표시되어야 합니다.

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|
```

```
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. 명령 프롬프트에 다음 명령을 입력하여 DB 클러스터의 프라이머리 DB 인스턴스에 연결합니다.

   -h 파라미터에서 기본 인스턴스의 엔드포인트 DNS 이름을 교체합니다. -u 파라미터에서 데이터베이스 사용자 계정의 사용자 ID를 교체합니다.

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   예:

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. 데이터베이스 사용자의 암호를 입력합니다.

   다음과 유사한 출력 화면이 표시되어야 합니다.

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   MySQL [(none)]>
   ```

4. SQL 명령을 입력합니다.

Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora MySQL에 연결

Amazon Web Services(AWS) JDBC 드라이버는 고급 JDBC 래퍼로 설계되었습니다. 이 래퍼는 기존 JDBC 드라이버를 보완하고 기능을 확장하여 애플리케이션이 Aurora MySQL과 같은 클러스터링된 데이터베이스의 기능을 활용할 수 있도록 합니다. 이 드라이버는 커뮤니티 MySQL Connector/J 드라이버 및 커뮤니티 MariaDB Connector/J 드라이버와 드롭인 호환됩니다.

AWS JDBC 드라이버를 설치하려면 CLASSPATH 애플리케이션에 있는 AWS JDBC 드라이버 .jar 파일을 추가하고 해당 커뮤니티 드라이버에 대한 참조를 보관해 두세요. 다음과 같이 해당 연결 URL 접두사를 업데이트하세요.

- `jdbc:mysql://~jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://~jdbc:aws-wrapper:mariadb://`

AWS JDBC 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 [Amazon Web Services (AWS) JDBC Driver GitHub repository]를 참조하세요.

> **Note**
>
> MariaDB Connector/J 유틸리티 버전 3.0.3은 Aurora DB 클러스터에 대한 지원을 중단하므로 AWS JDBC 드라이버로 이동할 것을 권장합니다.

Amazon Web Services(AWS) Python 드라이버를 사용하여 Aurora MySQL에 연결

Amazon Web Services(AWS) Python 드라이버는 고급 Python 래퍼로 설계되었습니다. 이 래퍼는 오픈 소스 Psycopg 드라이버의 기능을 보완하고 확장합니다. AWS Python 드라이버는 Python 버전 3.8 이상을 지원합니다. `pip` 명령을 사용하여 `psycopg` 오픈 소스 패키지와 함께 `aws-advanced-python-wrapper` 패키지를 설치할 수 있습니다.

AWS Python 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 [Amazon Web Services (AWS) JDBC Python GitHub repository]를 참조하세요.

MySQL용 Amazon Web Services(AWS) ODBC 드라이버를 사용하여 Aurora MySQL에 연결

MySQL용 AWS ODBC 드라이버는 Aurora MySQL의 고가용성을 위해 설계된 클라이언트 드라이버입니다. 드라이버는 MySQL Connector/ODBC 드라이버와 함께 존재할 수 있으며 동일한 워크플로와 호환됩니다.

MySQL용 AWS ODBC 드라이버에 대한 자세한 내용 및 설치와 사용에 대한 전체 지침은 [MySQL용 Amazon Web Services(AWS) ODBC 드라이버] GitHub 리포지토리를 참조하세요.

Amazon Web Services(AWS) Advanced NodeJS Wrapper를 사용하여 Aurora MySQL에 연결

AWS Advanced NodeJS Wrapper는 기존 NodeJS 드라이버의 기능을 보완하고 확장합니다. 이를 통해 애플리케이션은 Aurora MySQL과 같은 클러스터링된 데이터베이스의 기능을 활용할 수 있습니다.

AWS Advanced NodeJS Wrapper에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 [Amazon Web Services (AWS) Advanced NodeJS Wrapper GitHub repository]를 참조하세요.

SSL을 사용하여 Aurora MySQL에 연결

Aurora MySQL DB 인스턴스에 연결할 때는 SSL 암호화를 사용할 수 있습니다. 자세한 내용은 Aurora MySQL DB 클러스터에 대한 TLS 연결 단원을 참조하세요.

SSL을 사용하여 연결하려면 다음 절차에 따라 MySQL 유틸리티를 사용합니다. IAM 데이터베이스 인증을 사용하고 있다면, SSL 연결을 사용해야 합니다. 자세한 정보는 IAM 데이터베이스 인증 섹션을 참조하세요.

> **Note**
>
> SSL을 사용하여 클러스터 엔드포인트에 연결하려면 클라이언트 연결 유틸리티에서 Subject Alternative Names(SAN)를 지원해야 합니다. 클라이언트 연결 유틸리티에서 SAN을 지원하지 않는 경우, Aurora DB 클러스터에서 인스턴스에 직접 연결할 수 있습니다. Aurora 엔드포인트에 대한 자세한 정보는 Amazon Aurora 엔드포인트 연결 단원을 참조하십시오.

MySQL 유틸리티를 사용하여 SSL에 DB 클러스터를 연결하려면

1. Amazon RDS 서명 인증서의 퍼블릭 키를 다운로드합니다.

   인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오.

2. MySQL 유틸리티에서 SSL이 포함된 DB 클러스터의 기본 인스턴스에 연결하려면 명령 프롬프트에 다음 명령을 입력합니다. -h 파라미터에서 기본 인스턴스의 엔드포인트 DNS 이름을 교체합니다. -u 파라미터에서 데이터베이스 사용자 계정의 사용자 ID를 교체합니다. `--ssl-ca` 파라미터에는 해당하는 SSL 인증서 파일 이름으로 대체합니다. 입력 프롬프트가 표시되면 마스터 사용자 암호를 입력합니다.

```
mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u
admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-
cert
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

RDS for MySQL 연결 문자열 구성 및 SSL 연결용 퍼블릭 키 검색에 대한 일반적인 지침은 MySQL 데이터베이스 엔진을 실행하는 DB 인스턴스에 연결을 참조하세요.

Amazon Aurora PostgreSQL DB 클러스터에 연결

PostgreSQL 데이터베이스에 연결할 때와 동일한 도구를 사용하여 Amazon Aurora PostgreSQL DB 클러스터의 DB 인스턴스에 연결할 수 있습니다. 이 설정의 일환으로 동일한 퍼블릭 키를 사용하여 Secure Sockets Layer(SSL)를 연결합니다. Aurora PostgreSQL DB 클러스터에 속한 기본 인스턴스나 Aurora 복제본의 엔드포인트 및 포트 정보를 PostgreSQL DB 인스턴스에 연결하는 모든 스크립트, 유틸리티 또는 애플리케이션의 연결 문자열에 사용할 수 있습니다. 연결 문자열에는 호스트 파라미터로 기본 인스턴스 또는 Aurora 복제본 엔드포인트의 DNS 주소를 지정하십시오. 엔드포인트의 포트 번호를 포트 파라미터로 지정하십시오.

Amazon Aurora PostgreSQL DB 클러스터의 DB 인스턴스에 연결한 경우 PostgreSQL과 호환되는 SQL 명령을 실행할 수 있습니다.

Aurora PostgreSQL DB 클러스터의 세부 정보 보기에서 클러스터 엔드포인트 이름, 상태, 유형 및 포트 번호를 검색할 수 있습니다. PostgreSQL 포트 번호에서 이 엔드포인트를 사용합니다. 예를 들어 엔드포인트 값이 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`이라면 PostgreSQL 연결 문자열에 다음과 같이 값을 지정합니다.

- 호스트 또는 호스트 이름은 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`으로 지정합니다.
- 포트의 경우, DB 클러스터를 생성할 때 사용한 포트 값 또는 `5432`를 지정합니다

클러스터 엔드포인트는 DB 클러스터의 기본 인스턴스에 연결하는 데 사용되며, 이를 통해 읽기 및 쓰기 연산을 실행할 수 있습니다. 또한 DB 클러스터에 DB 클러스터의 데이터에 대한 읽기 전용 액세스를 지원하는 Aurora 복제본이 최대 15개 있을 수 있습니다. Aurora 클러스터의 각 DB 인스턴스(즉, 기본 인스턴스와 각 Aurora 복제본)는 클러스터 엔드포인트와 무관한 고유한 엔드포인트를 가집니다. 이 고유 엔드포인트를 통해 클러스터의 특정 DB 인스턴스에 직접 연결할 수 있습니다. 클러스터 엔드포인트는 항상 기본 인스턴스를 가리킵니다. 기본 인스턴스에 결함이 발생하여 교체되면 클러스터 엔드포인트는 새로운 기본 인스턴스로 향하게 됩니다.

클러스터 엔드포인트(라이터 엔드포인트)를 보려면 Amazon RDS 콘솔에서 Databases(데이터베이스)를 선택하고 DB 클러스터의 이름을 선택하여 DB 클러스터 세부 정보를 표시하십시오.

Aurora PostgreSQL 연결 유틸리티

다음과 같은 연결 유틸리티를 사용할 수 있습니다.

- 명령줄 – PostgreSQL 대화식 터미널인 psql과 같은 도구를 사용하여 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다. PostgreSQL 대화식 터미널 사용에 대한 자세한 정보는 PostgreSQL 설명서에서 psql을 참조하십시오.
- GUI – pgAdmin 유틸리티를 사용하여 UI 인터페이스로 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다. 자세한 정보는 pgAdmin 웹사이트에서 Download 페이지 단원을 참조하십시오.
- AWS 드라이버
  - Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora PostgreSQL에 연결
  - Amazon Web Services(AWS) Python 드라이버를 사용하여 Aurora PostgreSQL에 연결
  - Amazon Web Services(AWS) Advanced NodeJS Wrapper를 사용하여 Aurora PostgreSQL에 연결

Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora PostgreSQL에 연결

Amazon Web Services(AWS) JDBC 드라이버는 고급 JDBC 래퍼로 설계되었습니다. 이 래퍼는 기존 JDBC 드라이버를 보완하고 기능을 확장하여 애플리케이션이 Aurora PostgreSQL과 같은 클러스터링된 데이터베이스의 기능을 활용할 수 있도록 합니다. 이 드라이버는 커뮤니티 pgJDBC 드라이버와 드롭인 호환됩니다.

AWS JDBC 드라이버를 설치하려면 `CLASSPATH` 애플리케이션에 있는 AWS JDBC 드라이버 .jar 파일을 추가하고 pgJDBC 커뮤니티 드라이버에 대한 참조를 보관해 두세요. 연결 URL 접두사를 `jdbc:postgresql://`에서 `jdbc:aws-wrapper:postgresql://`로 업데이트하세요.

AWS JDBC 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 Amazon Web Services (AWS) JDBC Driver GitHub repository를 참조하세요.

Amazon Web Services(AWS) Python 드라이버를 사용하여 Aurora PostgreSQL에 연결

Amazon Web Services(AWS) Python 드라이버는 고급 Python 래퍼로 설계되었습니다. 이 래퍼는 오픈 소스 Psycopg 드라이버의 기능을 보완하고 확장합니다. AWS Python 드라이버는 Python 버전 3.8 이상을 지원합니다. `pip` 명령을 사용하여 `psycopg` 오픈 소스 패키지와 함께 `aws-advanced-python-wrapper` 패키지를 설치할 수 있습니다.

AWS Python 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 Amazon Web Services (AWS) JDBC Python GitHub repository를 참조하세요.

Amazon Web Services(AWS) Advanced NodeJS Wrapper를 사용하여 Aurora PostgreSQL에 연결

AWS Advanced NodeJS Wrapper는 기존 NodeJS 드라이버의 기능을 보완하고 확장합니다. 이를 통해 애플리케이션은 Aurora PostgreSQL과 같은 클러스터 데이터베이스의 기능을 활용할 수 있습니다.

AWS Advanced NodeJS Wrapper에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 Amazon Web Services (AWS) Advanced NodeJS Wrapper GitHub repository를 참조하세요.

Aurora 연결 장애 문제 해결

새 Aurora DB 클래스에 공통적으로 발생하는 연결 실패의 원인은 다음과 같습니다.

- VPC의 보안 그룹이 액세스를 허용하지 않음 – VPC는 VPC의 보안 그룹을 적절히 구성하여 디바이스 또는 Amazon EC2 인스턴스로부터의 연결을 허용해야 합니다. 이 문제를 해결하려면 연결을 허용하도록 VPC의 보안 그룹 인바운드 규칙을 수정하십시오. 관련 예제는 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 섹션을 참조하세요
- 방화벽 규칙에 의해 차단된 포트 – Aurora DB 클러스터에 대해 구성된 포트 값을 확인합니다. 방화벽 규칙이 해당 포트를 차단하는 경우 다른 포트를 사용하여 인스턴스를 다시 생성할 수 있습니다.
- IAM 구성이 완료되지 않았거나 잘못된 경우 – IAM–기반 인증을 사용하기 위해 Aurora DB 인스턴스를 생성한 경우 해당 인스턴스가 제대로 구성되어 있는지 확인합니다. 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요.

문제 해결 Aurora DB 연결에 대한 자세한 내용은 Amazon RDS DB 인스턴스에 연결할 수 없음 단원을 참조하세요.

Amazon Aurora의 파라미터 그룹

[데이터베이스 파라미터(Database parameters)]에서 데이터베이스 구성 방법을 지정합니다. 예를 들어 데이터베이스 파라미터는 메모리를 비롯하여 데이터베이스에 할당할 리소스의 양을 지정할 수 있습니다.

DB 인스턴스 및 Aurora DB 클러스터를 파라미터 그룹과 연결하여 데이터베이스 구성을 관리합니다. Aurora는 기본 설정으로 파라미터 그룹을 정의합니다. 맞춤형 설정으로 자신만의 파라미터 그룹을 정의할 수 있습니다.

주제

파라미터 그룹 개요

DB 클러스터 파라미터 그룹은 Aurora DB 클러스터의 모든 DB 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다. 예를 들어 Aurora 공유 스토리지 모델에서는 Aurora 클러스터의 모든 DB 인스턴스가 `innodb_file_per_table`과 같은 파라미터에 동일한 설정을 사용해야 합니다. 따라서 물리적 스토리지 레이아웃에 영향을 미치는 파라미터는 클러스터 파라미터 그룹의 일부입니다. DB 클러스터 파라미터 그룹에는 모든 인스턴스 수준 파라미터의 기본값도 들어 있습니다.

DB 파라미터 그룹은 하나 이상의 DB 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다. DB 파라미터 그룹은 Amazon RDS와 Aurora 모두에 있는 DB 인스턴스에 적용됩니다. 이 구성 설정은 메모리 버퍼 크기와 같은 Aurora 클러스터 내의 DB 인스턴스 사이에서 변화할 수 있는 속성에 적용됩니다.

주제

- 문자 집합 파라미터
- 지원되는 파라미터 및 파라미터 값

기본 및 사용자 지정 파라미터 그룹

DB 파라미터 그룹을 지정하지 않고 DB 인스턴스를 만드는 경우 DB 인스턴스에서는 기본 DB 파라미터 그룹을 사용합니다. 이와 마찬가지로 DB 클러스터 파라미터 그룹을 지정하지 않고 Aurora 클러스터를 생성할 경우 이 DB 클러스터에서는 기본 DB 클러스터 파라미터 그룹을 사용합니다. 각 기본 파라미터 그룹에는 인스턴스의 엔진, 컴퓨팅 클래스 및 할당된 스토리지에 따른 데이터베이스 엔진 기본값과 Amazon RDS 시스템 기본값이 들어 있습니다.

기본 DB 파라미터 그룹의 파라미터 설정은 수정할 수 없습니다. 대신에 다음 작업을 할 수 있습니다.

1. 새 파라미터 그룹을 생성해야 합니다.
2. 원하는 파라미터의 설정을 변경합니다. 파라미터 그룹에서 모든 DB 엔진 파라미터를 수정할 수 있는 것은 아닙니다.
3. DB 인스턴스 또는 DB 클러스터를 수정하여 새로운 파라미터 그룹을 연결하세요.

   DB 클러스터 또는 DB 인스턴스 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

> **Note**
>
> 사용자 지정 파라미터 그룹을 사용하도록 DB 인스턴스를 수정하고 DB 인스턴스를 시작하면 RDS는 시작 프로세스의 일부로 DB 인스턴스를 자동으로 재부팅합니다. AlwaysOn 또는 미러링 옵션이 활성화된 RDS for SQL Server 다중 AZ 인스턴스의 경우 시작 프로세스 후 인스턴스를 재부팅하면 장애 조치가 수행될 수 있습니다.

RDS는 DB 인스턴스가 재부팅된 후에만 수정된 정적 및 동적 파라미터를 새로 연결된 파라미터 그룹에 적용합니다. 그러나 DB 파라미터 그룹을 DB 인스턴스에 연결한 후 DB 파라미터 그룹에서 동적 파라미터를 수정하면 이러한 변경 사항이 재부팅 없이 즉시 적용됩니다. DB 파라미터 그룹 변경에 대한 자세한 내용은 [g26]???[/g26][g25]Amazon Aurora DB 클러스터 수정[/g25] 단원을 참조하세요.

DB 파라미터 그룹 내의 파라미터를 업데이트하는 경우 변경 사항은 이 파라미터 그룹과 연결된 모든 DB 인스턴스에 적용됩니다. 이와 마찬가지로 Aurora DB 클러스터 파라미터 그룹 내의 파라미터를 업데이트할 경우, 변경 사항은 이 DB 클러스터 파라미터 그룹과 연결된 모든 Aurora DB 클러스터에 적용됩니다.

파라미터 그룹을 처음부터 생성하지 않으려면 AWS CLI [copy-db-parameter-group](#) 명령 또는 [copy-db-cluster-parameter-group](#) 명령을 사용하여 기존 파라미터 그룹을 복사할 수 있습니다. 경우에 따라 파라미터 그룹을 복사하는 것이 유용할 수 있습니다. 예를 들어, 기존 파라미터 그룹의 사용자 지정 파라미터 및 값의 대부분을 새 파라미터 그룹에 포함하고자 할 수 있습니다.

정적 및 동적 DB 클러스터 파라미터

DB 클러스터 파라미터는 정적이거나 동적입니다. 하지만 다음과 같은 차이가 있습니다.

- 정적 파라미터를 변경하고 DB 클러스터 파라미터 그룹을 저장하면 연결된 각 DB 클러스터에서 DB 인스턴스를 수동으로 재부팅한 후에 파라미터 변경 내용이 적용됩니다. AWS Management Console을 사용하여 DB 클러스터 파라미터 값을 변경하는 경우에는 `ApplyMethod`로 항상 `pending-reboot`를 사용합니다.
- 동적 파라미터를 변경하면 기본적으로 파라미터 변경이 재부팅 없이 즉시 적용됩니다. 콘솔을 사용할 때는 `ApplyMethod`로 항상 `immediate`를 사용합니다. 연결된 DB 클러스터 내의 DB 인스턴스가 재부팅된 후로 파라미터 변경을 연기하려면 AWS CLI 또는 RDS API를 사용합니다. 파라미터 변경을 위해 `ApplyMethod`를 `pending-reboot`로 설정합니다.

파라미터 값 변경을 위한 AWS CLI 사용과 관련한 자세한 내용은 [modify-db-cluster-parameter-group](#) 섹션을 참조하세요. 파라미터 값 변경을 위한 RDS API 사용과 관련한 자세한 내용은 [ModifyDBClusterParameterGroup](#) 섹션을 참조하세요.

DB 클러스터와 연결된 DB 클러스터 파라미터 그룹을 변경하면 DB 클러스터의 DB 인스턴스를 재부팅합니다. 재부팅하면 변경 사항이 DB 클러스터의 모든 DB 인스턴스에 적용됩니다. 변경 사항을 적용하기 위해 DB 클러스터의 DB 인스턴스를 재부팅해야 하는지 여부를 확인하려면 다음 AWS CLI 명령을 실행합니다.

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

출력에서 기본 DB 인스턴스의 `DBClusterParameterGroupStatus` 값을 확인합니다. 값이 `pending-reboot`인 경우, DB 클러스터의 DB 인스턴스를 재부팅합니다.

정적 및 동적 DB 인스턴스 파라미터

DB 인스턴스 파라미터는 정적이거나 동적입니다. 이들은 다음과 같은 차이가 있습니다.

- 정적 파라미터를 변경하고 DB 파라미터 그룹을 저장한 후 연결된 DB 인스턴스를 수동으로 재부팅하면 파라미터 변경 내용이 적용됩니다. 정적 파라미터의 경우 콘솔은 `ApplyMethod`로 항상 `pending-reboot`를 사용합니다.

- 동적 파라미터를 변경하면 기본적으로 파라미터 변경이 재부팅 없이 즉시 적용됩니다. AWS Management Console을 사용하여 DB 인스턴스 파라미터 값을 변경하는 경우에는 동적 파라미터의 `ApplyMethod`로 항상 `immediate`를 사용합니다. 연결된 DB 인스턴스가 재부팅된 후로 파라미터 변경을 연기하려면 AWS CLI 또는 RDS API를 사용합니다. 파라미터 변경을 위해 `ApplyMethod`를 `pending-reboot`로 설정합니다.

파라미터 값 변경을 위한 AWS CLI 사용과 관련한 자세한 내용은 modify-db-parameter-group 섹션을 참조하세요. 파라미터 값 변경을 위한 RDS API 사용과 관련한 자세한 내용은 ModifyDBParameterGroup 섹션을 참조하세요.

DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않은 경우 콘솔에 DB 파라미터 그룹이 재시작-보류 중 상태로 표시됩니다. 이 상태로 인해 다음번 유지 관리 기간 중에 자동 재부팅이 되지는 않습니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다.

문자 집합 파라미터

DB 클러스터를 만들기 전에 파라미터 그룹에 있는 데이터베이스의 문자 세트 또는 데이터 정렬과 관련된 파라미터를 모두 설정합니다. 인스턴스 또는 클러스터 안에 데이터베이스를 생성하기 전에도 이 설정을 수행해야 합니다. 이렇게 하면 기본 데이터베이스와 새 데이터베이스가 지정한 문자 세트 및 데이터 정렬 값을 사용하게 됩니다. 문자 세트 또는 데이터 정렬 파라미터를 변경해도 기존 데이터베이스에는 변경된 파라미터가 적용되지 않습니다.

일부 DB 엔진의 경우 다음과 같이 `ALTER DATABASE` 명령을 사용하여 기존 데이터베이스의 문자 세트 또는 데이터 정렬 값을 변경할 수 있습니다.

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

데이터베이스의 문자 집합 또는 데이터 정렬 값 변경에 대한 자세한 내용은 DB 엔진 설명서를 참조하세요.

지원되는 파라미터 및 파라미터 값

DB 엔진에서 지원되는 파라미터를 확인하려면 DB 인스턴스 또는 DB 클러스터에서 사용되는 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹에서 파라미터를 확인합니다. 자세한 내용은 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 값 보기](#) 및 [Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 값 보기](#)(을)를 참조하세요.

대부분의 경우 표현식, 수식 및 함수를 사용하여 정수 및 부울 파라미터 값을 지정할 수 있습니다. 함수에 수학 로그식을 넣을 수 있습니다. 그러나 일부 파라미터는 파라미터 값에 대한 표현식, 수식 및 함수를 지원하지 않습니다. 자세한 내용은 DB 파라미터 지정 섹션을 참조하세요.

Aurora Global Database의 경우 개별 Aurora 클러스터마다 서로 다른 구성 설정을 지정할 수 있습니다. 보조 클러스터를 기본 클러스터로 승격하는 경우 일관되게 작동할 수 있을 만큼 설정이 충분히 비슷한지 확인해야 합니다. 예를 들면 Aurora Global Database의 모든 클러스터 간에 시간대와 문자 세트에 대해 동일한 설정을 사용합니다.

파라미터 그룹에 파라미터를 잘못 설정하면 성능 저하나 시스템 불안정 등의 의도하지 않은 부작용이 있을 수 있습니다. 데이터베이스 파라미터를 수정할 때 항상 주의하고 DB 파라미터 그룹을 수정하기 전에 데이터를 백업하세요. 파라미터 그룹 변경 내용을 프로덕션 DB 인스턴스나 DB 클러스터에 적용하기 전에 테스트 DB 인스턴스나 DB 클러스터에 적용해 봐야 합니다.

Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹

Amazon Aurora DB 클러스터는 DB 클러스터 파라미터 그룹을 사용합니다. 다음 섹션에서는 DB 클러스터 파라미터 그룹 구성 및 관리에 대해 설명합니다.

주제

Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터

Aurora에서는 두 가지 수준의 구성 설정 시스템을 사용합니다.

- DB 클러스터 파라미터 그룹의 파라미터는 DB 클러스터의 모든 DB 인스턴스에 적용됩니다. 데이터는 Aurora 공유 스토리지 하위 시스템에 저장됩니다. 이로 인해 테이블 데이터의 물리적 레이아웃과

관련된 모든 파라미터는 Aurora 클러스터의 모든 DB 인스턴스에 대해 동일해야 합니다. 이와 마찬가지로 Aurora DB 인스턴스가 복제에 의해 연결되므로 복제 설정에 대한 모든 파라미터는 Aurora 클러스터 전체에 걸쳐 동일해야 합니다.

- DB 파라미터 그룹의 파라미터는 Aurora DB 클러스터의 단일 DB 인스턴스에 적용됩니다. 이 파라미터는 동일한 Aurora 클러스터에 있는 DB 인스턴스 전반에 걸쳐 변경할 수 있는 메모리 사용량과 같은 속성과 관련이 있습니다. 예를 들어 클러스터에는 AWS 인스턴스 클래스가 다양한 DB 인스턴스르 포함하는 경우가 많습니다.

모든 Aurora 클러스터는 DB 클러스터 파라미터 그룹과 연결됩니다. 이 파라미터 그룹은 해당 DB 엔진의 모든 구성 값에 기본값을 할당합니다. 해당 클러스터 파라미터 그룹에는 클러스터 수준 파라미터와 인스턴스 수준 파라미터의 기본값이 모두 들어 있습니다. 프로비저닝된 클러스터 또는 Aurora Serverless v2 클러스터 내의 각 DB 인스턴스는 해당 DB 클러스터 파라미터 그룹의 설정을 상속합니다.

각 DB 인스턴스 역시 하나의 DB 파라미터 그룹에 연결되어 있습니다. DB 파라미터 그룹의 값은 클러스터 파라미터 그룹의 기본값을 재정의합니다. 예를 들어 클러스터의 한 인스턴스에 문제가 발생한 경우 사용자 지정 DB 파라미터 그룹을 해당 인스턴스에 할당할 수 있습니다. 이 사용자 정의 파라미터 그룹에는 디버깅 또는 성능 튜닝과 관련된 파라미터에 대한 특정 설정이 있을 수 있습니다.

Aurora는 지정된 데이터베이스 엔진 및 버전에 따라 클러스터 또는 새 DB 인스턴스를 생성할 때 기본 파라미터 그룹을 할당합니다. 대신 사용자 지정 파라미터 그룹을 지정할 수 있습니다. 이러한 파라미터 그룹을 직접 생성하고 파라미터 값을 편집할 수 있습니다. 생성 시 이러한 사용자 정의 파라미터 그룹을 지정할 수 있습니다. 나중에 DB 클러스터 또는 인스턴스를 수정하여 사용자 지정 파라미터 그룹을 사용할 수도 있습니다.

프로비저닝된 인스턴스 및 Aurora Serverless v2 인스턴스의 경우 사용자가 DB 클러스터 파라미터 그룹에서 수정한 구성 값으로 DB 파라미터 그룹의 기본값을 재정의합니다. DB 파라미터 그룹의 해당 값을 편집하면 그 값으로 DB 클러스터 파라미터 그룹의 설정을 재정의합니다.

고객님이 수정하는 모든 DB 파라미터 설정은 고객님이 구성 파라미터를 다시 기본값으로 변경하더라도 DB 클러스터 파라미터 그룹 값에 우선합니다. describe-db-parameters AWS CLI 명령 또는 DescribeDBParameters RDS API 작업을 사용하여 어떤 파라미터가 재정의되는지 확인할 수 있습니다. 해당 파라미터를 수정한 경우 `Source` 필드에는 `user`라는 값이 포함되어 있습니다. DB 클러스터 파라미터 그룹의 값이 우선하도록 한 개 이상의 파라미터를 재설정하려면 reset-db-parameter-group AWS CLI 명령 또는 ResetDBParameterGroup RDS API 작업을 사용합니다.

Aurora에서 제공되는 DB 클러스터 및 DB 인스턴스 파라미터는 데이터베이스 엔진 호환성에 따라 다릅니다.

데이터베이스 엔진	파라미터
Aurora MySQL	Aurora MySQL 구성 파라미터을(를) 참조하세요. Aurora Serverless 클러스터의 경우 Aurora Serverless v2에 대한 파라미터 그룹 작업 및 Aurora Serverless v1 파라미터 그룹 단원에서 추가 세부 정보를 참조하세요.
Aurora PostgreSQL	Amazon Aurora PostgreSQL parameters을(를) 참조하세요. Aurora Serverless 클러스터의 경우 Aurora Serverless v2에 대한 파라미터 그룹 작업 및 Aurora Serverless v1 파라미터 그룹 단원에서 추가 세부 정보를 참조하세요.

> **Note**
>
> Aurora Serverless v1 클러스터에는 DB 파라미터 그룹이 아닌 DB 클러스터 파라미터 그룹만 있습니다. Aurora Serverless v2 클러스터의 경우 DB 클러스터 파라미터 그룹의 사용자 지정 파라미터에 대한 모든 변경을 수행합니다.
> Aurora Serverless v2는 DB 클러스터 파라미터 그룹과 DB 파라미터 그룹을 모두 사용합니다. Aurora Serverless v2를 사용하면 거의 모든 구성 파라미터를 수정할 수 있습니다. Aurora Serverless v2는 Aurora Serverless v2 인스턴스가 축소될 때 워크로드가 중단되지 않도록 일부 용량 관련 구성 파라미터의 설정을 재정의합니다.
> Aurora Serverless 구성 설정과 수정할 수 있는 설정에 대해 자세히 알아보려면 Aurora Serverless v2에 대한 파라미터 그룹 작업 및 Aurora Serverless v1 파라미터 그룹 단원을 참조하세요.

Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 새 DB 클러스터 파라미터 그룹을 생성할 수 있습니다.

DB 클러스터 파라미터 그룹을 생성한 후 해당 DB 클러스터 파라미터 그룹을 사용하는 DB 클러스터를 생성하려면 5분 이상 기다려야 합니다. 이렇게 하면 새 DB 클러스터에서 사용하기 전에 Amazon RDS에서 파라미터 그룹을 완전히 생성할 수 있습니다. Amazon RDS 콘솔의 파라미터 그룹 페이지 또는

[describe-db-cluster-parameters](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-db-cluster-parameters.html) 명령을 사용하여 DB 클러스터 파라미터 그룹이 생성되었는지 확인할 수 있습니다.

DB 클러스터 파라미터 그룹 이름에는 다음과 같은 제한이 적용됩니다.

- 이름은 1~255자의 문자, 숫자 또는 하이픈으로 구성되어야 합니다.

  기본 파라미터 그룹 이름에는 마침표(예: `default.aurora-mysql5.7`)가 포함될 수 있습니다. 하지만 사용자 지정 파라미터 그룹 이름에는 마침표를 포함할 수 없습니다.
- 첫 번째 자리는 문자여야 합니다.
- 이름은 하이픈으로 끝나거나 2개 연속 하이픈을 포함할 수 없습니다.

콘솔

DB 클러스터 파라미터 그룹을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. [Create parameter group]을 선택합니다.
4. 파라미터 그룹 이름에 새 DB 클러스터 파라미터 그룹의 이름을 입력합니다.
5. 설명에 새 DB 클러스터 파라미터 그룹에 대한 설명을 입력합니다.
6. 엔진 유형에서 데이터베이스 엔진을 선택합니다.
7. 파라미터 그룹 패밀리에서 DB 파라미터 그룹 패밀리를 선택합니다.
8. Create를 선택합니다.

AWS CLI

DB 클러스터 파라미터 그룹을 생성하려면 AWS CLI `create-db-cluster-parameter-group` 명령을 사용합니다.

다음 예에서는 "My new cluster parameter group(내 새로운 클러스터 파라미터 그룹)"이라는 설명과 함께 mydbclusterparametergroup이라는 Aurora MySQL 버전 5.7용 DB 클러스터 파라미터 그룹을 생성합니다.

다음 필수 파라미터를 포함합니다.

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

사용 가능한 모든 파라미터 그룹 패밀리를 나열하려면 다음 명령을 사용합니다.

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
> 출력에 중복이 있습니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

Windows의 경우:

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

다음과 비슷한 출력이 생성됩니다.

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
```

```
    }
}
```

RDS API

DB 클러스터 파라미터 그룹을 생성하려면 RDS API `CreateDBClusterParameterGroup` 작업을 사용합니다.

다음 필수 파라미터를 포함합니다.

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Amazon Aurora에서 DB 클러스터 파라미터 그룹과 DB 클러스터 연결

사용자 지정 설정으로 사용자의 DB 클러스터 파라미터 그룹을 생성할 수 있습니다. AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 파라미터 그룹을 DB 클러스터와 연결할 수 있습니다. DB 클러스터를 생성하거나 수정할 때 이 작업을 수행할 수 있습니다.

DB 클러스터 파라미터 그룹 생성에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 단원을 참조하세요. DB 클러스터 생성에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하세요. DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

> **Note**
>
> Aurora PostgreSQL 15.2, 14.7, 13.10, 12.14 및 모든 11 버전에서, DB 클러스터와 연결된 DB 클러스터 파라미터 그룹을 변경하면 각 복제본 인스턴스를 재부팅하여 변경 사항을 적용합니다.
> 변경 사항을 적용하기 위해 DB 클러스터의 기본 DB 인스턴스를 재부팅해야 하는지 여부를 확인하려면 다음 AWS CLI 명령을 실행합니다.
> `aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> 출력에서 기본 DB 인스턴스의 `DBClusterParameterGroupStatus` 값을 확인합니다. 값이 `pending-reboot`인 경우, DB 클러스터의 기본 DB 인스턴스를 재부팅합니다.

콘솔

DB 클러스터 파라미터 그룹을 DB 클러스터와 연결하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 수정하려는 DB 클러스터를 선택합니다.
3. 수정을 선택합니다. Modify DB cluster(DB 클러스터 수정) 페이지가 나타납니다.
4. DB 클러스터 파라미터 그룹 설정을 변경합니다.
5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.

   변경 사항은 수정 예약 설정에 관계없이 즉시 적용됩니다.
6. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 올바른 경우 클러스터 수정을 선택하여 변경 내용을 저장합니다.

   그렇지 않으면 [Back]을 선택하여 변경 내용을 편집하거나 [Cancel]을 선택하여 변경 내용을 취소합니다.

AWS CLI

DB 클러스터 파라미터 그룹을 DB 클러스터와 연결하려면 다음 옵션과 함께 AWS CLI `modify-db-cluster` 명령을 사용합니다.

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

다음 예제에서는 `mydbclpg` DB 파라미터 그룹을 `mydbcluster` DB 클러스터와 연결합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

DB 클러스터 파라미터 그룹을 DB 클러스터와 연결하려면 RDS API `ModifyDBCluster` 작업을 다음 파라미터와 함께 사용합니다.

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정

고객이 생성한 DB 클러스터 파라미터 그룹에서 파라미터 값을 수정할 수 있습니다. 기본 DB 클러스터 파라미터 그룹에서는 파라미터 값을 변경할 수 없습니다. 고객이 생성한 DB 클러스터 파라미터 그룹의 파라미터를 변경하면 DB 클러스터 파라미터 그룹과 연결된 모든 DB 클러스터에 해당 변경 내용이 적용됩니다.

콘솔

DB 클러스터 파라미터 그룹을 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 수정할 파라미터 그룹을 선택합니다.
4. 파라미터 그룹 작업에서 편집을 선택합니다.
5. 수정하려는 파라미터의 값을 변경합니다. 대화 상자 오른쪽 위의 화살표 키를 사용하여 파라미터를 스크롤할 수 있습니다.

   기본 파라미터 그룹의 값은 변경할 수 없습니다.
6. [Save changes]를 선택합니다.
7. 클러스터의 기본(라이터) DB 인스턴스를 재부팅하여 변경 사항을 적용합니다.
8. 그런 다음 리더 DB 인스턴스를 재부팅하여 변경 사항을 적용합니다.

   DB 인스턴스를 재부팅하지 않으면 장애 조치 작업이 평소보다 오래 걸릴 수 있습니다.

AWS CLI

DB 클러스터 파라미터 그룹을 수정하려면 AWS CLI modify-db-cluster-parameter-group 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-cluster-parameter-group-name`
- `--parameters`

다음 예에서는 mydbclusterparametergroup이라는 DB 클러스터 파라미터 그룹에서 `server_audit_logging` 및 `server_audit_logs_upload` 값을 수정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

다음과 같은 출력이 생성됩니다.

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

DB 클러스터 파라미터 그룹을 수정하려면 RDS API `ModifyDBClusterParameterGroup` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `DBClusterParameterGroupName`
- `Parameters`

Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 재설정

고객이 생성한 DB 클러스터 파라미터 그룹에서 파라미터를 기본값으로 수정할 수 있습니다. 고객이 생성한 DB 클러스터 파라미터 그룹의 파라미터를 변경하면 DB 클러스터 파라미터 그룹과 연결된 모든 DB 클러스터에 해당 변경 내용이 적용됩니다.

> **Note**
>
> 기본 DB 클러스터 파라미터 그룹에서 파라미터는 항상 기본값으로 설정됩니다.

콘솔

DB 클러스터 파라미터 그룹의 파라미터를 기본값으로 수정

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 파라미터 그룹을 선택합니다.
4. 파라미터 그룹 작업에서 편집을 선택합니다.
5. 기본값으로 재설정할 파라미터를 선택합니다. 대화 상자 오른쪽 위의 화살표 키를 사용하여 파라미터를 스크롤할 수 있습니다.

   기본 파라미터 그룹의 값은 변경할 수 없습니다.
6. Reset을 선택한 다음 Reset parameters를 선택하여 확인합니다.
7. DB 클러스터의 기본 DB 인스턴스를 재부팅하여 변경 사항을 DB 클러스터의 모든 DB 인스턴스에 적용합니다.

AWS CLI

DB 클러스터 파라미터 그룹의 파라미터를 기본값으로 수정하려면 AWS CLI `reset-db-cluster-parameter-group` 명령을 필수 옵션 `--db-cluster-parameter-group-name`과 함께 사용합니다.

DB 클러스터 파라미터 그룹의 모든 파라미터를 수정하려면 `--reset-all-parameters` 옵션을 지정합니다. 특정 파라미터를 수정하려면 `--parameters` 옵션을 지정합니다.

다음 예제에서는 mydbparametergroup이라는 DB 파라미터 그룹의 모든 파라미터를 기본값으로 수정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Windows의 경우:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

다음 예제에서는 mydbclusterparametergroup이라는 DB 클러스터 파라미터 그룹에서 `server_audit_logging` 및 `server_audit_logs_upload` 값을 수정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

다음과 같은 출력이 생성됩니다.

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS API

DB 클러스터 파라미터 그룹의 파라미터를 기본값으로 재설정하려면 다음과 같은 필수 파라미터와 함께 RDS API `ResetDBClusterParameterGroup` 명령을 사용합니다: `DBClusterParameterGroupName`.

DB 클러스터 파라미터 그룹의 모든 파라미터를 재설정하려면 `ResetAllParameters` 파라미터를 `true`로 설정합니다. 특정 파라미터를 재설정하려면 `Parameters` 파라미터를 지정합니다.

Amazon Aurora에서 DB 클러스터 파라미터 그룹 복사

생성하는 사용자 지정 DB 클러스터 파라미터 그룹을 복사할 수 있습니다. DB 클러스터 파라미터 그룹을 이미 생성했으며 해당 그룹의 사용자 지정 파라미터와 값의 대부분을 새 DB 클러스터 파라미터 그룹에 포함하려는 경우 파라미터 그룹을 복사하면 편리합니다. AWS CLIcopy-db-cluster-parameter-group 명령이나 RDS API CopyDBClusterParameterGroup 작업을 사용하여 DB 클러스터 파라미터 그룹을 복사할 수 있습니다.

DB 클러스터 파라미터 그룹을 복사한 후 5분 이상 기다렸다가 해당 DB 클러스터 파라미터 그룹을 기본 파라미터 그룹으로 사용하는 첫 번째 DB 클러스터를 복사합니다. 이렇게 하면 새 DB 클러스터에서 사용하기 전에 Amazon RDS에서 파라미터 그룹을 완전히 복사할 수 있습니다. Amazon RDS 콘솔의 파라미터 그룹 페이지 또는 describe-db-cluster-parameters 명령을 사용하여 DB 클러스터 파라미터 그룹이 생성되었는지 확인할 수 있습니다.

> Note
>
> 기본 파라미터 그룹은 복사할 수 없습니다. 하지만 기본 파라미터 그룹을 바탕으로 하는 새로운 파라미터 그룹을 만들 수 있습니다.
> DB 클러스터 파라미터 그룹을 다른 AWS 계정 또는 AWS 리전에 복사할 수 없습니다.

콘솔

DB 클러스터 파라미터 그룹을 복사하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

3. 목록에서 복사할 사용자 지정 파라미터 그룹을 선택합니다.
4. 파라미터 그룹 작업에서 복사를 선택합니다.
5. 새로운 DB 파라미터 그룹 식별자에 새로운 파라미터 그룹의 이름을 입력합니다.
6. 설명에 새로운 파라미터 그룹에 대한 설명을 입력합니다.
7. [Copy]를 선택합니다.

AWS CLI

DB 클러스터 파라미터 그룹을 복사하려면 AWS CLI `copy-db-cluster-parameter-group` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

다음 예에서는 DB 클러스터 파라미터 그룹 `mygroup2`을 복사하여 `mygroup1`라는 새 DB 클러스터 파라미터 그룹을 생성합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

Windows의 경우:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

DB 클러스터 파라미터 그룹을 복사하려면 RDS API `CopyDBClusterParameterGroup` 작업을 다음 필수 파라미터와 함께 사용합니다.

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Amazon Aurora에서 DB 클러스터 파라미터 그룹 나열

AWS 계정에 대해 생성한 DB 클러스터 파라미터 그룹을 나열할 수 있습니다.

> **Note**
> 특정 DB 엔진과 버전에 대한 DB 클러스터를 생성할 때 기존 파라미터 템플릿에서 기본 파라미터 그룹이 자동으로 생성됩니다. 이 기본 파라미터 그룹은 기본 파라미터 설정을 포함하며 수정할 수 없습니다. 사용자 지정 파라미터 그룹을 생성할 때 파라미터 설정을 수정할 수 있습니다.

콘솔

AWS 계정에 대한 모든 DB 클러스터 파라미터 그룹을 나열하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   DB 클러스터 파라미터 그룹은 목록에서 DB 클러스터 파라미터 그룹유형에 나타납니다.

AWS CLI

AWS 계정에 사용할 수 있는 모든 DB 클러스터 파라미터 그룹을 나열하려면 AWS CLI `describe-db-cluster-parameter-groups` 명령을 사용합니다.

Example

다음 예에서는 AWS 계정에 사용할 수 있는 모든 DB 클러스터 파라미터 그룹을 나열합니다.

```
aws rds describe-db-cluster-parameter-groups
```

다음은 mydbclusterparametergroup 파라미터 그룹을 설명하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Windows의 경우:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

다음과 같은 응답이 반환됩니다.

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

RDS API

AWS 계정에 사용할 수 있는 모든 DB 클러스터 파라미터 그룹을 나열하려면 RDS API `DescribeDBClusterParameterGroups` 작업을 사용합니다.

Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 값 보기

DB 클러스터 파라미터 그룹의 모든 파라미터와 해당 값 목록을 가져올 수 있습니다.

콘솔

DB 클러스터 파라미터 그룹의 파라미터 값을 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   DB 클러스터 파라미터 그룹은 목록에서 DB 클러스터 파라미터 그룹유형에 나타납니다.

3. DB 클러스터 파라미터 그룹의 이름을 선택하여 파라미터 목록을 봅니다.

AWS CLI

DB 클러스터 파라미터 그룹의 파라미터 값을 보려면 AWS CLI `describe-db-cluster-parameters` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-cluster-parameter-group-name`

Example

다음 예에서는 mydbclusterparametergroup이라는 DB 클러스터 파라미터 그룹에 대한 파라미터와 파라미터 값을 JSON 형식으로 나열합니다.

다음과 같은 응답이 반환됩니다.

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
```

```
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

RDS API

DB 클러스터 파라미터 그룹의 파라미터 값을 보려면 RDS API `DescribeDBClusterParameters` 명령을 다음 필수 파라미터와 함께 사용하세요.

- `DBClusterParameterGroupName`

경우에 따라 파라미터에 허용된 값이 표시되지 않습니다. 이러한 파라미터는 항상 소스가 데이터베이스 엔진 기본값인 파라미터입니다.

이러한 파라미터의 값을 보려면 다음 SQL 문을 실행하면 됩니다.

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

Amazon Aurora에서 DB 클러스터 파라미터 그룹 삭제

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 파라미터 그룹을 삭제할 수 있습니다. DB 클러스터 파라미터 그룹 파라미터 그룹은 DB 클러스터와 연결되지 않은 경우에만 삭제할 수 있습니다.

콘솔

파라미터 그룹을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   파라미터 그룹이 목록에 나타납니다.
3. 삭제할 DB 클러스터 파라미터 그룹의 이름을 선택합니다.
4. 작업을 선택한 후 삭제를 선택합니다.
5. 파라미터 그룹 이름을 검토한 다음 삭제를 선택합니다.

AWS CLI

DB 클러스터 파라미터 그룹을 삭제하려면 AWS CLI `delete-db-cluster-parameter-group` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-parameter-group-name`

Example

다음 예제에서는 mydbparametergroup이라는 DB 클러스터 파라미터 그룹을 삭제합니다.

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

DB 클러스터 파라미터 그룹을 삭제하려면 RDS API `DeleteDBClusterParameterGroup` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `DBParameterGroupName`

Amazon Aurora DB 인스턴스용 DB 파라미터 그룹

DB 인스턴스는 DB 파라미터 그룹을 사용합니다. 다음 섹션에서는 DB 인스턴스 파라미터 그룹 구성 및 관리에 대해 설명합니다.

주제

Amazon Aurora에서 DB 파라미터 그룹 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 새 DB 파라미터 그룹을 생성할 수 있습니다.

DB 파라미터 그룹 이름에는 다음과 같은 제한이 적용됩니다.

- 이름은 1~255자의 문자, 숫자 또는 하이픈으로 구성되어야 합니다.

  기본 파라미터 그룹 이름에는 마침표(예: `default.mysql8.0`)가 포함될 수 있습니다. 하지만 사용자 지정 파라미터 그룹 이름에는 마침표를 포함할 수 없습니다.
- 첫 번째 자리는 문자여야 합니다.
- 이름은 하이픈으로 끝나거나 2개 연속 하이픈을 포함할 수 없습니다.

콘솔

DB 파라미터 그룹을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

3. [Create parameter group]을 선택합니다.
4. 파라미터 그룹 이름에 새 DB 파라미터 그룹의 이름을 입력합니다.
5. 설명에 새 DB 클러스터 파라미터 그룹에 대한 설명을 입력합니다.
6. 엔진 유형에서 DB 엔진을 선택합니다.
7. 파라미터 그룹 패밀리에서 DB 파라미터 그룹 패밀리를 선택합니다.
8. 유형에서 DB 파라미터 그룹을 선택합니다.
9. 생성(Create)을 선택합니다.

AWS CLI

DB 파라미터 그룹을 생성하려면 AWS CLI`create-db-parameter-group` 명령을 사용합니다. 다음 예에서는 'My new parameter group'이라는 설명과 함께 mydbparametergroup이라는 MySQL 버전 8.0 용 DB 파라미터 그룹을 생성합니다.

다음 필수 파라미터를 포함합니다.

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

사용 가능한 모든 파라미터 그룹 패밀리를 나열하려면 다음 명령을 사용합니다.

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
> 출력에 중복이 있습니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
```

```
    --description "My new parameter group"
```

Windows의 경우:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

다음과 비슷한 출력이 생성됩니다.

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS API

DB 파라미터 그룹을 생성하려면 RDS API `CreateDBParameterGroup` 작업을 사용합니다.

다음 필수 파라미터를 포함합니다.

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Amazon Aurora의 DB 인스턴스에 DB 파라미터 그룹 연결

사용자 지정 설정을 사용하여 사용자의 DB 파라미터 그룹을 생성할 수 있습니다. AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 파라미터 그룹을 DB 인스턴스와 연결할 수 있습니다. DB 인스턴스를 생성하거나 수정할 때 이 작업을 수행할 수 있습니다.

DB 파라미터 그룹 생성에 대한 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹 생성 단원을 참조하세요. DB 인스턴스 수정에 대한 자세한 내용은 DB 클러스터에서 DB 인스턴스 수정 단원을 참조하세요.

> **Note**
> 새 DB 파라미터 그룹을 DB 인스턴스와 연결하면 수정된 정적 파라미터 및 동적 파라미터는 DB 인스턴스가 재부팅된 후에만 적용됩니다. 그러나 DB 파라미터 그룹을 DB 인스턴스에 연결한 후 DB 파라미터 그룹에서 동적 파라미터를 수정하면 이러한 변경 사항이 재부팅 없이 즉시 적용됩니다.

콘솔

DB 파라미터 그룹을 DB 인스턴스와 연결하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 변경하려는 DB 인스턴스를 선택합니다.
3. 수정을 선택합니다. Modify DB instance(DB 인스턴스 수정) 페이지가 나타납니다.
4. DB 파라미터 그룹 설정을 변경합니다.
5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
6. (선택 사항) 즉시 적용을 선택하여 변경 내용을 즉시 적용합니다. 일부의 경우 이 옵션을 선택하면 중단이 발생할 수 있습니다.
7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 정확할 경우 DB 인스턴스 수정을 선택하여 변경 내용을 저장합니다.

   또는 뒤로를 선택하여 변경 내용을 편집하거나 취소를 선택하여 변경 내용을 취소합니다.

AWS CLI

DB 파라미터 그룹을 DB 인스턴스와 연결하려면 다음 옵션과 함께 AWS CLI `modify-db-instance` 명령을 사용합니다.

- `--db-instance-identifier`
- `--db-parameter-group-name`

다음 예제에서는 `mydbpg` DB 파라미터 그룹을 `database-1` DB 인스턴스와 연결합니다. `--apply-immediately`를 사용하면 변경 내용이 즉시 적용됩니다. `--no-apply-immediately`를 사용하여 다음 유지 관리 기간 동안 변경 사항을 적용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

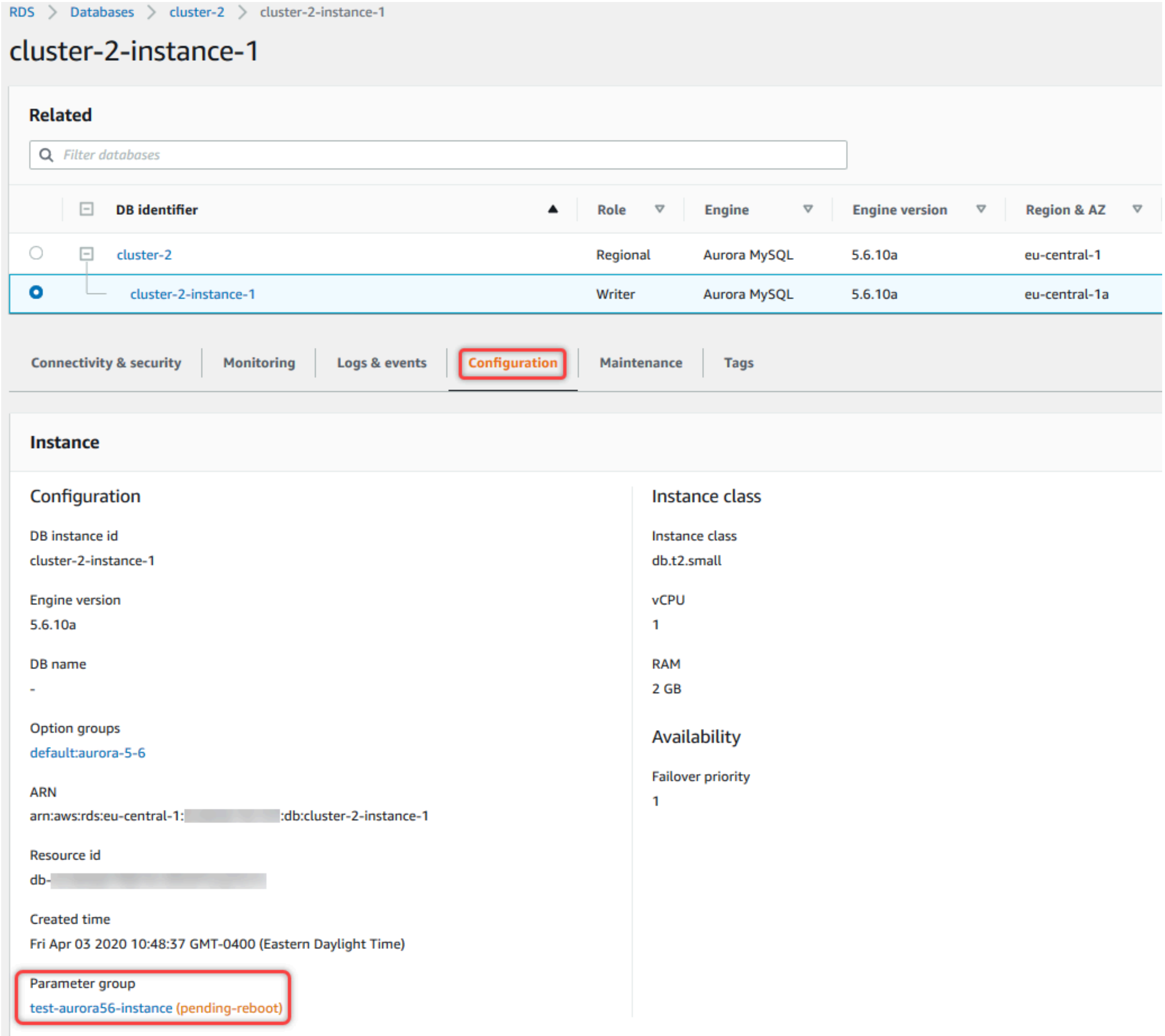
DB 파라미터 그룹을 DB 인스턴스와 연결하려면 RDS API `ModifyDBInstance` 작업을 다음 파라미터와 함께 사용합니다.

- `DBInstanceName`
- `DBParameterGroupName`

Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정

고객이 생성한 DB 파라미터 그룹의 파라미터 값은 수정할 수 있지만, 기본 DB 파라미터 그룹의 파라미터 값은 변경할 수 없습니다. 고객이 생성한 DB 파라미터 그룹의 파라미터를 변경하면 DB 파라미터 그룹과 연결된 모든 DB 인스턴스에 해당 변경 내용이 적용됩니다.

일부 파라미터에 대한 변경 사항은 재부팅 없이 DB 인스턴스에 즉시 적용됩니다. 다른 파라미터에 대한 변경 내용은 DB 인스턴스를 재부팅한 후에만 적용됩니다. RDS 콘솔에는 구성 탭에서 DB 인스턴스와 연결된 DB 파라미터 그룹의 상태가 표시됩니다. 예를 들어 DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않다고 가정해 봅시다. 이 경우 RDS 콘솔에 DB 파라미터 그룹이 pending-reboot(재시작 보류 중) 상태로 표시됩니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다. 파라미터를 수정한 후에는 항상 DB 인스턴스를 재부팅하는 것이 좋습니다. DB 인스턴스를 재부팅하지 않으면 장애 조치 작업이 평소보다 오래 걸릴 수 있습니다.



콘솔

DB 파라미터 그룹의 파라미터를 수정하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 수정할 파라미터 그룹의 이름을 선택합니다.
4. 파라미터 그룹 작업에서 편집을 선택합니다.

5. 수정할 파라미터의 값을 변경합니다. 대화 상자 오른쪽 위의 화살표 키를 사용하여 파라미터를 스크롤할 수 있습니다.

   기본 파라미터 그룹의 값은 변경할 수 없습니다.
6. 변경 사항 저장을 선택합니다.

AWS CLI

DB 파라미터 그룹을 수정하려면 AWS CLI`modify-db-parameter-group` 명령을 다음 필수 옵션과 함께 사용합니다.

- `--db-parameter-group-name`
- `--parameters`

다음 예에서는 mydbparametergroup이라는 DB 파라미터 그룹에서 `max_connections` 및 `max_allowed_packet` 값을 수정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

다음과 같은 출력이 생성됩니다.

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS API

DB 파라미터 그룹을 수정하려면 RDS API `ModifyDBParameterGroup` 작업을 다음 필수 파라미터와 함께 사용합니다.

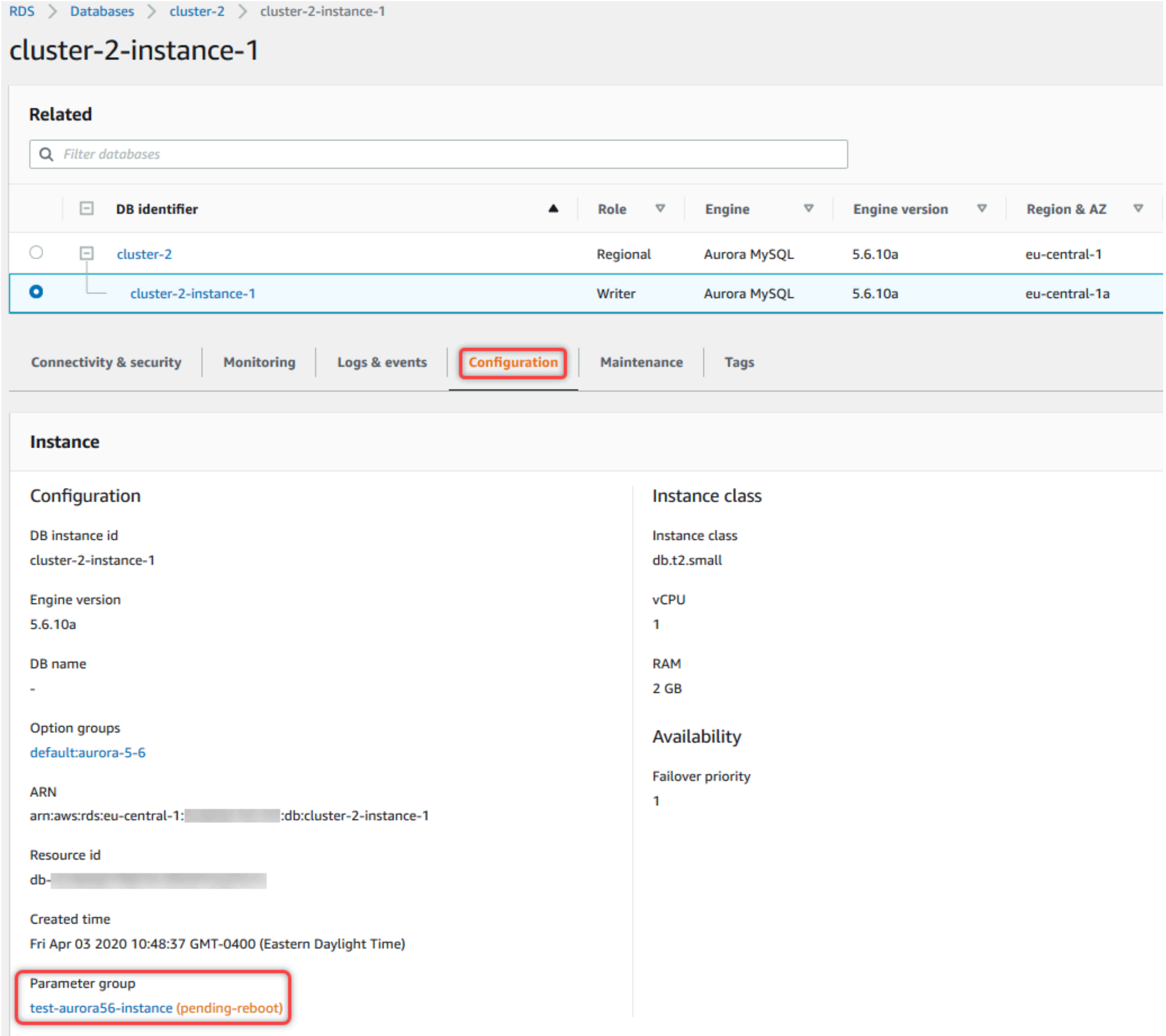
- `DBParameterGroupName`
- `Parameters`

Amazon Aurora에서 DB 파라미터 그룹의 파라미터를 기본값으로 재설정

고객이 생성한 DB 파라미터 그룹의 파라미터 값을 기본값으로 재설정할 수 있습니다. 고객이 생성한 DB 파라미터 그룹의 파라미터를 변경하면 DB 파라미터 그룹과 연결된 모든 DB 인스턴스에 해당 변경 내용이 적용됩니다.

콘솔을 사용하는 경우 특정 파라미터를 기본값으로 재설정할 수 있습니다. 하지만 DB 파라미터 그룹의 모든 파라미터를 한꺼번에 손쉽게 재설정할 수는 없습니다. AWS CLI 또는 RDS API를 사용하는 경우 특정 파라미터를 기본값으로 재설정할 수 있습니다. DB 파라미터 그룹의 모든 파라미터를 한꺼번에 재설정할 수도 있습니다.

일부 파라미터에 대한 변경 사항은 재부팅 없이 DB 인스턴스에 즉시 적용됩니다. 다른 파라미터에 대한 변경 내용은 DB 인스턴스를 재부팅한 후에만 적용됩니다. RDS 콘솔에는 구성 탭에서 DB 인스턴스와 연결된 DB 파라미터 그룹의 상태가 표시됩니다. 예를 들어 DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않다고 가정해 봅시다. 이 경우 RDS 콘솔에 DB 파라미터 그룹이 pending-reboot(재시작 보류 중) 상태로 표시됩니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다.



> **Note**
>
> 기본 DB 파라미터 그룹에서 파라미터는 항상 기본값으로 설정됩니다.

콘솔

DB 파라미터 그룹의 파라미터를 기본값으로 재설정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 파라미터 그룹을 선택합니다.
4. 파라미터 그룹 작업에서 편집을 선택합니다.
5. 기본값으로 재설정할 파라미터를 선택합니다. 대화 상자 오른쪽 위의 화살표 키를 사용하여 파라미터를 스크롤할 수 있습니다.

   기본 파라미터 그룹의 값은 변경할 수 없습니다.
6. Reset을 선택한 다음 Reset parameters를 선택하여 확인합니다.

AWS CLI

DB 파라미터 그룹의 파라미터를 일부 또는 모두 재설정하려면 필수 옵션 AWS CLI과 함께 `reset-db-parameter-group``--db-parameter-group-name` 명령을 사용합니다.

DB 파라미터 그룹의 모든 파라미터를 수정하려면 `--reset-all-parameters` 옵션을 지정합니다. 특정 파라미터를 수정하려면 `--parameters` 옵션을 지정합니다.

다음 예제에서는 mydbparametergroup이라는 DB 파라미터 그룹의 모든 파라미터를 기본값으로 수정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Windows의 경우:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

다음 예제에서는 mydbparametergroup이라는 DB 파라미터 그룹에서 `max_connections` 및 `max_allowed_packet` 옵션을 재설정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

다음과 같은 출력이 생성됩니다.

```
DBParameterGroupName  mydbparametergroup
```

RDS API

DB 파라미터 그룹의 파라미터를 기본값으로 재설정하려면 필수 파라미터 `ResetDBParameterGroup`과 함께 RDS API `DBParameterGroupName` 명령을 사용합니다.

DB 파라미터 그룹의 모든 파라미터를 재설정하려면 `ResetAllParameters` 파라미터를 `true`로 설정합니다. 특정 파라미터를 재설정하려면 `Parameters` 파라미터를 지정합니다.

Amazon Aurora에서 DB 파라미터 그룹 복사

생성하는 사용자 지정 DB 파라미터 그룹을 복사할 수 있습니다 파라미터 그룹을 복사하는 것이 편리한 솔루션이 될 수 있습니다. DB 파라미터 그룹을 만들고 이 그룹의 사용자 지정 파라미터 및 값의 대부분을 새 DB 파라미터 그룹에 포함하려는 경우를 예로 들 수 있습니다. AWS Management Console을 사용하여 DB 파라미터 그룹을 복사할 수 있습니다. AWS CLI copy-db-parameter-group 명령 또는 RDS API CopyDBParameterGroup 작업을 사용할 수도 있습니다.

DB 파라미터 그룹을 복사한 후 5분 이상 기다렸다가 해당 DB 파라미터 그룹을 기본 파라미터 그룹으로 사용하는 첫 번째 DB 인스턴스를 생성하십시오. 이렇게 하면 파라미터 그룹이 사용되기 전에

Amazon RDS에서 복사 작업을 완전히 마칠 수 있습니다. 이는 DB 인스턴스의 기본 데이터베이스를 생성할 때 필수적인 파라미터에 특히 중요합니다. 한 가지 예는 `character_set_database` 파라미터로 정의되는 기본 데이터베이스에 대한 문자 집합입니다. [Amazon RDS 콘솔](https://console.aws.amazon.com/rds/)의 파라미터 그룹 옵션이나 [describe-db-parameters](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-db-parameters.html) 명령을 사용하여 DB 파라미터 그룹이 생성되었는지 확인하십시오.

> **Note**
>
> 기본 파라미터 그룹은 복사할 수 없습니다. 하지만 기본 파라미터 그룹을 바탕으로 하는 새로운 파라미터 그룹을 만들 수 있습니다.
> DB 파라미터 그룹을 다른 AWS 계정 또는 AWS 리전에 복사할 수 없습니다.

콘솔

DB 파라미터 그룹을 복사하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 복사할 사용자 지정 파라미터 그룹을 선택합니다.
4. 파라미터 그룹 작업에서 복사를 선택합니다.
5. 새로운 DB 파라미터 그룹 식별자에 새로운 파라미터 그룹의 이름을 입력합니다.
6. 설명에 새로운 파라미터 그룹에 대한 설명을 입력합니다.
7. [Copy]를 선택합니다.

AWS CLI

DB 파라미터 그룹을 복사하려면 AWS CLI `copy-db-parameter-group` 명령을 다음 필수 옵션과 함께 사용합니다.

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

다음 예에서는 DB 파라미터 그룹 `mygroup2`을 복사하여 `mygroup1`라는 새 DB 파라미터 그룹을 생성합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

Windows의 경우:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

DB 파라미터 그룹을 복사하려면 RDS API `CopyDBParameterGroup` 작업을 다음 필수 파라미터와 함께 사용합니다.

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

Amazon Aurora에 DB 파라미터 그룹 나열

AWS 계정에 대해 생성한 DB 파라미터 그룹을 나열할 수 있습니다.

> **Note**
>
> 특정 DB 엔진과 버전에 대한 DB 인스턴스를 생성할 때 기존 파라미터 템플릿에서 기본 파라미터 그룹이 자동으로 생성됩니다. 이 기본 파라미터 그룹은 기본 파라미터 설정을 포함하며 수정할 수 없습니다. 사용자 지정 파라미터 그룹을 생성할 때 파라미터 설정을 수정할 수 있습니다.

콘솔

AWS 계정에 대한 모든 DB 파라미터 그룹을 나열하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   DB 파라미터 그룹이 목록에 나타납니다.

AWS CLI

AWS 계정에 사용할 수 있는 모든 DB 파라미터 그룹을 나열하려면 AWS CLI`describe-db-parameter-groups` 명령을 사용합니다.

Example

다음 예에서는 AWS 계정에 사용할 수 있는 모든 DB 파라미터 그룹을 나열합니다.

```
aws rds describe-db-parameter-groups
```

다음과 같은 응답이 반환됩니다.

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

다음은 mydbparamgroup1 파라미터 그룹을 설명하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

Windows의 경우:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

다음과 같은 응답이 반환됩니다.

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

AWS 계정에 사용할 수 있는 모든 DB 파라미터 그룹을 나열하려면 RDS API DescribeDBParameterGroups 작업을 사용합니다.

Amazon Aurora에서 DB 파라미터 그룹의 파라미터 값 보기

DB 파라미터 그룹의 모든 파라미터와 해당 값 목록을 가져올 수 있습니다.

콘솔

DB 파라미터 그룹의 파라미터 값을 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   DB 파라미터 그룹이 목록에 나타납니다.
3. 파라미터 그룹의 이름을 선택하여 파라미터 목록을 봅니다.

AWS CLI

DB 파라미터 그룹의 파라미터 값을 보려면 AWS CLIdescribe-db-parameters 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-parameter-group-name`

Example

다음 예에서는 mydbparametergroup이라는 DB 파라미터 그룹에 대한 파라미터와 파라미터 값을 나열합니다.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

다음과 같은 응답이 반환됩니다.

```
DBPARAMETER  Parameter Name             Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
```

```
DBPARAMETER   allow-suspicious-udfs                          engine-default   boolean
 static       false
DBPARAMETER   auto_increment_increment                       engine-default   integer
 dynamic      true
DBPARAMETER   auto_increment_offset                          engine-default   integer
 dynamic      true
DBPARAMETER   binlog_cache_size          32768               system           integer
 dynamic      true
DBPARAMETER   socket                     /tmp/mysql.sock     system           string
 static       false
```

RDS API

DB 파라미터 그룹의 파라미터 값을 보려면 RDS API `DescribeDBParameters` 명령을 다음 필수 파라미터와 함께 사용하세요.

- `DBParameterGroupName`

Amazon Aurora에서 DB 파라미터 그룹 삭제

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 파라미터 그룹을 삭제할 수 있습니다. 파라미터 그룹은 DB 인스턴스와 연결되지 않은 경우에만 삭제할 수 있습니다.

콘솔

DB 파라미터 그룹을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   DB 파라미터 그룹이 목록에 나타납니다.
3. 삭제할 파라미터 그룹의 이름을 선택합니다.
4. 작업을 선택한 후 삭제를 선택합니다.
5. 파라미터 그룹 이름을 검토한 다음 삭제를 선택합니다.

AWS CLI

DB 파라미터 그룹을 삭제하려면 AWS CLI `delete-db-parameter-group` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-parameter-group-name`

Example

다음 예제에서는 mydbparametergroup이라는 DB 파라미터 그룹을 삭제합니다.

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

DB 파라미터 그룹을 삭제하려면 RDS API [DeleteDBParameterGroup](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_DeleteDBParameterGroup.html) 명령을 다음 필수 파라미터와 함께 사용합니다.

- `DBParameterGroupName`

DB 파라미터 그룹 비교

AWS Management Console을 사용하여 두 DB 파라미터 그룹 간 차이를 확인할 수 있습니다.

지정된 파라미터 그룹은 둘 다 DB 파라미터 그룹이거나 둘 다 DB 클러스터 파라미터 그룹이어야 합니다. DB 엔진과 버전이 같더라도 마찬가지입니다. 예를 들어 `aurora-mysql8.0`(Aurora MySQL 버전 3) DB 파라미터 그룹과 `aurora-mysql8.0` DB 클러스터 파라미터 그룹을 비교할 수 없습니다.

버전이 다르더라도 Aurora MySQL 파라미터 그룹과 RDS for MySQL DB 파라미터 그룹을 비교할 수 있지만 Aurora PostgreSQL DB 파라미터 그룹과 RDS for PostgreSQL DB 파라미터 그룹을 비교할 수는 없습니다.

두 DB 파라미터 그룹을 비교하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 비교하려는 두 파라미터 그룹을 선택합니다.

> **Note**
>
> 기본 파라미터 그룹을 사용자 지정 파라미터 그룹과 비교하려면 먼저 기본 탭에서 기본 파라미터 그룹을 선택한 다음 사용자 지정 탭에서 사용자 지정 파라미터 그룹을 선택합니다.

4. 작업 에서 비교를 선택합니다.

DB 파라미터 지정

DB 파라미터 유형은 다음과 같습니다.

- Integer
- 불
- String
- Long
- Double
- Timestamp
- 다른 정의된 데이터 형식의 객체
- 정수, 부울, 문자열, long, double, 타임스탬프 또는 객체 형식의 값 배열

표현식, 수식 및 함수를 사용하여 정수 및 부울 파라미터를 지정할 수도 있습니다.

목차

DB 파라미터 수식

DB 파라미터 수식은 정수 값 또는 부울 값으로 확인되는 표현식입니다. 이 표현식은 중괄호({})로 묶여 있습니다. DB 파라미터 값에 수식을 사용하거나 DB 파라미터 함수의 인수로 수식을 사용할 수 있습니다.

구문

```
{FormulaVariable}
```

```
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

DB 파라미터 수식 변수

각 수식 변수는 정수 또는 부울 값을 반환합니다. 변수 이름은 대소문자를 구분합니다.

AllocatedStorage

데이터 볼륨의 크기(바이트)를 나타내는 정수를 반환합니다.

DBInstanceClassMemory

데이터베이스 프로세스에 사용할 수 있는 메모리의 바이트 수에 대한 정수를 반환합니다. 이 숫자는 DB 인스턴스 클래스에 대한 총 메모리 양에서 시작하여 내부적으로 계산됩니다. 이 계산에서 인스턴스를 관리하는 RDS 프로세스와 운영 체제용으로 예약된 메모리가 차감됩니다. 따라서 이 숫자는 Amazon Aurora DB 인스턴스 클래스의 인스턴스 클래스 표에 표시된 메모리 수치보다 항상 다소 낮습니다. 정확한 값은 여러 요인의 조합에 따라 달라집니다. 여기에는 인스턴스 클래스, DB 엔진이 포함되며 RDS 인스턴스에 적용되는지 아니면 Aurora 클러스터의 일부인 인스턴스에 적용되는지에 따라 값이 달라집니다.

DBInstanceVCPU

Amazon RDS가 인스턴스를 관리하는 데 사용하는 vCPU(가상 중앙 처리 유닛)의 수를 나타내는 정수를 반환합니다.

EndPointPort

DB 인스턴스에 연결하는 데 사용되는 포트를 나타내는 정수를 반환합니다.

TrueIfReplica

DB 인스턴스가 읽기 전용 복제본이면 `1`, 아니면 `0`을 반환합니다. Aurora MySQL의 `read_only` 파라미터에 대한 기본값입니다.

DB 파라미터 수식 연산자

DB 파라미터 수식은 나눗셈과 곱셈의 두 가지 연산자를 지원합니다.

나눗셈 연산자: /

나뉨수를 나눗수로 나누어 정수 몫을 반환합니다. 몫의 소수는 잘리며 반올림되지 않습니다.

구문

```
dividend / divisor
```

나뉨수 및 나눗수 인수는 정수 식이어야 합니다.

곱셈 연산자: *

표현식을 곱하여 해당 표현식의 곱을 반환합니다. 표현식 소수는 잘리며 반올림되지 않습니다.

구문

```
expression * expression
```

두 식 모두 정수여야 합니다.

DB 파라미터 함수

DB 파라미터 함수의 인수를 정수 또는 수식으로 지정합니다. 함수마다 인수가 하나 이상 있어야 합니다. 여러 인수를 쉼표로 구분된 목록으로 지정합니다. 목록에 argument1, argument3과 같은 빈 멤버를 넣을 수 없습니다. 함수 이름은 대소문자를 구분하지 않습니다.

IF

인수를 반환합니다.

구문

```
IF(argument1, argument2, argument3)
```

첫 번째 인수가 true이면 두 번째 인수를 반환합니다. 그렇지 않으면 세 번째 인수를 반환합니다.

GREATEST

정수 또는 파라미터 수식 목록에서 가장 큰 값을 반환합니다.

구문

```
GREATEST(argument1, argument2,...argumentn)
```

정수를 반환합니다.

LEAST

정수 또는 파라미터 수식 목록에서 가장 작은 값을 반환합니다.

구문

```
LEAST(argument1, argument2,...argumentn)
```

정수를 반환합니다.

SUM

지정된 정수나 파라미터 수식의 값을 더합니다.

구문

```
SUM(argument1, argument2,...argumentn)
```

정수를 반환합니다.

DB 파라미터 로그 표현식

정수 DB 파라미터 값을 로그 표현식으로 설정할 수 있습니다. 이 표현식은 중괄호({})로 묶여 있습니다. 예:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

`log` 함수는 로그 밑 2를 나타냅니다. 또한 이 예제에서는 `DBInstanceClassMemory` 수식 변수를 사용합니다. [DB 파라미터 수식 변수](#) 단원을 참조하십시오.

DB 파라미터 값 예제

이 예에서는 DB 파라미터 값에 수식, 함수 및 표현식을 사용하는 방법을 보여 줍니다.

> **Warning**
> DB 파라미터 그룹에 파라미터를 잘못 설정하면 의도하지 않은 부작용이 있을 수 있습니다. 이러한 부작용에는 성능 저하, 시스템 불안정 등이 포함될 수 있습니다. 데이터베이스 파라미터를 수정할 때 주의하고 DB 파라미터 그룹을 수정하기 전에 데이터를 백업하세요. 파라미터 그

를 변경 내용을 프로덕션 DB 인스턴스에 적용하기 전에 특정 시점으로 복원을 사용하여 생성한 테스트 DB 인스턴스에 적용해 봐야 합니다.

Example DB 파라미터 함수 LEAST 사용

MySQL Aurora MySQL LEAST 파라미터 값에 `table_definition_cache` 함수를 지정할 수 있습니다. 정의 캐시에 저장될 수 있는 테이블 정의 수를 `DBInstanceClassMemory`/393040 또는 20,000 중 더 작은 값으로 설정하려면 이 함수를 사용합니다.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Amazon Aurora DB 클러스터로 데이터 마이그레이션

데이터베이스 엔진 호환성에 따라 기존 데이터베이스의 데이터를 Amazon Aurora DB 클러스터로 마이그레이션하기 위한 여러 가지 옵션이 있습니다. 마이그레이션할 데이터베이스와 데이터 크기에 따라서도 마이그레이션 옵션이 달라집니다.

Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

다음 원본 중 하나의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션할 수 있습니다.

- RDS for MySQL DB 인스턴스
- Amazon RDS 외부의 MySQL 데이터베이스
- MySQL과 호환되지 않는 데이터베이스

자세한 내용은 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 단원을 참조하십시오.

Amazon Aurora PostgreSQL DB 클러스터로 데이터 마이그레이션

다음 원본 중 하나의 데이터를 Amazon Aurora PostgreSQL DB 클러스터로 마이그레이션할 수 있습니다.

- Amazon RDS PostgreSQL DB 인스턴스
- PostgreSQL과 호환되지 않는 데이터베이스

자세한 내용은 데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션 단원을 참조하십시오.

Aurora DB 클러스터 설정을 사용하여 Amazon ElastiCache 캐시 생성

ElastiCache는 유연한 실시간 사용 사례를 지원하는 마이크로초 단위의 읽기 및 쓰기 지연 시간을 제공하는 완전 관리형 인 메모리 캐싱 서비스입니다. ElastiCache는 애플리케이션 및 데이터베이스 성능을 가속화하는 데 도움이 될 수 있습니다. ElastiCache를 게임 리더보드, 스트리밍, 데이터 분석 같은 데이터 내구성이 필요하지 않은 사용 사례의 기본 데이터 스토어로 사용할 수 있습니다. ElastiCache는 분산된 컴퓨팅 환경의 배포 및 관리와 관련된 복잡성을 해소하는 데 도움을 줍니다. 자세한 내용은 일반적인 ElastiCache 사용 사례 및 ElastiCache가 도움이 되는 방법(Memcached) 및 일반적인 ElastiCache 사용 사례 및 ElastiCache가 도움이 되는 방법(Redis OSS)을 참조하세요. Amazon RDS 콘솔을 사용하여 ElastiCache 캐시를 생성할 수 있습니다.

Amazon ElastiCache는 2가지 형식으로 운영할 수 있습니다. 서버리스 캐시로 시작하거나 자체 캐시 클러스터를 설계하도록 선택할 수 있습니다. 자체 캐시 클러스터를 설계하기로 선택한 경우 ElastiCache는 Redis OSS 엔진 및 Memcached 엔진 모두와 함께 작동합니다. 어떤 엔진을 사용해야 할지 잘 모르겠는 경우 Memcached와 Redis OSS 비교를 참조하세요. Amazon ElastiCache에 대한 자세한 내용은 Amazon ElastiCache 사용 설명서를 참조하세요.

주제

Aurora DB 클러스터 설정을 사용하는 ElastiCache 캐시 생성 개요

새로 생성한 또는 기존의 Aurora DB 클러스터와 동일한 구성 설정을 사용하여 Amazon RDS에서 ElastiCache 캐시를 생성할 수 있습니다.

ElastiCache 캐시를 DB 클러스터와 연결하는 몇 가지 사용 사례:

- RDS에서만 실행하는 대신 ElastiCache를 RDS와 함께 사용하면 비용을 절감하고 성능을 개선할 수 있습니다.
- 데이터 내구성이 필요하지 않은 애플리케이션의 프라이머리 데이터 스토어로 ElastiCache 캐시를 사용할 수 있습니다. Redis OSS 또는 Memcached를 사용하는 애플리케이션은 거의 수정하지 않고 ElastiCache를 사용할 수 있습니다.

RDS에서 ElastiCache 캐시를 생성하는 경우, ElastiCache 캐시는 연결된 Aurora DB 클러스터에서 다음 설정을 상속합니다.

- ElastiCache 연결 설정
- ElastiCache 보안 설정

요구 사항에 따라 캐시 구성 설정을 설정할 수 있습니다.

애플리케이션에 ElastiCache 설정

ElastiCache 캐시를 활용하도록 애플리케이션을 설정해야 합니다. 또한 요구 사항에 따라 캐싱 전략을 사용하도록 애플리케이션을 설정하여 캐시 성능을 최적화하고 개선할 수 있습니다.

- ElastiCache 캐시에 액세스하여 시작하려면 ElastiCache 시작하기(Redis OSS) 및 ElastiCache 시작하기(Memcached)를 참조하세요.
- 캐싱 전략에 대한 자세한 내용은 캐싱 전략 및 모범 사례(Memcached) 및 캐싱 전략 및 모범 사례(Redis OSS)를 참조하세요.
- ElastiCache(Redis OSS) 클러스터의 고가용성에 대한 자세한 내용은 복제 그룹을 사용한 고가용성을 참조하세요.
- 백업 스토리지, 리전 내 또는 리전 간 데이터 전송 또는 AWS Outposts 사용과 관련된 비용이 발생할 수 있습니다 요금에 대한 자세한 내용은 Amazon ElastiCache 요금을 참조하세요.

Aurora DB 클러스터의 설정을 사용하여 ElastiCache 캐시 생성

DB 클러스터에서 상속되는 설정을 사용하여 Aurora DB 클러스터용 ElastiCache 캐시를 생성할 수 있습니다.

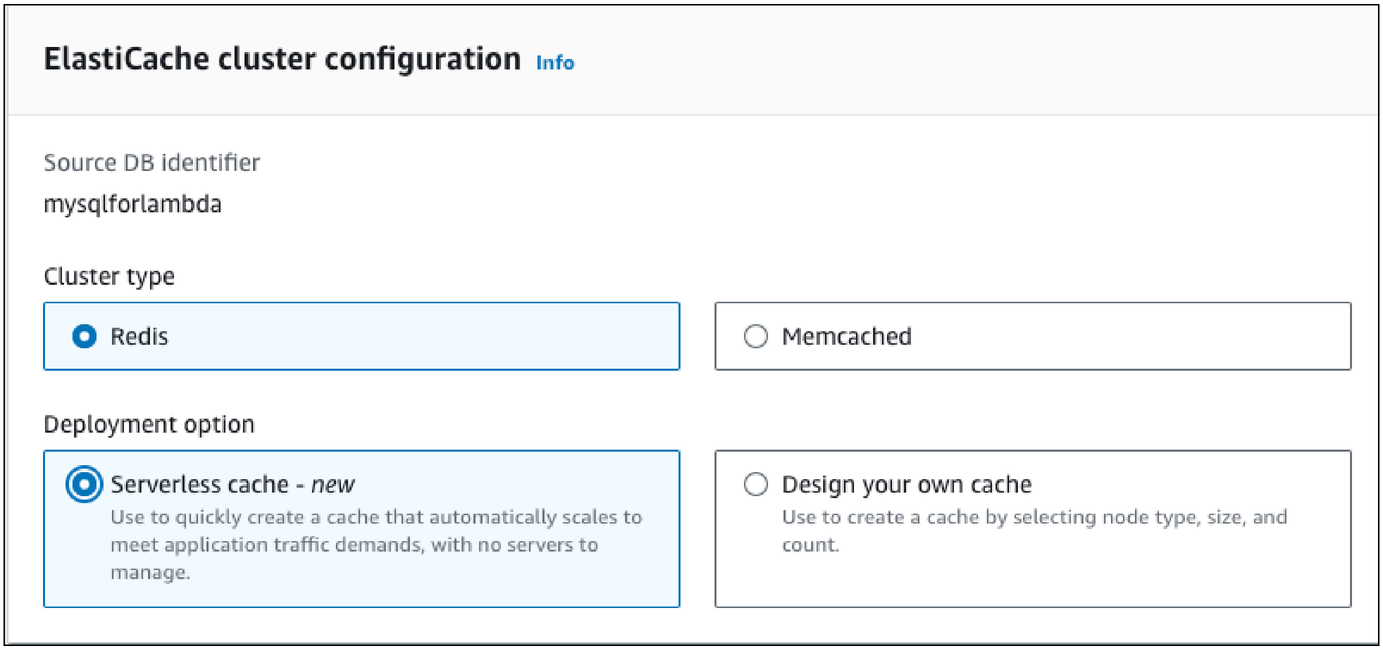
DB 클러스터의 설정을 사용하여 ElastiCache 캐시 생성

1. DB 클러스터를 만들려면 Amazon Aurora DB 클러스터 생성 단원의 지침을 따르십시오.
2. Aurora DB 클러스터를 생성하면 콘솔에 제안된 추가 기능 창이 표시됩니다. Create an ElastiCache cluster from RDS using your DB settings(DB 설정을 사용하여 RDS에서 ElastiCache 클러스터 생성)를 선택합니다.

   기존 데이터베이스의 경우 데이터베이스 페이지에서 필요한 DB 클러스터를 선택합니다. 작업 드롭다운 메뉴에서 ElastiCache 클러스터 생성을 선택하여 기존 Aurora DB 클러스터와 동일한 설정을 가진 RDS에 ElastiCache 캐시를 생성합니다.

ElastiCache 구성 섹션의 소스 DB 식별자에 ElastiCache 캐시가 설정을 상속받는 DB 클러스터가 표시됩니다.

3. Redis OSS 클러스터를 생성할지 또는 Memcached 클러스터를 생성할지 선택하세요. 자세한 내용은 Memcached와 Redis OSS 비교를 참조하세요.



4. 그런 다음 서버리스 캐시를 만들지 아니면 자체 캐시를 설계할지 선택합니다. 자세한 내용은 배포 옵션 간 선택을 참조하세요.

   서버리스 캐시를 선택하는 경우:

   a. 캐시 설정의 이름 및 설명에 값을 입력합니다.
   b. 기본 설정 보기에서 기본 설정을 그대로 두고 캐시와 DB 클러스터 간의 연결을 설정합니다.
   c. 기본 설정 사용자 지정을 선택하여 기본 설정을 편집할 수도 있습니다. ElastiCache 연결 설정, ElastiCache 보안 설정, 최대 사용량 제한을 선택합니다.

5. 자체 캐시 설계를 선택하는 경우:

   a. Redis OSS 클러스터를 선택한 경우 클러스터 모드를 활성화 또는 비활성화 상태로 유지할지 선택합니다. 자세한 내용은 복제: Redis OSS(클러스터 모드 비활성화됨)와 Redis OSS(클러스터 모드 활성화됨) 비교를 참조하세요.
   b. 이름과 설명, 엔진 버전을 입력합니다.

      엔진 버전의 경우 권장 기본값은 최신 엔진 버전입니다. 요구 사항에 가장 적합한 ElastiCache 캐시용 엔진 버전을 선택할 수도 있습니다.

c. 노드 유형 옵션에서 노드 유형을 선택합니다. 자세한 내용은 노드 관리를 참조하세요.

   클러스터 모드를 활성화됨으로 설정한 상태에서 Redis OSS 클러스터를 생성하려고 선택한 경우, 샤드 수 옵션에 샤드 수(파티션/노드 그룹 수)를 입력하세요.

   복제본 개수에 각 샤드의 복제본 수를 입력합니다.

   > **Note**
   >
   > 선택한 노드 유형, 샤드 수, 복제본 수는 모두 캐시 성능 및 리소스 비용에 영향을 미칩니다. 이러한 설정이 데이터베이스 요구 사항과 일치하는지 확인하세요. 요금에 대한 자세한 정보는 Amazon ElastiCache 요금을 참조하세요.

d. ElastiCache 연결 설정 및 ElastiCache 보안 설정을 선택합니다. 기본 설정을 유지하거나 요구 사항에 따라 이러한 설정을 사용자 지정할 수 있습니다.

6. ElastiCache 캐시의 기본 설정 및 상속된 설정을 확인합니다. 일부 설정은 생성 후에 변경할 수 없습니다.

   > **Note**
   >
   > RDS는 최소 기간 요구 사항인 60분을 충족하도록 ElastiCache 캐시의 백업 기간을 조정할 수 있습니다. 소스 데이터베이스의 백업 기간은 동일하게 유지됩니다.

7. 준비가 되면 ElastiCache 캐시 생성을 선택합니다.

콘솔에는 ElastiCache 캐시 생성을 위한 확인 배너가 표시됩니다. 배너의 링크를 따라 ElastiCache 콘솔로 이동하면 캐시 세부 정보를 볼 수 있습니다. ElastiCache 콘솔에는 새로 생성된 ElastiCache 캐시가 표시됩니다.

AWS Database Migration Service를 사용하여 EC2 데이터베이스를 Amazon Aurora로 자동 마이그레이션

Aurora 콘솔을 사용하여 EC2 데이터베이스를 Aurora로 마이그레이션할 수 있습니다. Aurora는 AWS Database Migration Service(AWS DMS)를 사용하여 소스 EC2 데이터베이스를 마이그레이션합니다. AWS DMS를 사용하면 관계형 데이터베이스를 AWS 클라우드로 마이그레이션할 수 있습니다. AWS Database Migration Service에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서의 AWS Database Migration Service이란 무엇인가요?를 참조하세요.

마이그레이션을 시작하려면 데이터를 마이그레이션하기 위해 상응하는 Aurora DB 클러스터를 만들어야 합니다. 대상 데이터베이스를 만든 후 EC2 데이터베이스를 가져올 수 있습니다. 1TiB보다 작은 소스 데이터베이스의 경우 이 마이그레이션 작업은 데이터를 Aurora로 마이그레이션하는 데 필요한 시간과 리소스를 줄입니다.

개요

Aurora 콘솔을 사용하면 EC2 데이터베이스를 상응하는 Aurora 데이터베이스로 마이그레이션할 수 있습니다. 콘솔에서 마이그레이션을 활성화하려면 Aurora 데이터베이스를 만들어야 합니다.

다음 데이터베이스 엔진의 EC2 데이터베이스를 마이그레이션할 수 있습니다.

- MySQL
- PostgreSQL

마이그레이션 프로세스는 다음과 같은 단계로 구성됩니다.

- Aurora에서 상응하는 데이터베이스를 만듭니다. 데이터베이스가 상응하려면 데이터베이스 엔진과 호환되는 엔진 버전이 동일해야 합니다. 또한 동일한 VPC에 있어야 합니다. 데이터베이스 만들기에 대한 지침은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.
- 데이터베이스의 복제 유형을 선택합니다.
  - 전체 로드 마이그레이션 - Aurora가 전체 소스 데이터베이스를 대상 데이터베이스에 복사하며 필요한 경우 대상에 새 테이블을 만듭니다.

> **Note**
>
> 이 옵션을 사용하면 Aurora 데이터베이스가 중단됩니다.

  - 전체 로드 및 변경 데이터 캡처(CDC) 마이그레이션 - 전체 로드 마이그레이션과 마찬가지로 이 옵션을 사용하면 Aurora가 전체 소스 데이터베이스를 대상으로 복사합니다. 그러나 전체 로드 마이그레이션 후 Aurora는 소스에서 캡처된 변경 사항을 대상 데이터베이스에 적용합니다. 변경 데이터 캡처는 데이터베이스 엔진의 기본 API를 사용하여 데이터베이스 로그에 대한 변경 사항을 수집합니다.

    > **Note**
    >
    > 이 옵션을 사용하면 Aurora 데이터베이스가 중단됩니다.

  - 변경 데이터 캡처(CDC) - 마이그레이션 과정 내내 대상 데이터베이스를 계속 사용할 수 있도록 하려면 이 옵션을 사용합니다. Aurora는 소스 데이터베이스의 진행 중인 변경 사항을 대상 데이터베이스로 마이그레이션합니다.
- Aurora는 마이그레이션을 용이하게 하는 데 필요한 네트워킹 리소스를 만듭니다. Aurora가 필요한 리소스를 만들면 만들어진 리소스에 대해 사용자에게 알리고, 사용자는 데이터 전송을 시작할 수 있습니다.

  마이그레이션을 완료하는 데 필요한 시간은 복제 유형과 소스 데이터베이스 크기에 따라 다릅니다.

사전 조건

MySQL

소스 데이터베이스로 MySQL 데이터베이스를 사용하기 전에 다음 사전 조건을 충족하는지 확인해야 합니다. 이러한 사전 요구 사항은 AWS 관리형 소스에 적용됩니다.

Replication Admin 역할이 할당된 AWS DMS용 계정이 있어야 합니다. 역할에는 다음 권한이 있어야 합니다.

- REPLICATION CLIENT - 이 권한은 변경 데이터 캡처(CDC) 작업에만 필요합니다. 즉, 전체 로드 전용 작업에서는 이 권한이 필요하지 않습니다.
- REPLICATION SLAVE - 이 권한은 변경 데이터 캡처(CDC) 작업에만 필요합니다. 즉, 전체 로드 전용 작업에서는 이 권한이 필요하지 않습니다.

또한, AWS DMS 사용자에게는 복제를 위해 지정된 원본 테이블에 대해 SELECT 권한이 있어야 합니다.

MySQL별 사전 마이그레이션 평가를 사용하는 경우 다음 권한을 부여합니다.

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

PostgreSQL

AWS 관리형 PostgreSQL 소스 데이터베이스에서 데이터를 마이그레이션하기 전에 다음을 수행합니다.

- PostgreSQL DB 인스턴스에 필요한 최소 권한이 있는 AWS 사용자 계정을 AWS DMS의 PostgreSQL 소스 엔드포인트 사용자 계정으로 사용하는 것이 좋습니다. 마스터 계정을 사용하는 것은 권장되지 않습니다. 계정에는 `rds_superuser` 역할과 `rds_replication` 역할이 있어야 합니다. `rds_replication` 역할은 논리적 슬롯을 관리하고 논리적 슬롯을 사용하여 데이터를 스트리밍할 수 있는 권한을 부여합니다.

> **Note**
>
> 일부 AWS DMS 트랜잭션은 DMS 엔진에서 다시 사용하기 전까지 얼마 동안 유휴 상태입니다. PostgreSQL 버전 9.6 이상에서 `idle_in_transaction_session_timeout` 파라미터를 사용하면 유휴 트랜잭션이 시간 초과되어 실패하도록 할 수 있습니다.

제한 사항

자동 마이그레이션 프로세스에는 다음 제한 사항이 적용됩니다.

- 소스 데이터베이스 마이그레이션을 시작하려면 대상 데이터베이스 상태가 사용 가능이어야 합니다.
- MySQL 소스 데이터베이스에서 마이그레이션할 때 Aurora 계정에 복제 관리자 역할이 있어야 합니다. 또한 해당 역할에 적절한 권한이 적용되어 있어야 합니다.
- EC2 인스턴스와 대상 데이터베이스는 동일한 VPC에 있어야 합니다.
- EC2 데이터베이스에서 데이터 마이그레이션 작업을 사용할 때는 EC2 데이터베이스를 다음 대상 데이터베이스로 마이그레이션할 수 없습니다.
    - Aurora global database
    - Aurora Limitless database

- Aurora Serverless v1
- MySQL 버전이 5.7 미만인 데이터베이스
- PostgreSQL 버전이 10.4 미만인 데이터베이스

동종 마이그레이션을 위한 IAM 리소스 만들기

Aurora는 AWS DMS를 사용하여 데이터를 마이그레이션합니다. 데이터베이스에 액세스하고 데이터를 마이그레이션하기 위해 AWS DMS는 동종 데이터 마이그레이션을 위한 서버리스 환경을 생성합니다. 이 환경에서 AWS DMS는 VPC 피어링, 라우팅 테이블, 보안 그룹 및 기타 AWS 리소스에 대한 액세스가 필요합니다. 또한 AWS DMS는 Amazon CloudWatch에 각 데이터 마이그레이션에 대한 로그, 지표 및 진행 상황을 저장합니다. 데이터 마이그레이션 프로젝트를 생성하려면 AWS DMS는 이러한 서비스에 대한 액세스 권한이 필요합니다.

또한 AWS DMS에서는 소스 및 대상 연결에 대한 데이터베이스 연결을 인증하기 위해 사용자 자격 증명 세트를 나타내는 보안 암호에 대한 액세스 권한이 필요합니다.

> **Note**
> EC2 인스턴스에서 데이터 마이그레이션 작업을 사용하면 Aurora 콘솔을 사용하여 이러한 IAM 리소스를 생성할 수 있습니다. 콘솔에서 생성한 IAM 리소스를 사용하는 경우 이 단계를 건너뜁니다.

이 프로세스에는 다음과 같은 IAM 리소스가 필요합니다.

주제

동종 데이터 마이그레이션을 위한 IAM 정책 만들기

이 단계에서는 Amazon EC2 및 CloudWatch 리소스에 대한 액세스 권한을 AWS DMS에 부여하는 IAM 정책을 생성합니다. 그런 다음, IAM 역할을 생성하고 이 정책을 연결합니다.

동종 데이터 마이그레이션을 위한 IAM 정책을 만드는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택합니다.
4. 정책 생성 페이지에서 JSON 탭을 선택합니다.
5. 다음 JSON을 편집기에 붙여넣습니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
```

```
                "Effect": "Allow",
                "Action": [
                    "ec2:AuthorizeSecurityGroupEgress",
                    "ec2:AuthorizeSecurityGroupIngress",
                    "ec2:RevokeSecurityGroupEgress",
                    "ec2:RevokeSecurityGroupIngress"
                ],
                "Resource": "arn:aws:ec2:*:*:security-group/*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "ec2:AcceptVpcPeeringConnection",
                    "ec2:ModifyVpcPeeringConnectionOptions"
                ],
                "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
            },
            {
                "Effect": "Allow",
                "Action": "ec2:AcceptVpcPeeringConnection",
                "Resource": "arn:aws:ec2:*:*:vpc/*"
            }
        ]
}
```

6. 다음: 태그와 다음: 검토를 선택합니다.
7. 이름*의 경우, **HomogeneousDataMigrationsPolicy**를 입력한 후 정책 생성을 선택합니다.

동종 데이터 마이그레이션을 위한 IAM 역할 만들기

이 단계에서는 AWS Secrets Manager, Amazon EC2 및 CloudWatch에 대한 액세스 권한을 제공하는 IAM 역할을 만듭니다.

데이터 마이그레이션을 위한 IAM 역할을 만드는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지의 신뢰할 수 있는 엔터티 유형 아래에서 AWS 서비스를 선택합니다. 기타 AWS 서비스 사용 사례에서 DMS를 선택합니다.

5. DMS 확인란을 선택하고 다음을 선택합니다.
6. 권한 추가 페이지에서 이전에 만든 HomogeneousDataMigrationsPolicy를 선택합니다. 다음을 선택합니다.
7. 이름 지정, 검토 및 생성 페이지에서 역할 이름에 **HomogeneousDataMigrationsRole**을 입력하고 역할 생성을 선택합니다.
8. 역할 페이지에서 역할 이름으로 **HomogeneousDataMigrationsRole**을 입력합니다. HomogeneousDataMigrationsRole을 선택합니다.
9. HomogeneousDataMigrationsRole 페이지에서 신뢰 관계 탭을 선택합니다. 신뢰 정책 편집을 선택합니다.
10. 신뢰 정책 편집 페이지에서 기존 텍스트를 대체하여 다음 JSON을 편집기에 붙여넣습니다.

    JSON

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "",
                "Effect": "Allow",
                "Principal": {
                    "Service": [
                        "dms-data-migrations.amazonaws.com",
                        "dms.your_region.amazonaws.com"
                    ]
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }
    ```

    이전 예제에서 *your_region*을 AWS 리전의 이름으로 바꾸십시오.

    위의 리소스 기반 정책은 고객 관리형 HomogeneousDataMigrationsPolicy 정책에 따라 작업을 수행할 수 있는 권한을 AWS DMS 서비스 위탁자에게 부여합니다.
11. 정책 업데이트를 선택합니다.

보안 암호 액세스 정책 및 역할 만들기

아래 프로시저에 따라 DMS가 소스 및 대상 데이터베이스의 사용자 자격 증명에 액세스할 수 있도록 허용하는 보안 암호 액세스 정책 및 역할을 만듭니다.

Amazon RDS가 적절한 보안 암호에 액세스하기 위해 AWS Secrets Manager에 액세스하도록 허용하는 보안 암호 액세스 정책 및 역할을 만드는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 AWS Identity and Access Management(IAM) 콘솔을 엽니다.
2. 정책을 선택한 후 정책 생성을 선택합니다.
3. JSON을 선택하고 다음 정책을 입력하여 보안 암호에 대한 액세스 및 암호 해독을 활성화합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": secret_arn,
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": kms_key_arn,
        }
     ]
}
```

여기서 *secret_arn*은 보안 암호의 ARN(상황에 따라 `SecretsManagerSecretId`에서 가져올 수 있음)이고, *kms_key_arn*은 다음 예시와 같이 보안 암호를 암호화하는 데 사용하는 AWS KMS 키의 ARN입니다.

JSON

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> AWS Secrets Manager에서 생성한 기본 암호화 키를 사용하는 경우, *kms_key_arn*에 대한 AWS KMS 권한을 지정하지 않아도 됩니다.
> 정책에서 두 보안 암호에 대한 액세스를 제공하도록 하려면 다른 *secret_arn*에 대해 추가 JSON 리소스 객체를 지정하면 됩니다.

4. 기억하기 쉬운 이름과 설명(선택 사항)이 있는 정책을 검토하고 생성합니다.
5. 역할을 선택한 다음, 역할 생성을 선택합니다.
6. 신뢰할 수 있는 엔터티 유형으로서 AWS 서비스를 선택합니다.
7. 서비스 목록에서 DMS를 신뢰할 수 있는 서비스로 선택하고 다음: 권한을 선택합니다.
8. 4단계에서 생성한 정책을 찾아 연결한 다음, 계속 진행하여 태그를 추가하고 역할을 검토합니다. 이때 해당 역할의 신뢰 관계를 편집하여 Amazon RDS 리전 서비스 위탁자를 신뢰할 수 있는 엔터티로 사용합니다. 이 보안 주체의 형식은 다음과 같습니다.

```
dms.region-name.amazonaws.com
```

여기서 *region-name*은 리전의 이름입니다(예: `us-east-1`). 따라서 이 리전의 Amazon RDS 리전 서비스 위탁자는 다음과 같습니다.

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

AWS DMS에 대한 IAM 역할을 만들어 Amazon VPC 관리

리소스의 VPC 설정을 관리하려면 AWS DMS에 대한 IAM 역할을 만들어야 합니다. 마이그레이션에 성공하려면 이 역할을 사용할 수 있어야 합니다.

데이터베이스 마이그레이션을 위해 **dms-vpc-role** 만들기

<result>

이렇게 하면 DMS가 마이그레이션을 위한 VPC 설정을 관리하는 데 사용할 역할이 만들어집니다.
</result>

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 콘솔의 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 선택 옵션에서 AWS 서비스 옵션을 선택합니다.

   사용 사례에서 DMS를 선택합니다.
4. 권한 추가 단계에서 `AmazonDMSVPCManagementRole`을 선택하고 다음을 선택합니다.
5. 이름 지정, 검토 및 생성 페이지에서 역할 이름을 `dms-vpc-role`로 설정하고 역할 생성을 선택합니다.

EC2 데이터베이스에 대한 데이터 마이그레이션 설정

EC2 소스 데이터베이스에서 데이터 마이그레이션을 시작하려면 상승하는 Aurora 데이터베이스를 만들어야 합니다. 데이터베이스 만들기에 대한 지침은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.

대상 데이터베이스를 만든 후 다음 단계에 따라 데이터 마이그레이션을 설정합니다.

데이터 마이그레이션 프로젝트 설정

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.

2. 작업 드롭다운을 선택하고 EC2 데이터베이스에서 데이터 마이그레이션 옵션을 선택합니다. 지원되는 대상 데이터베이스는 [제한 사항](#) 섹션을 참조하세요.
3. 소스 EC2 데이터베이스 선택 섹션에서 다음을 수행합니다.
   1. 엔진 유형을 확인하고 소스 데이터베이스와 동일한지 확인합니다.

      또한 엔진 버전이 호환되는지 확인합니다.
   2. EC2 인스턴스에서 소스 데이터베이스가 있는 EC2 인스턴스를 선택합니다.
   3. 포트에 소스 데이터베이스가 트래픽을 허용하는 포트를 입력합니다.
   4. 기존 보안 암호가 없는 경우 보안 암호에서 새 보안 암호를 만들고 사용하기를 선택합니다. 소스 데이터베이스의 사용자 이름과 암호를 입력합니다. 또한 보안 암호를 암호화할 KMS 키를 선택합니다.

      기존 보안 암호가 있는 경우 기존 보안 암호 사용을 선택한 후 드롭다운 메뉴에서 보안 암호를 선택합니다.
   5. 보안 암호의 IAM 역할의 경우 기존 IAM 역할이 있으면 기존 IAM 역할 사용을 선택하고 드롭다운에서 이전 단계의 보안 암호 ID에 액세스할 수 있는 IAM 역할을 선택합니다.

      기존 IAM 역할이 없는 경우 새 IAM 역할을 만들고 사용하기를 선택합니다. IAM 역할 이름에 새 역할의 이름을 입력합니다. 아래 링크에서 이 역할과 연결된 권한을 볼 수 있습니다.
4. 대상 RDS 데이터베이스 보기 섹션에서 다음을 수행합니다.
   1. 섹션 상단에서 대상 데이터베이스의 설정을 확인합니다.
   2. 대상 데이터베이스 자격 증명을 보유한 기존 보안 암호가 없는 경우 보안 암호에서 새 보안 암호를 만들고 사용하기를 선택합니다.

      기존 보안 암호가 있는 경우 드롭다운에서 보안 암호를 선택합니다.
   3. 보안 암호의 IAM 역할의 경우 이전 단계에서 보안 암호에 액세스할 수 있는 IAM 역할을 선택합니다. 기존 IAM 역할이 없는 경우 새 IAM 역할을 만들 수도 있습니다.

      드롭다운에 IAM 역할이 채워지지 않으면 IAM 역할 ARN을 `arn:aws:iam:`*`account_id`*`:role/`*`roleName`* 형식으로 지정합니다.
5. 데이터 마이그레이션 구성 섹션에서 다음을 수행합니다.
   1. 전체 로드, 전체 로드 및 변경 데이터 캡처(CDC) 또는 변경 데이터 캡처(CDC) 중에서 선택하여 데이터 마이그레이션 유형을 선택합니다. 이러한 옵션에 대한 자세한 내용은 [개요](#) 섹션을 참조하세요.

마이그레이션이 시작된 후에는 마이그레이션 유형을 수정할 수 없습니다.

2. 데이터 마이그레이션의 IAM 역할의 경우 기존 IAM 역할이 있는 경우 기존 IAM 역할 사용을 선택하고 드롭다운에서 마이그레이션에 필요한 리소스를 만들 수 있는 권한을 DMS에 부여하는 IAM 역할을 선택합니다. 기존 IAM 역할이 없는 경우 새 IAM 역할을 만들고 사용하기를 선택합니다.

6. 마이그레이션 설정 보기 탭에 데이터 마이그레이션을 성공적으로 설정하는 데 필요한 설정이 표시되는지 확인합니다.
7. 마이그레이션을 선택하여 마이그레이션 설정을 완료합니다.

이 단계를 완료한 후 콘솔의 진행률 배너에서 세부 정보 보기를 선택하여 데이터 마이그레이션을 위해 설정되고 있는 리소스를 확인할 수 있습니다. 필요한 리소스가 설정되면 마이그레이션이 자동으로 시작됩니다. 만드는 경우

여러 데이터베이스를 대상 데이터베이스로 마이그레이션하려면 새 EC2 데이터베이스에 대한 세부 정보를 사용하여 이 프로세스를 다시 시작합니다.

데이터 마이그레이션 관리

RDS 콘솔에서 EC2 데이터베이스에서 데이터 마이그레이션 작업을 사용하면 Aurora가 마이그레이션을 자동으로 시작합니다.

AWS DMS 콘솔을 사용하여 마이그레이션 리소스를 만든 경우 마이그레이션 프로세스를 시작할 수 있습니다.

데이터 마이그레이션 시작

다음 단계에 따라 데이터 마이그레이션을 시작합니다.

데이터 마이그레이션 시작

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 데이터 마이그레이션 탭에서 연결된 데이터 마이그레이션에는 사용 가능한 데이터 마이그레이션이 나열됩니다.

   Aurora 콘솔을 사용하여 설정된 마이그레이션은 필요한 리소스가 설정되면 자동으로 시작됩니다.

   DMS 콘솔을 사용하여 설정된 마이그레이션은 준비 완료로 설정됩니다.

이러한 마이그레이션을 시작하려면 작업 드롭다운을 선택하고 시작을 선택합니다.

4. 그러면 EC2 데이터베이스의 데이터 마이그레이션이 시작됩니다.

데이터 마이그레이션 중지

복제 유형이 전체 로드인 데이터 마이그레이션의 경우 마이그레이션을 중지하면 프로세스가 중지되고 재개할 수 없습니다. 중지하고 나면 마이그레이션을 다시 시작해야 합니다.

복제 유형이 변경 데이터 캡처(CDC) 또는 전체 로드 및 CDC로 설정된 마이그레이션의 경우 연속 복제 프로세스를 중지하고 나중에 프로세스를 재개할 수 있습니다.

데이터 마이그레이션 중지

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 데이터 마이그레이션 탭에서 연결된 데이터 마이그레이션에는 진행 중인 데이터 마이그레이션이 나열됩니다.

   마이그레이션을 중지하려면 데이터 마이그레이션을 선택하고 작업 드롭다운에서 중지를 선택합니다.
4. 그러면 EC2 데이터베이스의 데이터 마이그레이션이 중지됩니다.

데이터 마이그레이션 재개

복제 유형이 전체 로드 및 변경 데이터 캡처(CDC) 또는 변경 데이터 캡처(CDC) 마이그레이션인 데이터 마이그레이션의 경우 마지막 중지 지점에서 CDC 프로세스를 재개할 수 있습니다.

데이터 마이그레이션 재개

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 데이터 마이그레이션 탭에서 연결된 데이터 마이그레이션에는 중지된 데이터 마이그레이션이 나열됩니다.

   마이그레이션을 재개하려면 데이터 마이그레이션을 선택하고 작업 드롭다운에서 처리 재개를 선택합니다.

4. 그러면 EC2 데이터베이스의 데이터 마이그레이션이 재개됩니다.

데이터 마이그레이션 삭제

연결된 데이터 마이그레이션을 삭제하려면 다음 지침을 사용합니다.

데이터 마이그레이션 삭제

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 마이그레이션을 삭제하려면 데이터 마이그레이션을 선택하고 작업 드롭다운에서 삭제를 선택합니다.
4. 이렇게 하면 데이터 마이그레이션이 삭제됩니다.

진행 중이던 데이터 마이그레이션을 삭제해도 대상 데이터베이스에 이미 로드된 데이터는 영향을 받지 않습니다.

데이터 마이그레이션 다시 시작

CDC 시작 지점에서 연결된 데이터 마이그레이션을 다시 시작하려면 다음 지침을 사용합니다.

데이터 마이그레이션 다시 시작

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 마이그레이션을 다시 시작하려면 데이터 마이그레이션을 선택하고 작업 드롭다운에서 다시 시작을 선택합니다.
4. 그러면 CDC 시작 지점에서 데이터 마이그레이션이 다시 시작됩니다.

진행 중이던 데이터 마이그레이션을 다시 시작해도 대상 데이터베이스에 이미 로드된 데이터는 영향을 받지 않습니다.

데이터 마이그레이션 모니터링

데이터 마이그레이션을 시작한 후 상태 및 진행 상황을 모니터링할 수 있습니다. 대규모 데이터집합의 데이터 마이그레이션은 완료하는 데 몇 시간이 걸립니다. 데이터 마이그레이션의 안정성, 가용성 및 고성능을 유지하려면 데이터 마이그레이션 진행 상태를 정기적으로 모니터링하십시오.

데이터 마이그레이션의 상태 및 진행 상황을 확인하려면

1. RDS 콘솔의 데이터베이스 페이지에서 대상 데이터베이스를 선택합니다.
2. 데이터베이스 세부 정보 페이지에서 데이터 마이그레이션 탭을 선택합니다.
3. 연결된 데이터 마이그레이션 섹션에는 데이터 마이그레이션이 나열됩니다. 상태 열을 확인합니다.
4. 실행 중인 데이터 마이그레이션의 경우, 마이그레이션 진행 상황 열에 마이그레이션된 데이터의 비율(%)이 표시됩니다.
5. CloudWatch에서 프로세스를 모니터링하려면 CloudWatch 열에 있는 링크를 사용합니다.

마이그레이션 상태

실행하는 각 데이터 마이그레이션마다 Aurora 콘솔에 상태가 표시됩니다. 다음 목록에 상태가 포함되어 있습니다.

- `Ready`: 데이터 마이그레이션을 시작할 준비가 되었습니다.
- `Starting`: Aurora가 데이터 마이그레이션을 위한 서버리스 환경을 만드는 중입니다.
- `Load running`: Aurora가 전체 로드 마이그레이션을 수행하는 중입니다.
- `Load complete, replication ongoing`: Aurora가 전체 로드를 완료했으며 현재 진행 중인 변경 사항을 복제합니다. 이 상태는 전체 로드 및 CDC 유형 마이그레이션에만 적용됩니다.
- `Replication ongoing`: Aurora가 진행 중인 변경 사항을 복제하고 있습니다. 이 상태는 CDC 유형 마이그레이션에만 적용됩니다.
- `Stopping`: Aurora가 데이터 마이그레이션을 중지하고 있습니다. 이 상태는 작업 메뉴에서 데이터 마이그레이션을 중지하도록 선택할 때 적용됩니다.
- `Stopped`: Aurora가 데이터 마이그레이션을 중지했습니다.
- `Failed`: 데이터 마이그레이션이 실패했습니다. 자세한 내용은 로그 파일을 참조하십시오.
- `Restarting`: 데이터 마이그레이션이 CDC 시작점에서 진행 중인 데이터 복제를 다시 시작했습니다.

자습서: 사용자 지정 파라미터 그룹을 사용하여 MySQL DB 클러스터 만들기

이 자습서에서는 사용자 지정 파라미터와 그룹을 사용하여 MySQL DB 클러스터를 만듭니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹.의 내용을 참조하시기 바랍니다.

> ⚠ Important
>
> AWS 계정 생성은 무료입니다. 그러나 이 자습서를 완료하면 사용하는 AWS 리소스에 대한 비용이 발생할 수 있습니다. 자습서가 더 이상 필요하지 않은 경우 자습서를 완료한 후에 이러한 리소스를 삭제할 수 있습니다.

사용자 지정 구성 및 설정이 있는 DB 클러스터를 만들려면 사용자 지정 파라미터 그룹을 사용하면 됩니다. 사용자 지정 파라미터 그룹은 여러 데이터베이스를 사용하고 이에 대한 설정을 균일하게 구성하려는 경우 특히 유용합니다.

이 단계를 완료하면 다음을 학습하게 됩니다.

- Amazon Aurora를 사용하여 사용자 지정 파라미터 그룹이 있는 MySQL DB 클러스터를 만드는 방법입니다.
- MySQL DB 클러스터에 대한 특정 파라미터을 사용하는 방법입니다.

이 자습서를 완료하려면 다음 작업을 수행하십시오.

1. MySQL 파라미터 `default_password_lifetime`를 사용하여 DB 클러스터 파라미터 그룹을 만듭니다.
2. 사용자가 만든 사용자 지정 DB 클러스터 파라미터 그룹 으로 MySQL DB 클러스터를 만듭니다.

주제

사전 조건

이 자습서에서는 AWS 계정 및 관리자 액세스 권한이 있는 사용자가 있어야 합니다. 아직 설정하지 않았다면 다음 섹션에 나와 있는 단계를 완료합니다.

- AWS 계정에 등록
- 관리자 액세스 권한이 있는 사용자 생성

Amazon Aurora DB 클러스터 파라미터 그룹 만들기

이 자습서에서는 콘솔에서 MySQL DB 클러스터에 대해 default_password_lifetime 를 사용하여 사용자 지정 파라미터 그룹을 만드는 방법을 알아봅니다. `default_password_lifetime` 파라미터는 클라이언트 암호가 자동으로 만료되기까지의 일수를 제어합니다. MySQL DB 클러스터에 사용할 수 있는 기타 파라미터에 대한 자세한 내용은 Aurora MySQL 구성 파라미터 섹션을 참조하시기 바랍니다.

파라미터 그룹을 생성하려면

1. Amazon RDS 콘솔을 열고 파라미터 그룹을 선택합니다.
2. 사용자 지정 파라미터 그룹에서 파라미터 그룹 생성을 선택합니다.
3. 파라미터 그룹 세부 정보를 설정합니다.
    1. 파라미터 그룹의 이름을 입력합니다.
    2. 파라미터 그룹에 대한 설명을 입력합니다.
    3. 엔진 유형에서 를 선택합니다. Aurora MySQL.
    4. 파라미터 그룹 패밀리에서 aurora-mysql8.0을 선택합니다.
    5. 유형에서 DB 클러스터 파라미터 그룹을 선택합니다.
4. 생성(Create)을 선택합니다.

새 DB 클러스터 파라미터 그룹이 Amazon RDS 콘솔의 파라미터 그룹 페이지에 나타납니다. 다음 단계에서는 파라미터 값을 수정하여 파라미터 그룹을 사용자 지정하는 방법을 보여줍니다.

사용자 지정 파라미터 그룹에서 파라미터 값 수정

다음 단계에 따라 Amazon Aurora DB 클러스터 파라미터 그룹 만들기에서 만든 파라미터 그룹의 파라미터 값을 수정합니다.

파라미터 그룹에서 파라미터 값을 수정하려면 다음과 같이 합니다.

1. Amazon RDS 콘솔을 열고 파라미터 그룹을 선택합니다.
2. 사용자 지정 파라미터 그룹에서 사용자가 만든 DB 클러스터 파라미터 그룹의 이름을 선택합니다.
3. 편집을 선택합니다.
4. 파라미터 필터링 검색 상자에서 사용자 지정 파라미터(`default_password_lifetime`)를 검색합니다.
5. 파라미터 옆의 확인란을 선택하고 이 암호 수명 파라미터에 대해 설정할 일 수 값을 입력합니다.
6. 변경 사항 저장을 선택합니다.

이제 사용자 지정 파라미터 그룹을 MySQL 8.0 DB 클러스터에 대해 Amazon Aurora와 연결할 수 있습니다.

DB 클러스터 파라미터 그룹을 사용하여 MySQL DB 클러스터 만들기

마지막으로 이전 단계에서 만든 사용자 지정 파라미터 그룹으로 MySQL DB 클러스터를 만듭니다. 다음 단계는 사용자 지정 파라미터 그룹을 사용하여 MySQL DB 클러스터를 만드는 방법을 보여줍니다.

사용자 지정 파라미터 및 새 옵션 그룹을 사용하여 DB 클러스터를 만들려면 다음과 같이 합니다.

1. Amazon RDS 콘솔을 열고 데이터베이스를 선택합니다.
2. 데이터베이스 생성를 선택합니다.
3. 데이터베이스 생성 방식 선택에서 표준 생성을 선택합니다.
4. 엔진 옵션에서 Aurora(MySQL 호환)를 선택합니다.
5. 추가 구성을 선택합니다.
   - 초기 데이터베이스 이름에서 DB 클러스터의 이름을 선택합니다.
   - DB 클러스터 파라미터 그룹 드롭다운에서 이전에 만든 DB 클러스터 파라미터 그룹의 이름을 선택합니다.
6. 이 자습서에서는 다른 DB 설정의 기본 설정을 그대로 두거나 필요에 따라 수정할 수 있습니다.
7. 데이터베이스 생성를 선택합니다.

RDS는 사용자 지정 파라미터 그룹 그룹을 사용하여 새로운 MySQL DB 클러스터를 만듭니다. 이 데이터베이스에 대한 자세한 내용은 Amazon RDS 콘솔의 데이터베이스 페이지를 참조하시기 바랍니다.

이 자습서에서는 사용자 지정 파라미터 그룹을 사용하여 맞춤형 설정으로 MySQL DB 클러스터를 구성했습니다. 새로 만들어진 MySQL DB 클러스터는 `default_password_lifetime` 파라미터로 사용자 암호 수명을 관리합니다. 데이터베이스를 최적화하기 위해 사용자 지정 파라미터 그룹에 추가 설정을 적용하고 옵션을 추가할 수 있습니다.

사용자 지정된 DB 클러스터 만들기를 완료한 후에는 원치 않는 비용이 발생하지 않도록 리소스를 삭제해야 합니다. DB 클러스터를 삭제하려면 Aurora DB 클러스터 및 DB 인스턴스 삭제 섹션의 설명을 따릅니다.

Amazon Aurora DB 클러스터 관리

이번 섹션에서는 Aurora DB 클러스터를 관리하고 유지하는 방법에 대해 설명합니다. Aurora는 복제 토폴로지에서 연결된 데이터베이스 서버의 클러스터와 관련 있습니다. 따라서 Aurora 관리를 위해서는 여러 개의 서버로 변경 사항을 배포하고 모든 Aurora 복제본이 소스 서버를 따라잡는지 확인해야 하는 경우가 종종 있습니다. Aurora은 데이터 증가에 따라 기반 스토리지를 투명하게 확장하기 때문에 Aurora 관리 시 디스크 스토리지를 관리할 필요가 거의 없습니다. 마찬가지로 Aurora은 연속 백업을 자동으로 수행하기 때문에 Aurora 클러스터에서는 백업 수행을 위한 광범위한 계획 또는 다운타임이 필요하지 않습니다.

주제

Amazon Aurora DB 클러스터 중지 및 시작

Aurora DB 클러스터를 중지하고 시작하면 개발 및 테스트 환경 비용을 관리하는 데 도움이 됩니다. 클러스터를 사용할 때마다 모든 DB 인스턴스를 설정 및 해제하는 대신 클러스터의 모든 DB 인스턴스를 일시적으로 중지할 수 있습니다.

주제

Aurora DB 클러스터의 중지 및 시작 개요

Aurora DB 클러스터가 필요하지 않은 기간에는 이 클러스터의 모든 인스턴스를 한번에 중지할 수 있습니다. 사용해야 할 때는 언제든지 클러스터를 다시 시작할 수 있습니다. 시작 및 중지를 사용하면 개발, 테스트 또는 연속 가용성을 필요로 하지 않는 유사한 활동에 사용되는 클러스터의 설정 및 해제 프로세스가 간소화됩니다. 클러스터 내의 인스턴스 수와 관계없이 단일 작업만 관련된 모든 AWS Management Console 절차를 수행할 수 있습니다.

DB 클러스터가 중지되어 있는 동안에는 지정된 보존 기간 내 클러스터 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용만 청구됩니다. DB 인스턴스 시간에 대해서는 요금이 부과되지 않습니다

> **Important**
>
> DB 클러스터는 최대 7일간 중지할 수 있습니다. 7일이 경과한 후 DB 클러스터를 수동으로 시작하지 않으면 DB 클러스터가 자동으로 시작되므로 필요한 유지 관리 업데이트가 지체되지 않습니다.

로드가 적은 Aurora 클러스터의 요금을 최소화하려면 Aurora 복제본을 모두 삭제하는 대신 클러스터를 중지할 수 있습니다. 1~2개 이상의 인스턴스가 있는 클러스터의 경우 AWS CLI 또는 Amazon RDS API를 사용하여 DB 인스턴스를 자주 삭제하고 다시 생성하는 방법만 실용적입니다. 이러한 일련의 작업은 올바른 순서대로 수행하기 어려울 수도 있습니다(예를 들어 기본 인스턴스를 삭제하기 전에 모든 Aurora 복제본을 삭제하여 장애 조치 메커니즘 활성화를 방지).

DB 클러스터를 계속 실행해야 하지만 필요 이상 용량이 크면 시작 및 중지를 사용하지 마십시오. 클러스터의 비용이 너무 많이 들거나 사용량이 많지 않은 경우 하나 이상의 DB 인스턴스를 삭제하거나 모든 DB 인스턴스를 스몰 인스턴스 클래스로 변경하십시오. 개별 Aurora DB 인스턴스는 중지할 수 없습니다.

DB 클러스터를 중지하는 데 걸리는 시간은 DB 인스턴스 클래스, 네트워크 상태, DB 엔진 유형, 데이터베이스 상태 등의 요인에 따라 달라집니다. 프로세스에는 몇 분이 걸릴 수 있습니다. Amazon RDS 서비스가 수행하는 작업은 다음과 같습니다.

- 데이터베이스 엔진 프로세스를 종료합니다.
- RDS 플랫폼 프로세스를 종료합니다.
- 기본 Amazon EC2 인스턴스를 종료합니다.

DB 클러스터를 재시작하는 데 걸리는 시간은 데이터베이스 크기, DB 인스턴스 클래스, 네트워크 상태, DB 엔진 유형 및 클러스터 종료 시 데이터베이스 상태 등의 요인에 따라 달라집니다. 시작 프로세스는 몇 분에서 몇 시간이 걸릴 수 있지만, 일반적으로 몇 분이면 완료됩니다. 가용성 계획을 세울 때는 시작 시간의 변동성을 고려하는 것이 좋습니다.

서비스는 DB 클러스터를 시작하기 위해 다음과 같은 작업을 수행합니다.

- 기본 Amazon EC2 인스턴스를 제공합니다.
- RDS 플랫폼 프로세스를 시작합니다.
- 데이터베이스 엔진 프로세스를 시작합니다.
- DB 인스턴스를 복구합니다(정상 종료 후에도 복구 수행).

Aurora DB 클러스터 중지 및 시작에 대한 제한 사항

다음과 같은 일부 Aurora 클러스터는 중지 및 시작이 불가능합니다.

- Aurora Global Database의 일부인 클러스터는 글로벌 데이터베이스의 유일한 클러스터인 경우에만 중지하고 시작할 수 있습니다.
- 리전 간 읽기 전용 복제본이 있는 클러스터를 중지하고 시작할 수 없습니다.
- 블루/그린 배포의 일부인 클러스터는 중지 및 시작이 불가능합니다.
- Aurora Serverless v1 클러스터를 중지하거나 시작할 수 없습니다. Aurora Serverless v2에서는 클러스터를 중지하거나 시작할 수 없습니다.

Aurora DB 클러스터 중지

Aurora DB 클러스터를 사용하거나 관리를 수행하려면 항상 실행 중인 Aurora DB 클러스터로 실행한 후 클러스터를 중지한 다음 클러스터를 다시 시작하십시오. 클러스터가 중지되어 있는 동안에는 DB 인스턴스 시간이 아니라 지정된 보존 기간 내 클러스터 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용이 청구됩니다.

중지 작업은 장애 조치 메커니즘 활성화를 피하기 위해 Aurora 복제본 인스턴스를 먼저 중지한 후 기본 인스턴스를 중지합니다.

콘솔

Aurora 클러스터를 중지하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 후 클러스터를 선택합니다. 이 페이지에서 중지 작업을 수행하거나 중지하려는 DB 클러스터의 세부 정보 페이지로 이동하세요.
3. Actions(작업)에서 Stop temporarily(일시적으로 중지)를 선택합니다.
4. Stop DB cluster temporarily(DB 클러스터 일시적으로 중지) 창에서 DB 클러스터가 7일 후에 자동으로 다시 시작된다는 확인을 선택합니다.
5. Stop temporarily(일시적으로 중지)를 선택하여 DB 클러스터를 중지하거나 Cancel(취소)을 선택하여 작업을 취소합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스를 중지하려면 다음 파라미터와 함께 stop-db-cluster 명령을 호출하십시오.

- `--db-cluster-identifier` – Aurora 클러스터의 이름입니다.

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

Amazon RDS API를 사용하여 DB 인스턴스를 중지하려면 다음 파라미터와 함께 StopDBCluster 작업을 호출하십시오.

- `DBClusterIdentifier` – Aurora 클러스터의 이름입니다.

Aurora DB 클러스터가 중지된 상태에서 가능한 작업

Aurora 클러스터가 중지되어 있는 동안, 지정된 자동 백업 보존 기간 내에서 원하는 시점으로 특정 시점 복원을 수행할 수 있습니다. 특정 시점으로 복원에 대한 세부 정보는 데이터 복구 단원을 참조하십시오.

클러스터가 중지된 동안 Aurora DB 클러스터 또는 해당 DB 인스턴스의 구성을 수정할 수 없습니다. 또한 클러스터에서 DB 인스턴스를 추가하거나 제거할 수 없으며 연결된 DB 인스턴스가 있는 경우에도 클러스터를 삭제할 수 없습니다. 이러한 관리 작업을 수행하기 전에 클러스터를 시작해야 합니다.

DB 클러스터를 중지하면 DB 클러스터 파라미터 그룹 또는 DB 클러스터 인스턴스의 DB 파라미터 그룹을 제외한 보류 중인 작업이 제거됩니다.

Aurora은 다시 시작한 후 중지된 클러스터에 예약 유지보수를 적용합니다. 7일 후 Aurora은 중지된 클러스터를 자동으로 시작하므로 유지 관리 상태에서 너무 늦어지지 않는다는 점을 기억하십시오.

Aurora는 클러스터가 중지된 동안 기본 데이터를 변경할 수 없기 때문에 백업 자동화도 수행하지 않습니다. Aurora는 클러스터가 중지되는 동안 백업 보존 기간을 연장하지 않습니다.

Aurora DB 클러스터 시작

항상 이미 중지 상태인 Aurora 클러스터로 시작하는 Aurora DB 클러스터를 시작합니다. 클러스터를 시작하면 모든 DB 인스턴스가 다시 사용 가능하게 됩니다. 클러스터는 엔드포인트, 파라미터 그룹 및 VPC 보안 그룹과 같은 구성 설정을 유지합니다.

DB 클러스터를 시작하는 데는 보통 몇 분 정도 걸립니다.

콘솔

Aurora 클러스터를 시작하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 Databases(데이터베이스)를 선택한 후 클러스터를 선택합니다. 이 페이지에서 시작 작업을 수행하거나 시작하려는 DB 클러스터의 세부 정보 페이지로 이동하세요.
3. Actions(작업)에는 Start(시작)을 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터를 시작하려면 다음 파라미터와 함께 [start-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/start-db-cluster.html) 명령을 호출하십시오.

- `--db-cluster-identifier` – Aurora 클러스터의 이름입니다. 이 이름은 클러스터를 생성할 때 선택한 특정 클러스터 식별자이거나 끝에 `-cluster`가 추가된 상태에서 선택한 DB 인스턴스 식별자입니다.

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

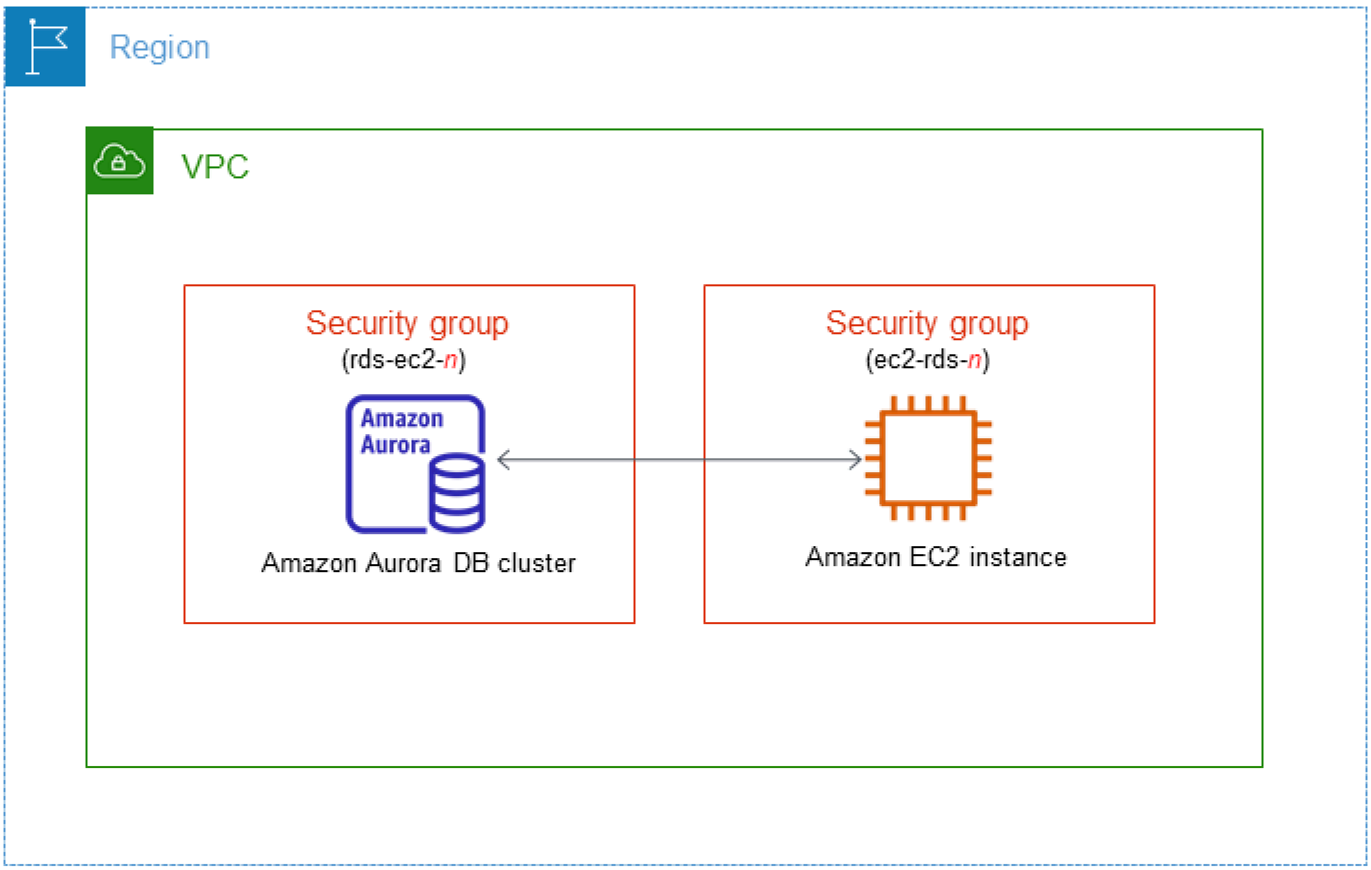
RDS API

Amazon RDS API를 사용하여 Aurora DB 클러스터를 시작하려면 다음 파라미터와 함께 StartDBCluster 작업을 호출하십시오

- `DBCluster` – Aurora 클러스터의 이름입니다. 이 이름은 클러스터를 생성할 때 선택한 특정 클러스터 식별자이거나 끝에 `-cluster`가 추가된 상태에서 선택한 DB 인스턴스 식별자입니다.

EC2 인스턴스와 Aurora DB 클러스터를 자동으로 연결

Amazon RDS 콘솔을 사용하여 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스와 Aurora DB 클러스터 간의 연결 설정을 간소화할 수 있습니다. DB 클러스터는 프라이빗 서브넷에 있고, EC2 인스턴스는 VPC 내의 퍼블릭 서브넷에 있는 경우가 많습니다. EC2 인스턴스의 SQL 클라이언트를 사용하여 DB 클러스터에 연결할 수 있습니다. EC2 인스턴스는 프라이빗 DB 클러스터에 액세스하는 웹 서버 또는 애플리케이션을 실행할 수도 있습니다.



Aurora DB 클러스터와 동일한 VPC에 있지 않은 EC2 인스턴스에 연결하려는 경우 VPC에서 DB 클러스터에 액세스하는 시나리오의 시나리오를 참조하세요.

주제

EC2 인스턴스와의 자동 연결 개요

EC2 인스턴스와 Aurora DB 클러스터 간의 연결을 설정하는 경우 Amazon RDS는 EC2 인스턴스 및 DB 클러스터에 VPC 보안 그룹을 자동으로 구성합니다.

다음은 EC2 인스턴스를 Aurora DB 클러스터와 연결하기 위한 요구 사항입니다.

- EC2 인스턴스는 DB 클러스터와 동일한 VPC 있어야 합니다.

  EC2 인스턴스가 동일한 VPC에 없는 경우 콘솔은 EC2 인스턴스를 생성하기 위한 링크를 제공합니다.
- 현재 DB 클러스터는 Aurora Serverless DB 클러스터일 수 없으며 Aurora Global Database의 일부일 수도 없습니다.
- 연결을 설정하는 사용자는 다음 Amazon EC2 작업을 수행할 수 있는 권한이 있어야 합니다.
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

DB 인스턴스와 EC2 인스턴스가 서로 다른 가용 영역에 있는 경우 계정에 교차 가용 영역 비용이 발생할 가능성이 있습니다.

EC2 인스턴스에 대한 연결을 설정하면 Amazon RDS는 다음 테이블에 설명된 대로 DB 클러스터 및 EC2 인스턴스와 연결된 보안 그룹의 현재 구성을 기반으로 조치를 취합니다.

현재 RDS 보안 그룹 구성	현재 EC2 보안 그룹 구성	RDS 작업
`rds-ec2-`*n*(*n*은 숫자를 나타냄) 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹	`ec2-rds-`*n*(*n*은 숫자를 나타냄) 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그	RDS는 아무 작업도 수행하지 않습니다. EC2 인스턴스와 DB 클러스터 간의 연결이 이미 자동으로 구

현재 RDS 보안 그룹 구성	현재 EC2 보안 그룹 구성	RDS 작업
이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	룹이 수정되지 않았습니다. 이 보안 그룹에는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙이 하나만 포함됩니다.	성되었습니다. EC2 인스턴스와 RDS 데이터베이스 사이에 이미 연결이 존재하기 때문에 보안 그룹은 수정되지 않습니다.
다음 중 하나의 조건이 적용됩니다. • `rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 없습니다. • `rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 EC2 인스턴스와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다. 수정 사항의 예로는 규칙 추가 또는 기존 규칙의 포트 변경이 있습니다.	다음 중 하나의 조건이 적용됩니다. • `ec2-rds-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 없습니다. • `ec2-rds-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	RDS action: create new security groups

현재 RDS 보안 그룹 구성	현재 EC2 보안 그룹 구성	RDS 작업
`rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	`ec2-rds-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	[RDS action: create new security groups]()
`rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	연결에 유효한 EC2 보안 그룹이 있지만 EC2 인스턴스와 연결되어 있지 않습니다. 이 보안 그룹의 이름이 `ec2-rds-n` 패턴과 일치합니다. 수정되지 않았습니다. 여기에는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙이 하나만 포함됩니다.	[RDS action: associate EC2 security group]()

현재 RDS 보안 그룹 구성	현재 EC2 보안 그룹 구성	RDS 작업
다음 중 하나의 조건이 적용됩니다. • `rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 없습니다. • `rds-ec2-n` 패턴과 일치하는 이름을 가진 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 EC2 인스턴스와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	`ec2-rds-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 DB 클러스터의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙이 하나만 포함됩니다.	[RDS action: create new security groups](#)

RDS 작업: 새 보안 그룹 생성

Amazon RDS에서 다음 작업을 수행합니다.

- `rds-ec2-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나 포함됩니다. 이 보안 그룹은 DB 클러스터와 연결되어 있으며 EC2 인스턴스가 DB 클러스터에 액세스하도록 허용합니다.
- `ec2-rds-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 DB 클러스터의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙이 하나 포함됩니다. 이 보안 그룹은 EC2 인스턴스와 연결되어 있으며 EC2 인스턴스가 DB 클러스터에 트래픽을 보내도록 허용합니다.

RDS 작업: EC2 보안 그룹 연결

Amazon RDS는 유효한 기존 EC2 보안 그룹을 EC2 인스턴스와 연결합니다. 이 보안 그룹은 EC2 인스턴스가 DB 클러스터에 트래픽을 보내도록 허용합니다.

EC2 인스턴스와 Aurora DB 클러스터 자동 연결

EC2 인스턴스와 Aurora DB 클러스터 간의 연결을 설정하기 전에 EC2 인스턴스와의 자동 연결 개요에 설명된 요구 사항을 충족하는지 확인하세요.

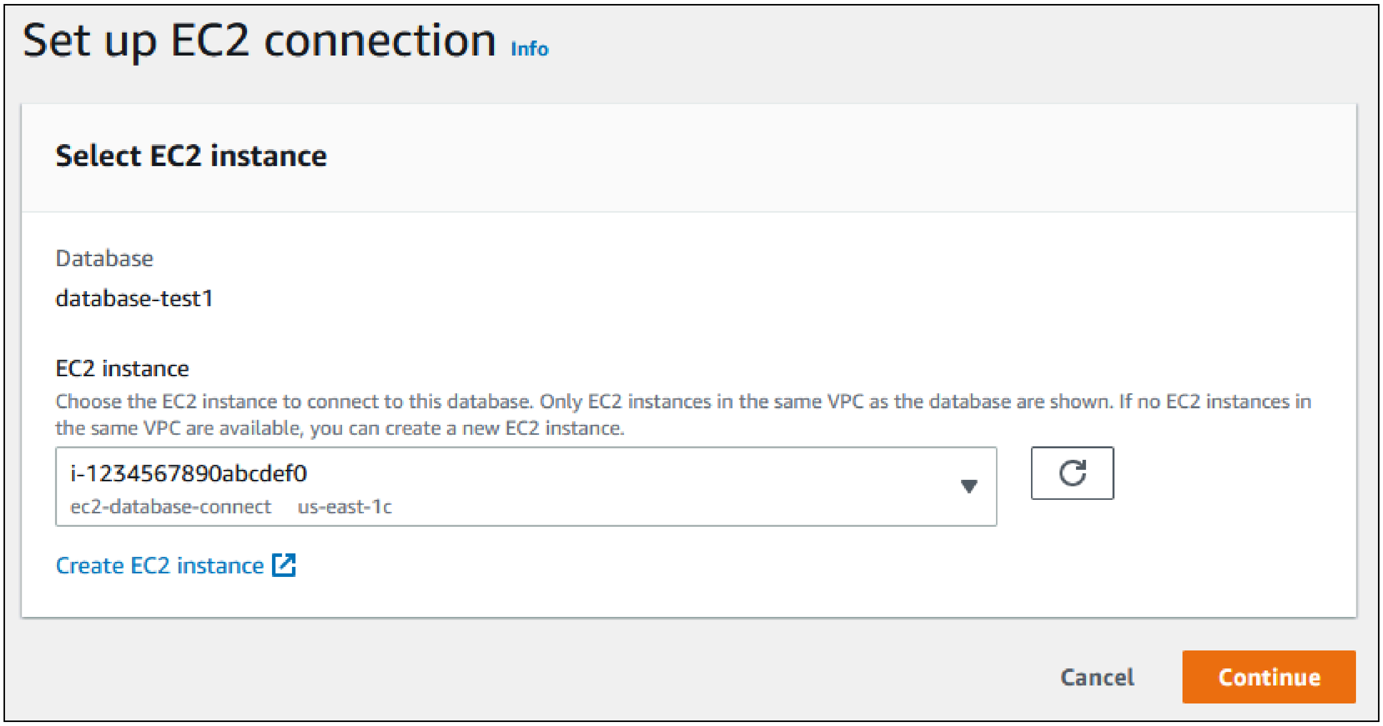
연결을 구성한 후에 보안 그룹을 변경할 경우 변경 사항이 EC2 인스턴스와 Aurora DB 클러스터 간의 연결에 영향을 미칠 수 있습니다.

> **Note**
>
> AWS Management Console을 사용해야만 EC2 인스턴스와 Aurora DB 클러스터 간의 연결을 자동으로 설정할 수 있습니다. AWS CLI 또는 RDS API로는 연결을 자동으로 설정할 수 없습니다.

EC2 인스턴스와 Aurora DB 클러스터를 자동으로 연결하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택하고 DB 클러스터를 선택합니다.
3. 작업에서 EC2 연결 설정을 선택합니다.

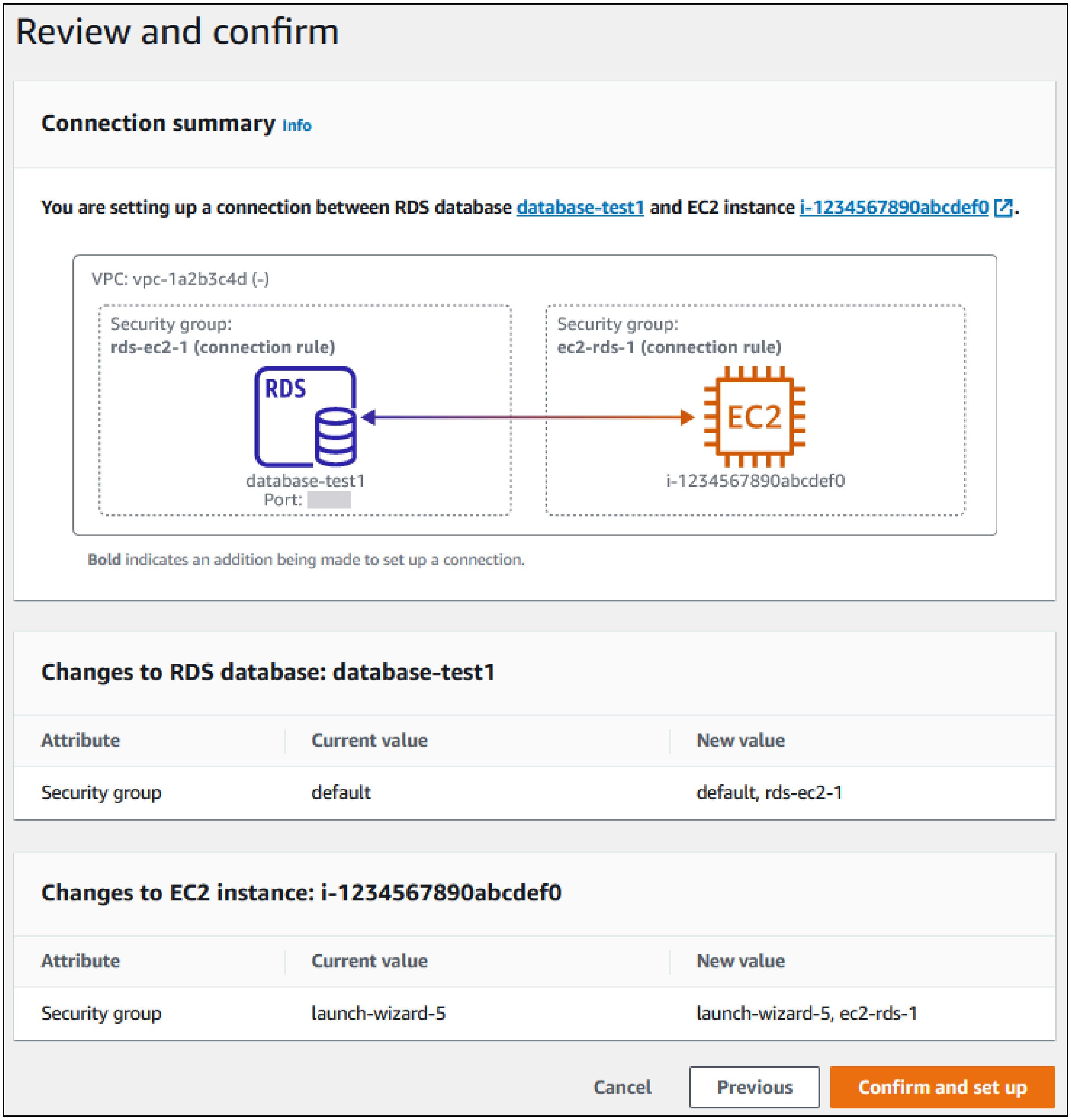
   EC2 연결 설정 페이지가 나타납니다.
4. EC2 연결 설정 페이지에서 EC2 인스턴스를 선택합니다.



동일한 VPC에 EC2 인스턴스가 없는 경우 EC2 인스턴스 생성을 선택하여 인스턴스를 생성합니다. 이 경우 새 EC2 인스턴스가 DB 클러스터와 동일한 VPC 있어야 합니다.

5. 계속을 선택합니다.

   검토 및 확인 페이지가 나타납니다.



6. 검토 및 확인 페이지에서 RDS가 EC2 인스턴스와의 연결을 설정하기 위해 적용할 변경 사항을 검토합니다.

   변경 내용이 올바르면 확인 및 설정을 선택합니다.

   변경 내용이 올바르지 않으면 이전 또는 취소를 선택합니다.

연결된 컴퓨팅 리소스 보기

AWS Management Console을 사용하여 Aurora DB 클러스터에 연결된 컴퓨팅 리소스를 볼 수 있습니다. 표시되는 리소스에는 자동으로 설정된 컴퓨팅 리소스 연결이 포함됩니다. 다음과 같은 방법으로 컴퓨팅 리소스와의 연결을 자동으로 설정할 수 있습니다.

- 데이터베이스를 생성할 때 컴퓨팅 리소스를 선택할 수 있습니다.

  자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.
- 기존 데이터베이스와 컴퓨팅 리소스 간의 연결을 설정할 수 있습니다.

  자세한 내용은 EC2 인스턴스와 Aurora DB 클러스터 자동 연결 단원을 참조하십시오.

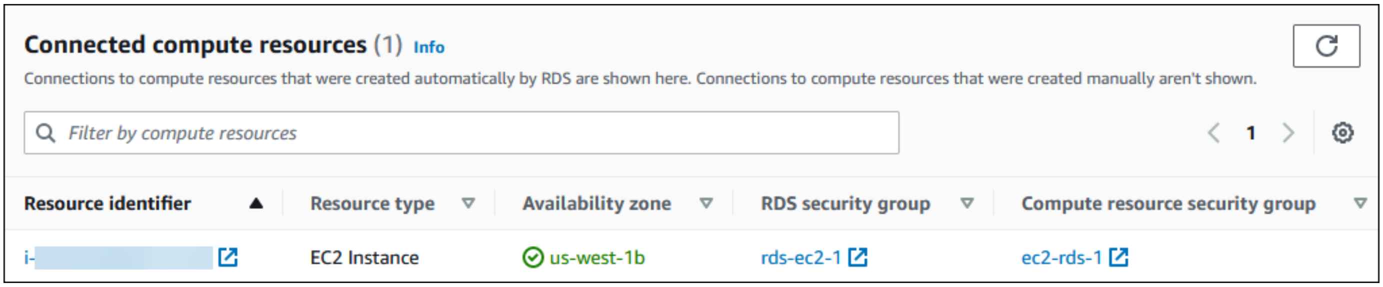
나열된 컴퓨팅 리소스에는 데이터베이스에 수동으로 연결된 리소스는 포함되지 않습니다. 예를 들어 데이터베이스와 연결된 VPC 보안 그룹에 규칙을 추가하여 컴퓨팅 리소스가 데이터베이스에 수동으로 액세스하도록 허용할 수 있습니다.

컴퓨팅 리소스가 나열되려면 다음 조건이 적용되어야 합니다.

- 컴퓨팅 리소스와 연결된 보안 그룹의 이름은 패턴 `ec2-rds-n`(여기서*n*은 숫자)과 일치합니다.
- 컴퓨팅 리소스와 연결된 보안 그룹에는 포트 범위가 DB 클러스터에서 사용하는 포트로 설정된 아웃바운드 규칙이 있습니다.
- 컴퓨팅 리소스와 연결된 보안 그룹에는 소스가 DB 클러스터에 연결된 보안 그룹으로 설정된 아웃바운드 규칙이 있습니다.
- DB 클러스터와 연결된 보안 그룹의 이름은 패턴 `rds-ec2-n`(여기서 *n*은 숫자)과 일치합니다.
- DB 클러스터와 연결된 보안 그룹에는 포트 범위가 DB 클러스터에서 사용하는 포트로 설정된 인바운드 규칙이 있습니다.
- DB 클러스터와 연결된 보안 그룹에는 소스가 컴퓨팅 리소스에 연결된 보안 그룹으로 설정된 인바운드 규칙이 있습니다.

Aurora DB 클러스터에 연결된 컴퓨팅 리소스를 보는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 DB 클러스터의 이름을 선택합니다.
3. 연결 및 보안 탭의 연결된 컴퓨팅 리소스에서 컴퓨팅 리소스를 확인합니다.



특정 DB 엔진을 실행하는 DB 인스턴스에 연결

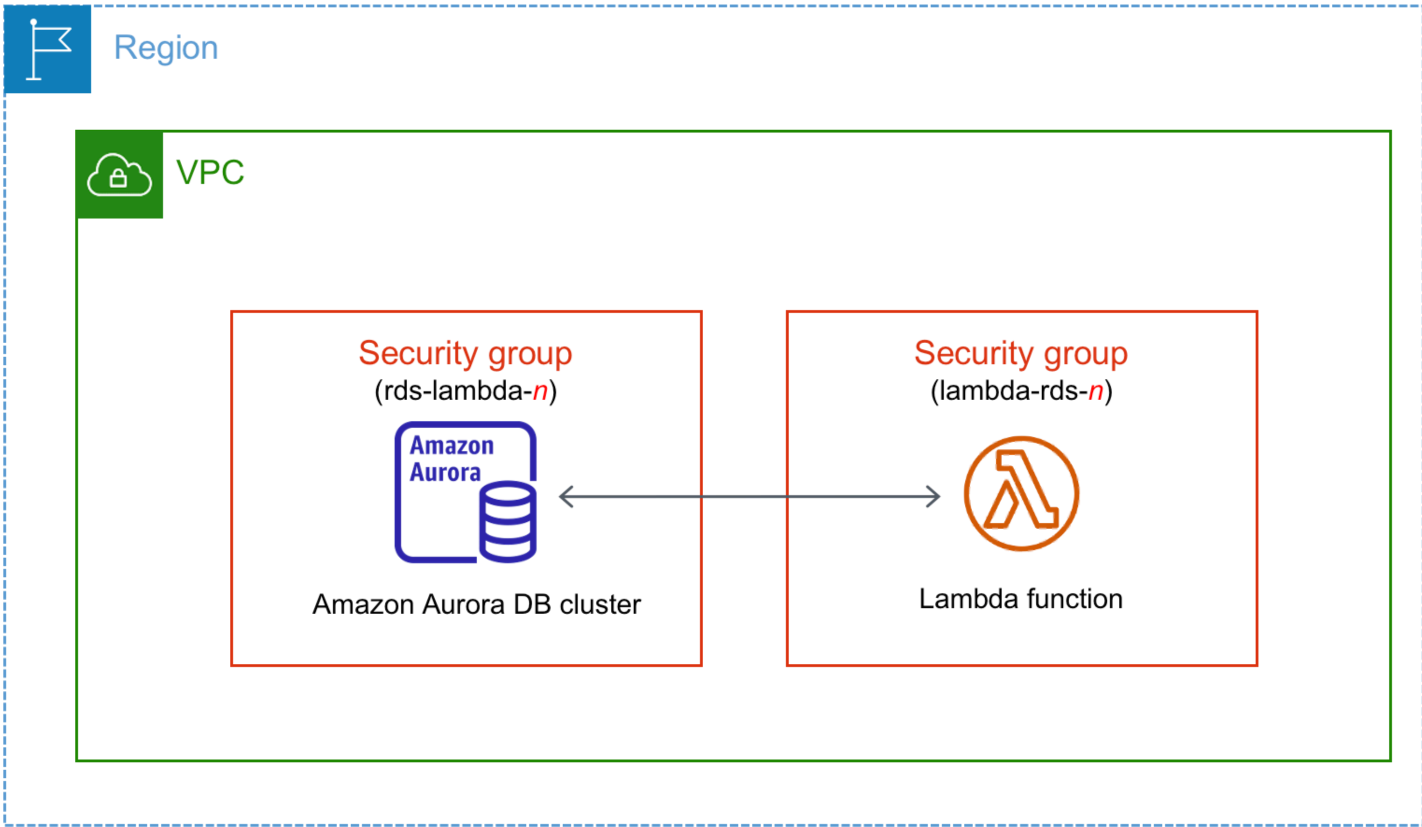
특정 DB 엔진을 실행하는 DB 인스턴스에 연결하는 방법에 대한 자세한 내용은 DB 엔진의 지침을 따르세요.

- Amazon Aurora MySQL DB 클러스터에 연결
- Amazon Aurora PostgreSQL DB 클러스터에 연결

Lambda 함수와 Aurora DB 클러스터 자동 연결

Amazon RDS 콘솔을 사용하여 Lambda 함수와 Aurora DB 클러스터 간의 연결 설정을 간소화할 수 있습니다. DB 클러스터가 VPC 내 프라이빗 서브넷에 있는 경우가 많습니다. 애플리케이션에서 프라이빗 DB 클러스터에 액세스하는 데 Lambda 함수를 사용할 수 있습니다.

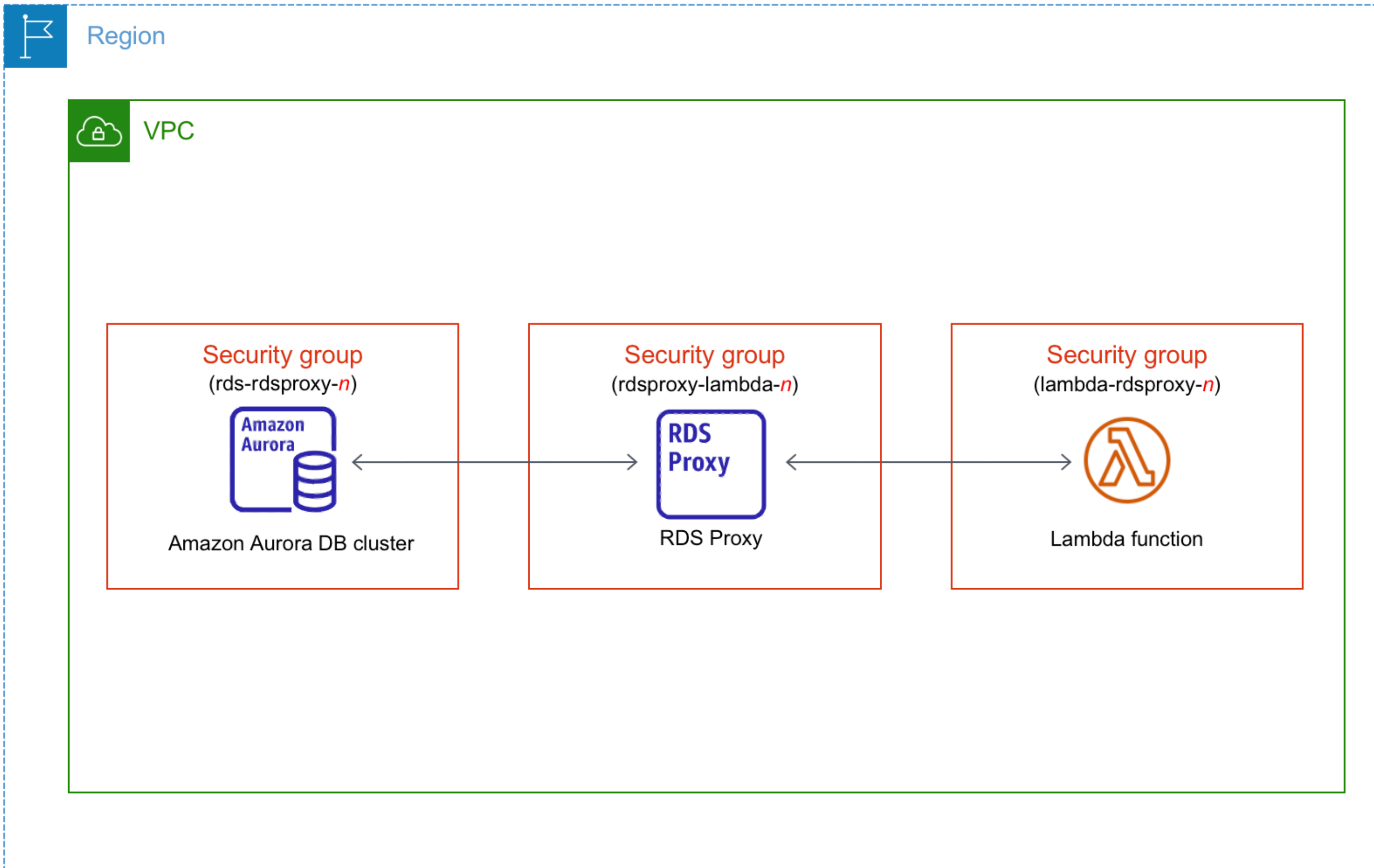
다음 이미지는 DB 클러스터와 Lambda 함수 간의 직접 연결을 보여줍니다.



RDS 프록시를 통해 Lambda 함수와 DB 클러스터 간의 연결을 설정하여 데이터베이스 성능과 복원력을 개선할 수 있습니다. Lambda 함수는 RDS 프록시가 제공하는 연결 풀링의 이점을 누릴 수 있는 단기 데이터베이스 연결을 자주 만드는 경우가 많습니다. Lambda 애플리케이션 코드에서 데이터베이스 보안 인증 정보를 관리하는 대신 Lambda 함수에 대해 이미 가지고 있는 AWS Identity and Access Management(IAM) 인증을 활용할 수 있습니다. 자세한 내용은 Amazon RDS Proxy for Aurora 단원을 참조하십시오.

콘솔을 사용하여 기존 프록시에 연결하면 Amazon RDS가 DB 클러스터와 Lambda 함수 간의 연결을 허용하도록 프록시 보안 그룹을 업데이트합니다.

동일한 콘솔 페이지에서 새 프록시를 만들 수도 있습니다. 콘솔에서 프록시를 만들 때 DB 클러스터에 액세스하려면 데이터베이스 보안 인증 정보를 입력하거나 AWS Secrets Manager 보안 암호를 선택해야 합니다.



> **Tip**
>
> Lambda 함수를 Aurora DB 클러스터에 빠르게 연결하려면 콘솔 내 안내 마법사를 사용할 수도 있습니다. 마법사에서 다음을 수행합니다.
>
> 1. Lambda 콘솔의 [함수 페이지]를 엽니다.
> 2. 데이터베이스를 연결할 함수를 선택합니다.
> 3. 구성 탭에서 RDS 데이터베이스를 선택합니다.
> 4. RDS 데이터베이스에 연결을 선택합니다.
>
> 함수를 데이터베이스에 연결한 후 프록시 추가를 선택하여 프록시를 생성할 수 있습니다.

주제

- [Lambda 함수와의 자동 연결 개요]
- [Lambda 함수와 Aurora DB 클러스터 자동 연결]
- [연결된 컴퓨팅 리소스 보기]

Lambda 함수와의 자동 연결 개요

다음은 Lambda 함수를 Aurora DB 클러스터와 연결하기 위한 요구 사항입니다.

- Lambda 함수가 DB 클러스터와 같은 VPC에 있어야 합니다.
- 현재 DB 클러스터는 Aurora Serverless DB 클러스터일 수 없으며 Aurora Global Database의 일부일 수도 없습니다.
- 연결을 설정하는 사용자는 다음과 같은 Amazon RDS, Amazon EC2, Lambda, Secrets Manager 및 IAM 작업을 수행할 권한이 있어야 합니다.
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBClusters`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBCluster`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`
    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`
  - Lambda
    - `lambda:CreateFunctions`
    - `lambda:ListFunctions`

    - `lambda:UpdateFunctionConfiguration`
- Secrets Manager
    - `secretsmanager:CreateSecret`
    - `secretsmanager:DescribeSecret`
- IAM
    - `iam:AttachPolicy`
    - `iam:CreateRole`
    - `iam:CreatePolicy`
- AWS KMS
    - `kms:describeKey`

> **Note**
>
> DB 클러스터 및 Lambda 함수가 서로 다른 가용 영역에 있는 경우 계정에 교차 가용 영역 비용이 발생할 수 있습니다.

Lambda 함수와 Aurora DB 클러스터 간의 연결을 자동으로 설정할 때 RDS는 함수 및 DB 클러스터에 대한 VPC 보안 그룹을 구성합니다. RDS 프록시를 사용하는 경우 Amazon RDS는 프록시에 대한 VPC 보안 그룹도 구성합니다. Amazon RDS는 다음 테이블에 설명된 것과 같이 DB 클러스터, Lambda 함수 및 프록시와 관련된 보안 그룹의 현재 구성에 따라 작동합니다.

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 하나 이상 있습니다. 또는 프록시가 이미 DB 클러스터에 연결되어 있는 경우, RDS는 연결된 프록	이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n*(*n*은 숫자를 나타냄) 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 하나 이상 있습니다.	이름이 `rdsproxy-lambda-` *n*(*n*은 숫자를 나타냄) 패턴과 일치하며 프록시와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안	Amazon RDS는 아무런 조치도 취하지 않습니다. Lambda 함수, 프록시(선택 사항) 및 DB 클러스터 간에 연결이 이미 자동으로 구성되었습니다. 함수, 프록시, 데이터베이스 사이에

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
시의 `TargetHealth` 가 `AVAILABLE` 인지 확인합니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 있습니다.	해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 DB 클러스터 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙이 하나만 있습니다.	그룹에는 Lambda 함수 및 DB 클러스터의 VPC 보안 그룹이 포함된 인바운드 및 아웃바운드 규칙이 있습니다.	이미 연결이 존재하기 때문에 보안 그룹은 수정되지 않습니다.

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
다음 중 하나의 조건이 적용됩니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. 하지만 이러한 보안 그룹 중 어느 것도 Lambda 함수와의 연결에 사용할 수 없습니다. Amazon RDS는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니	다음 중 하나의 조건이 적용됩니다. • 이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 없습니다. • 이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그	다음 중 하나의 조건이 적용됩니다. • 이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 없습니다. • 이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터 또는 Lambda 함수와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 및 Lambda 함수의 VPC 보안 그룹을 대상으로 하는 인바운드 및 아웃바운드 규칙이 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	RDS action: create new security groups

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
다. 수정 사항의 예로는 규칙 추가 또는 기존 규칙의 포트 변경이 있습니다.	룹을 사용할 수 없습니다.		
이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 있습니다.	이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터 또는 Lambda 함수와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 및 Lambda 함수의 VPC 보안 그룹을 대상으로 하는 인바운드 및 아웃바운드 규칙이 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	[RDS action: create new security groups](#)

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
이름이 `rds-lambda- n` 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 있습니다.	연결에 유효한 Lambda 보안 그룹이 존재하지만 Lambda 함수와 연결되어 있지 않습니다. 이 보안 그룹의 이름이 `lambda-rds- n` 또는 `lambda-rdsproxy- n` 패턴과 일치합니다. 수정되지 않았습니다. DB 클러스터 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙이 하나만 있습니다.	연결에 유효한 프록시 보안 그룹이 있지만 프록시와 연결되어 있지 않습니다. 이 보안 그룹의 이름이 `rdsproxy-lambda- n` 패턴과 일치합니다. 수정되지 않았습니다. DB 클러스터와 Lambda 함수의 VPC 보안 그룹이 포함된 인바운드 및 아웃바운드 규칙이 있습니다.	[RDS action: associate Lambda security group](#)

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
다음 중 하나의 조건이 적용됩니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. 그러나 Amazon RDS는 Lambda 함수 또는 프록시와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그	이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 DB 인스턴스 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙이 하나만 있습니다.	이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 DB 클러스터 및 Lambda 함수의 VPC 보안 그룹이 포함된 인바운드 및 아웃바운드 규칙이 있습니다.	[RDS action: create new security groups]()

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
룹을 사용할 수 없습니다.			

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
다음 중 하나의 조건이 적용됩니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. • 이름이 `rds-lambda-` *n* 패턴과 일치하며 DB 클러스터와 연결된 보안 그룹이 없거나 연결된 프록시의 `TargetHealth` 가 `AVAILABLE` 입니다. 그러나 Amazon RDS는 Lambda 함수 또는 프록시와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그	다음 중 하나의 조건이 적용됩니다. • 이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 없습니다. • 이름이 `lambda-rds-` *n* 또는 `lambda-rdsproxy-` *n* 패턴과 일치하며 Lambda 함수와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 또는 프록시의 VPC 보안 그룹을 소스로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그	다음 중 하나의 조건이 적용됩니다. • 이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 없습니다. • 이름이 `rdsproxy-lambda-` *n* 패턴과 일치하며 프록시와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon RDS는 DB 클러스터 또는 Lambda 함수와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon RDS는 DB 클러스터 및 Lambda 함수의 VPC 보안 그룹을 대상으로 하는 인바운드 및 아웃바운드 규칙이 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 수정된 보안 그룹을 사용할 수 없습니다.	[RDS action: create new security groups](#)

현재 RDS 보안 그룹 구성	현재 Lambda 보안 그룹 구성	현재 프록시 보안 그룹 구성	RDS 작업
룹을 사용할 수 없습니다.	룹을 사용할 수 없습니다.		

RDS 작업: 새 보안 그룹 생성

Amazon RDS에서 다음 작업을 수행합니다.

- RDS 프록시를 사용하도록 선택하는 경우, `rds-lambda-n` 또는 `rds-rdsproxy-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 Lambda 함수 또는 프록시의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나 있습니다. 이 보안 그룹은 DB 클러스터와 연결되어 있으며, 함수가 DB 클러스터에 액세스하도록 허용합니다.
- `lambda-rds-n` 또는 `lambda-rdsproxy-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 DB 클러스터 또는 프록시의 VPC 보안 그룹을 대상으로 하는 아웃바운드 규칙이 하나 있습니다. 이 보안 그룹은 Lambda 함수와 연결되어 있으며, 함수가 트래픽을 DB 클러스터로 보내거나 프록시를 통해 보내도록 허용합니다.
- `rdsproxy-lambda-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 DB 클러스터 및 Lambda 함수의 VPC 보안 그룹이 포함된 인바운드 및 아웃바운드 규칙이 있습니다.

RDS 작업: Lambda 보안 그룹 연결

Amazon RDS는 유효한 기존 Lambda 보안 그룹을 Lambda 함수와 연결합니다. 이 보안 그룹은 함수가 트래픽을 DB 클러스터로 보내거나 프록시를 통해 보내도록 허용합니다.

Lambda 함수와 Aurora DB 클러스터 자동 연결

Amazon RDS 콘솔을 사용하여 Lambda 함수를 DB 클러스터에 자동으로 연결할 수 있습니다. 이렇게 하면 이러한 리소스 간의 연결을 설정하는 프로세스가 간소화됩니다.

RDS 프록시를 사용하여 연결에 프록시를 포함할 수도 있습니다. Lambda 함수는 RDS 프록시가 제공하는 연결 풀링의 이점을 누릴 수 있는 단기 데이터베이스 연결을 자주 만듭니다. Lambda 애플리케이션 코드에서 데이터베이스 보안 인증 정보를 관리하는 대신 Lambda 함수에 대해 이미 설정한 IAM 인증을 사용할 수도 있습니다.

Lambda 연결 설정 페이지를 사용하여 기존 DB 클러스터를 신규 및 기존 Lambda 함수에 연결할 수 있습니다. 이 설정 프로세스는 필요한 보안 그룹을 자동으로 설정합니다.

Lambda 함수와 DB 클러스터 간의 연결을 설정하기 전에 다음 요구 사항을 충족해야 합니다.

- Lambda 함수와 DB 클러스터가 동일한 VPC에 있습니다.
- 사용자 계정에 대한 올바른 권한이 있습니다. 요구 사항에 대한 자세한 내용은 Lambda 함수와의 자동 연결 개요 섹션을 참조하세요.

연결을 구성한 후에 보안 그룹을 변경할 경우 변경 사항이 Lambda 함수와 DB 클러스터 간의 연결에 영향을 미칠 수 있습니다.

> **Note**
>
> AWS Management Console에서만 DB 클러스터와 Lambda 함수의 연결을 자동으로 설정할 수 있습니다. Lambda 함수를 연결하려면 DB 클러스터의 모든 인스턴스가 사용 가능 상태여야 합니다.

Lambda 함수와 DB 클러스터를 자동으로 연결하는 방법

<result>

설정을 확인하면 Amazon RDS가 Lambda 함수, RDS 프록시(프록시를 사용한 경우) 및 DB 클러스터 연결 프로세스를 시작합니다. 콘솔에 연결 세부 정보 대화 상자가 표시되며, 여기에 리소스 간 연결을 허용하는 보안 그룹 변경 사항이 나열됩니다.
</result>

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음, Lambda 함수에 연결할 DB 클러스터를 선택합니다.
3. 작업에서 Lambda 연결 설정을 선택합니다.
4. Lambda 연결 설정 페이지의 Lambda 함수 선택에서 다음 중 하나를 수행합니다.
   - DB 클러스터와 동일한 VPC에 기존 Lambda 함수가 있는 경우 기존 함수 선택을 선택하고 해당 함수를 선택합니다.
   - 동일한 VPC에 Lambda 함수가 없는 경우 새 함수 생성을 선택한 다음 함수 이름을 입력합니다. 기본 런타임은 Nodejs.18로 설정되어 있습니다. 연결 설정을 완료한 후 Lambda 콘솔에서 새 Lambda 함수의 설정을 수정할 수 있습니다.
5. (선택 사항) RDS 프록시에서 RDS 프록시를 사용하여 연결을 선택하고 다음 중 하나를 수행합니다.

- 사용하려는 기존 프록시가 있는 경우 기존 프록시 선택을 선택하고 해당 프록시를 선택합니다.
- 프록시가 없고 Amazon RDS에서 자동으로 프록시를 생성하도록 하려면 새 프록시 생성을 선택합니다. 그런 다음, 데이터베이스 보안 인증 정보에서 다음 중 하나를 수행합니다.

  a. 데이터베이스 사용자 이름 및 암호를 선택한 다음, DB 클러스터의 사용자 이름과 암호를 입력합니다.

  b. Secrets Manager 보안 암호를 선택합니다. 그런 다음, 보안 암호 선택에서 AWS Secrets Manager 보안 암호를 선택합니다. Secrets Manager 보안 암호가 없다면 새 Secrets Manager 보안 암호 생성을 선택하여 새 보안 암호를 생성합니다. 보안 암호를 생성한 후 보안 암호 선택에서 새 보안 암호를 선택합니다.

  새 프록시를 생성한 후 기존 프록시 선택을 선택하고 해당 프록시를 선택합니다. 프록시를 연결에 사용하려면 다소 시간이 걸릴 수 있습니다.

6. (선택 사항) 연결 요약을 확장하고 리소스에서 강조 표시된 업데이트를 확인합니다.
7. 설정을 선택합니다.

연결된 컴퓨팅 리소스 보기

AWS Management Console을 사용하여 DB 클러스터에 연결된 Lambda 함수를 볼 수 있습니다. 표시되는 리소스에는 Amazon RDS가 자동으로 설정한 컴퓨팅 리소스 연결이 포함됩니다.

나열된 컴퓨팅 리소스에는 DB 클러스터에 수동으로 연결된 컴퓨팅 리소스가 포함되지 않습니다. 예를 들어 데이터베이스와 연결된 VPC 보안 그룹에 규칙을 추가하여 컴퓨팅 리소스가 DB 클러스터에 수동으로 액세스하도록 허용할 수 있습니다.

콘솔에 Lambda 함수를 나열하려면 다음 조건을 적용해야 합니다.

- 컴퓨팅 리소스와 연결된 보안 그룹의 이름이 `lambda-rds-`*n* 또는 `lambda-rdsproxy-`*n*(*n*은 숫자를 나타냄) 패턴과 일치합니다.
- 컴퓨팅 리소스와 연결된 보안 그룹에 포트 범위가 DB 클러스터 또는 프록시의 포트로 설정된 아웃바운드 규칙이 있습니다. 아웃바운드 규칙의 대상이 DB 클러스터 또는 관련 프록시와 연결된 보안 그룹으로 설정되어 있어야 합니다.
- 구성에 프록시가 포함된 경우 데이터베이스와 연결된 프록시에 연결된 보안 그룹의 이름이 `rdsproxy-lambda-`*n*(*n*은 숫자를 나타냄) 패턴과 일치합니다.

- 함수와 연결된 보안 그룹에 포트가 DB 클러스터 또는 관련 프록시가 사용하는 포트로 설정된 아웃바운드 규칙이 있습니다. 대상이 DB 클러스터 또는 관련 프록시와 연결된 보안 그룹으로 설정되어 있어야 합니다.

DB 클러스터에 자동으로 연결된 컴퓨팅 리소스를 보는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택하고 DB 클러스터를 선택합니다.
3. 연결 및 보안 탭의 연결된 컴퓨팅 리소스에서 컴퓨팅 리소스를 확인합니다.

Amazon Aurora DB 클러스터 수정

백업 보관 기간 변경 또는 데이터베이스 포트 변경과 같은 작업을 수행하기 위해 DB 클러스터의 설정을 변경할 수 있습니다. DB 인스턴스 클래스 변경이나 성능 개선 도우미 사용 설정 같은 작업을 수행하기 위해 DB 클러스터에서 DB 인스턴스를 수정할 수 있습니다. 이번 주제에서는 Aurora DB 클러스터와 그 DB 인스턴스를 수정하는 과정을 안내하고 각각에 대한 설정에 대해서 설명하겠습니다.

프로덕션 DB 클러스터 또는 DB 인스턴스를 수정하기 전에 테스트 DB 클러스터 또는 DB 인스턴스에서 변경 사항을 테스트하면 각 변경 사항이 미칠 영향을 완전히 이해하는 데 도움이 됩니다. 이는 특히 데이터베이스 버전을 업그레이드할 때 중요합니다.

주제

콘솔, CLI, API를 사용하여 DB 클러스터 수정

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 수정할 수 있습니다.

> **Note**
>
> 대부분의 수정 사항은 즉시 적용하거나 예정된 다음 유지 관리 기간에 적용할 수 있습니다. 삭제 방지 기능 활성화와 같은 일부 수정 사항은 적용 시기를 언제로 선택하든 상관없이 즉시 적용됩니다.
> AWS Management Console에서 마스터 암호를 변경하면 항상 즉시 적용됩니다. 그러나 AWS CLI 또는 RDS API를 사용하는 경우 이 변경 사항을 즉시 적용할지 아니면 예정된 다음 유지 관리 기간에 적용할지 선택할 수 있습니다.
> SSL 엔드포인트를 사용하고 있으며 DB 클러스터 식별자를 변경하는 경우, DB 클러스터를 중지하고 다시 시작하여 SSL 엔드포인트를 업데이트해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터 중지 및 시작 단원을 참조하십시오.

콘솔

DB 클러스터를 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 수정하려는 DB 클러스터를 선택합니다.
3. 수정을 선택합니다. Modify DB cluster(DB 클러스터 수정) 페이지가 나타납니다.
4. 원하는 설정을 모두 변경합니다. 각 설정에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

   > **Note**
   >
   > AWS Management Console에서는 현재 DB 인스턴스에 인스턴스 수준의 일부 변경 사항만 적용되지만 전체 DB 클러스터에는 다른 변경 사항이 적용됩니다. DB 인스턴스 또는 DB 클러스터에 설정 적용 여부에 대한 내용은 Amazon Aurora에 대한 설정의 설정 범위를 참조하십시오. AWS Management Console에서 인스턴스 수준에서 전체 DB 클러스터를 수정하는 설정을 변경하려면 DB 클러스터에서 DB 인스턴스 수정 단원의 지침을 따르십시오.

5. 원하는 대로 모두 변경되었으면 [Continue]를 선택하고 수정 사항 요약을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.
7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 올바른 경우 클러스터 수정을 선택하여 변경 내용을 저장합니다.

   그렇지 않으면 [Back]을 선택하여 변경 내용을 편집하거나 [Cancel]을 선택하여 변경 내용을 취소합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터를 수정하려면 modify-db-cluster 명령을 호출하십시오. DB 클러스터 식별자와 수정하려는 설정 값을 지정하십시오. 각 설정에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

> **Note**
>
> 일부 설정은 DB 인스턴스에만 적용됩니다. 이러한 설정을 변경하려면 DB 클러스터에서 DB 인스턴스 수정의 지침을 따르십시오.

Example

다음 명령은 백업 보관 기간을 1주(7일)로 설정하여 `mydbcluster`를 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터를 수정하려면 ModifyDBCluster 작업을 호출하십시오. DB 클러스터 식별자와 수정하려는 설정 값을 지정하십시오. 각 파라미터에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

> **Note**
>
> 일부 설정은 DB 인스턴스에만 적용됩니다. 이러한 설정을 변경하려면 DB 클러스터에서 DB 인스턴스 수정의 지침을 따르십시오.

DB 클러스터에서 DB 인스턴스 수정

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터에서 DB 인스턴스를 수정할 수 있습니다.

DB 인스턴스를 수정하는 경우 변경 사항을 즉시 적용할 수 있습니다. 변경 사항을 즉시 적용하려면 AWS Management Console에서 즉시 적용(Apply Immediately) 옵션을 선택하거나, AWS CLI 호출 시 `--apply-immediately` 파라미터를 사용하거나, Amazon RDS API 사용 시 `ApplyImmediately` 파라미터를 `true`로 설정합니다.

변경 사항을 즉시 적용하도록 선택하지 않으면 변경 사항이 다음 유지 관리 기간까지 연기됩니다. 다음 유지 관리 기간 동안 지연된 변경 사항이 모두 적용됩니다. 변경 사항을 즉시 적용하도록 선택하면 새 변경 사항과 이전에 지연된 변경 사항이 적용됩니다.

다음 유지 관리 기간에 보류 중인 수정 사항을 보려면 describe-db-clusters AWS CLI 명령을 사용하고 `PendingModifiedValues` 필드를 확인합니다.

> **Important**
>
> 보류 중인 수정 작업으로 인해 가동 중지가 필요한 경우, 즉시 적용을 선택하면 DB 인스턴스에서 예기치 못한 가동 중지가 발생할 수 있습니다. DB 클러스터의 다른 DB 인스턴스에서는 가동 중지가 발생하지 않습니다.
> 지연한 수정 사항은 `describe-pending-maintenance-actions` CLI 명령의 출력에 나열되지 않습니다. 유지 관리 작업에는 다음 유지 관리 기간에 대해 예약하는 시스템 업그레이드만 포함됩니다.

콘솔

DB 클러스터에서 DB 인스턴스를 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 변경하려는 DB 인스턴스를 선택합니다.
3. 작업에서 수정을 선택합니다. DB 인스턴스 수정 페이지가 나타납니다.
4. 원하는 설정을 모두 변경합니다. 각 설정에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

   > **Note**
   >
   > 일부 설정은 전체 DB 클러스터에 적용되며, 클러스터 수준에서 변경이 필요합니다. 이러한 설정을 변경하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정의 지침을 따르십시오.

AWS Management Console에서는 현재 DB 인스턴스에 인스턴스 수준의 일부 변경 사항만 적용되지만 전체 DB 클러스터에는 다른 변경 사항이 적용됩니다. DB 인스턴스 또는 DB 클러스터에 설정 적용 여부에 대한 내용은 Amazon Aurora에 대한 설정의 설정 범위를 참조하십시오.

5. 원하는 대로 모두 변경되었으면 [Continue]를 선택하고 수정 사항 요약을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.
7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 정확할 경우 DB 인스턴스 수정을 선택하여 변경 내용을 저장합니다.

   그렇지 않으면 [Back]을 선택하여 변경 내용을 편집하거나 [Cancel]을 선택하여 변경 내용을 취소합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터의 DB 인스턴스를 수정하려면 modify-db-instance 명령을 호출하십시오. DB 인스턴스 식별자와 수정하려는 설정 값을 지정하십시오. 각 파라미터에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

> **Note**
>
> 일부 설정은 전체 DB 클러스터에 적용됩니다. 이러한 설정을 변경하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정의 지침을 따르십시오.

Example

다음 코드는 DB 인스턴스 클래스를 `mydbinstance`로 설정하여 `db.r4.xlarge`를 수정합니다. 변경 사항은 `--no-apply-immediately`를 사용하여 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 `--apply-immediately`를 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS API

Amazon RDS API를 사용하여 DB 인스턴스를 수정하려면 ModifyDBInstance 작업을 호출합니다. DB 인스턴스 식별자와 수정하려는 설정 값을 지정하십시오. 각 파라미터에 대한 자세한 내용은 Amazon Aurora에 대한 설정 단원을 참조하십시오.

> **Note**
> 일부 설정은 전체 DB 클러스터에 적용됩니다. 이러한 설정을 변경하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정의 지침을 따르십시오.

데이터베이스 마스터 사용자의 암호 변경

AWS Management Console 또는 AWS CLI를 사용하여 마스터 사용자의 암호를 변경할 수 있습니다.

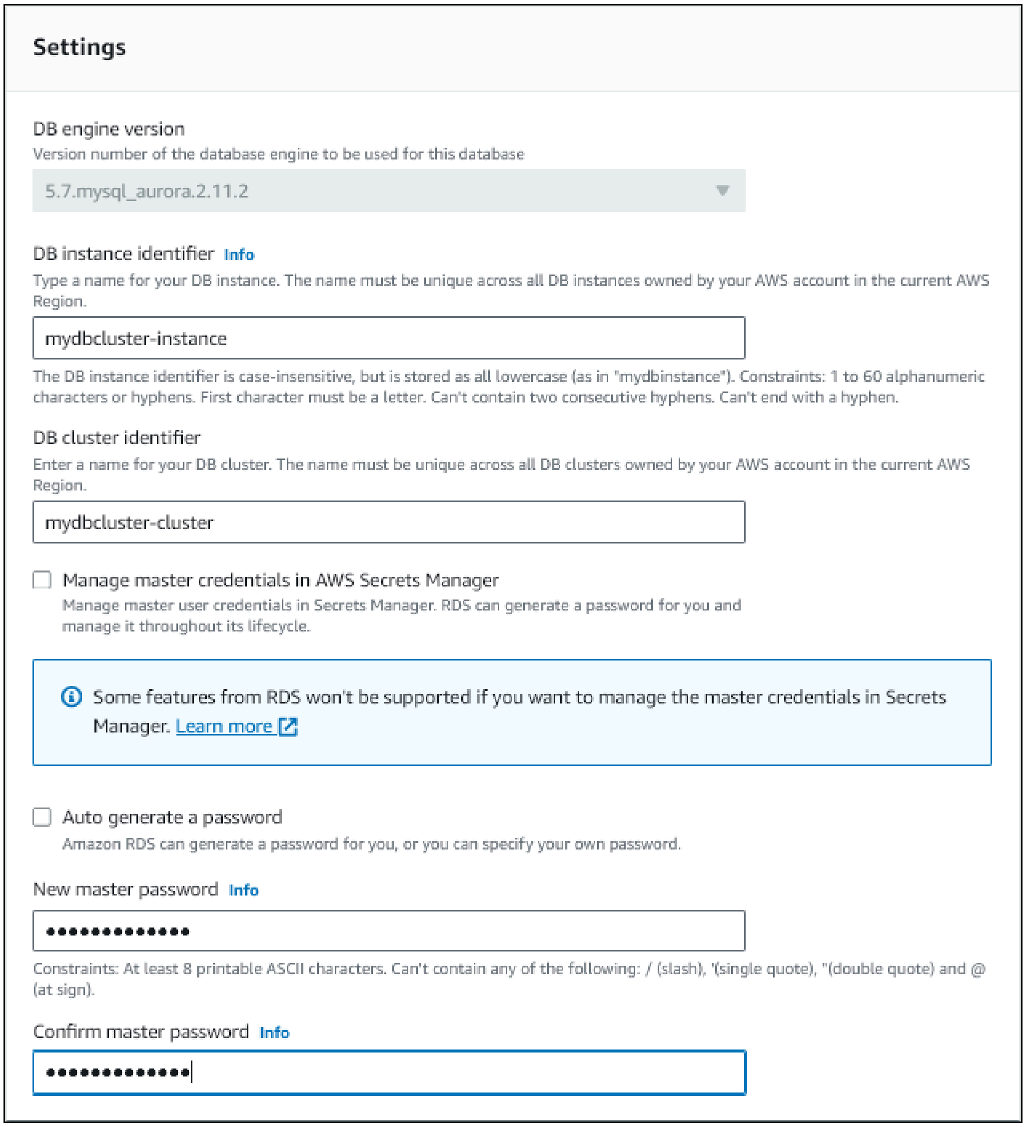
콘솔

AWS Management Console을 사용하여 마스터 사용자 암호를 변경하려면 라이터 DB 인스턴스를 수정합니다.

마스터 사용자의 암호를 변경하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 변경하려는 DB 인스턴스를 선택합니다.
3. 작업에서 수정을 선택합니다.

   DB 인스턴스 수정 페이지가 나타납니다.
4. 새 마스터 암호를 입력합니다.
5. 마스터 암호 확인에 동일한 새 암호를 입력합니다.



6. 계속해서 수정 사항을 요약한 내용을 확인합니다.

> **Note**
>
> 변경 사항이 바로 적용됩니다.

7. 확인 페이지에서 DB 인스턴스 수정(Modify DB instance)을 선택합니다.

CLI

AWS CLI를 사용하여 마스터 사용자 암호를 변경하려면 modify-db-cluster 명령을 호출합니다. 다음 예제와 같이, DB 클러스터 식별자와 새 암호를 지정합니다.

암호 변경은 항상 즉시 적용되므로 `--apply-immediately|--no-apply-immediately`를 지정할 필요가 없습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Amazon Aurora에 대한 설정

다음 표에는 수정이 가능한 설정, 설정 수정 방법, 설정 범위에 대한 세부 정보가 나와 있습니다. 이 범위를 통해 설정을 전체 DB 클러스터에 적용할지 또는 특정 DB 인스턴스에만 설정할 수 있는지 여부를 결정합니다.

> **Note**
>
> Aurora Serverless v1 또는 Aurora Serverless v2 DB 클러스터를 수정하는 경우 추가 설정을 사용할 수 있습니다. 이러한 설정에 대한 내용은 Aurora Serverless v1 DB 클러스터 수정 및 Aurora Serverless v2 DB 클러스터 관리 섹션을 참조하세요.
> Aurora Serverless v1 및 Aurora Serverless v2에서는 제한 사항이 적용되어 일부 설정을 사용할 수 없습니다. 자세한 내용은 Aurora Serverless v1의 제한 사항 및 Aurora Serverless v2 요구 사항 및 제한 사항 단원을 참조하세요.

설정 및 설명	방법	범위	가동 중지 참고 사항
자동 마이너 버전 업그레이드 사용 가능하게 되면 DB 인스턴스에 기본	**Note** 이 설정은 기본적으로 활성화되어 있습니	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다. 이후 유지 관리 기간 동안 Aurora가 자동 업

설정 및 설명	방법	범위	가동 중지 참고 사항
마이너 엔진 버전 업그레이드를 자동으로 받을지 여부. 업그레이드는 예약된 유지 관리 기간 중에만 설치됩니다. Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트의 엔진 업데이트에 대한 자세한 정보는 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오. Aurora MySQL의 마이너 버전 자동 업그레이드(Auto minor version upgrade) 설정에 대한 자세한 내용은 마이너 Aurora MySQL 버전 간 자동 업그레이드 활성화 단원을 참조하세요.	다. 각 새 클러스터에 대해 중요도, 예상 수명 및 각 업그레이드 후에 수행하는 확인 테스트 양에 따라 이 설정에 적합한 값을 선택합니다. 이 설정을 변경하는 경우 Aurora 클러스터의 모든 DB 인스턴스에 대해 이 수정 작업을 수행합니다. 클러스터의 DB 인스턴스에서 이 설정이 꺼져 있으면 클러스터가 자동으로 업그레이드되지 않습니다. AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 modify-db-instance를 실행하고 `--auto-minor-version-upgrade|--no-auto-`		그레이드를 적용할 때는 중단이 발생합니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
	`minor-version-upgrade` 옵션을 설정합니다. RDS API를 사용하여 [ModifyDBInstance](#)를 직접 호출하고 `AutoMinorVersionUpgrade` 파라미터를 설정합니다.		
백업 보관 기간 자동 백업을 보관할 일 수. 최소값은 1입니다. 자세한 내용은 [백업](#) 섹션을 참조하세요.	AWS Management Console, [콘솔, CLI, API를 사용하여 DB 클러스터 수정](#) 사용. AWS CLI를 사용하여 [modify-db-cluster](#)를 실행하고 `--backup-retention-period` 옵션을 설정합니다. RDS API를 사용하여 [ModifyDBCluster](#)를 직접 호출하고 `BackupRetentionPeriod` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
백업 기간(시작 시간) 데이터베이스의 자동 백업이 실행되는 기간. 백업 기간은 국제 표준시(UTC)의 시작 시간과 시간 단위의 지속 기간으로 구성됩니다. Aurora 백업은 연속 및 증분 백업이지만 백업 기간은 백업 유지 기간 내에 보관되는 일일 시스템 백업을 만드는 데 사용됩니다. 이것을 복사하여 유지 기간이 지난 뒤에도 보관할 수 있습니다. DB 클러스터에 대한 유지 관리 기간 및 백업 기간은 겹칠 수 없습니다. 자세한 내용은 백업 기간 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 modify-db-cluster를 실행하고 `--preferred-backup-window` 옵션을 설정합니다. RDS API를 사용하여 ModifyDBCluster를 직접 호출하고 `PreferredBackupWindow` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
용량 설정 Aurora Serverless v1 DB 클러스터의 규모 조정 속성입니다. DB 클러스터의 조정 속성은 `serverless` DB 엔진 모드에서만 수정할 수 있습니다. Aurora Serverless v1에 대한 자세한 내용은 Amazon Aurora Serverless v1 사용 단원을 참조하십시오.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 modify-db-cluster를 실행하고 `--scaling-configuration` 옵션을 설정합니다. RDS API를 사용하여 ModifyDBCluster를 직접 호출하고 `ScalingConfiguration` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다. 변경 사항이 즉시 적용됩니다. 이 설정은 즉시 적용 설정을 무시합니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
인증 기관 DB 인스턴스에서 사용하는 서버 인증서의 CA(인증 기관)입니다.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 modify-db-instance 를 실행하고 `--ca-certificate-identifier` 옵션을 설정합니다. RDS API를 사용하여 ModifyDBInstance를 직접 호출하고 `CACertificateIdentifier` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	중단은 DB 엔진이 재시작 없는 교체를 지원하지 않는 경우에만 발생합니다. describe-db-engine-versions AWS CLI 명령을 사용하여 DB 엔진이 재시작 없는 교체를 지원하는지 여부를 확인할 수 있습니다.
클러스터 스토리지 구성 DB 클러스터의 스토리지 유형(Aurora I/O-Optimized 또는 Aurora Standard)입니다. 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 단원을 참조하십시오.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 modify-db-cluster를 실행하고 `--storage-type` 옵션을 설정합니다. RDS API를 사용하여 ModifyDBCluster를 직접 호출하고 `StorageType` 파라미터를 설정합니다.	전체 DB 클러스터	최적화된 읽기 인스턴스 클래스가 있는 Aurora PostgreSQL DB 클러스터의 스토리지 유형을 변경하면 중단이 발생합니다. 다른 인스턴스 클래스 유형이 있는 클러스터의 스토리지 유형을 변경할 때는 이 문제가 발생하지 않습니다. DB 인스턴스 클래스 유형에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

설정 및 설명	방법	범위	가동 중지 참고 사항
스냅샷으로 태그 복사 이 옵션을 선택하면 현재 DB 클러스터에 정의되어 있는 태그가 현재 DB 클러스터에서 생성된 DB 스냅샷으로 복사됩니다. 자세한 내용은 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html를 실행하고 `--copy-tags-to-snapshot` 또는 `--no-copy-tags-to-snapshot` 옵션을 설정합니다. RDS API를 사용하여 https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBCluster.html를 호출하고 `CopyTagsToSnapshot` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
데이터 API Aurora Serverless v1 및 AWS Lambda를 포함한 웹 서비스 기반 애플리케이션을 사용하여 AWS AppSync에 액세스할 수 있습니다. 이 설정은 Aurora Serverless v1 DB 클러스터에만 적용됩니다. 자세한 내용은 Amazon RDS Data API 사용 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--enable-http-endpoint` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `EnableHttpEndpoint` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
데이터베이스 인증 사용하고자 하는 데이터베이스 인증입니다. MySQL의 경우: • 데이터베이스 암호로만 데이터베이스 사용자를 인증할 수 있도록 Password authentication(암호 인증)을 선택합니다. • IAM 사용자 및 역할을 통해 데이터베이스 암호 및 사용자 자격 증명으로 데이터베이스 사용자를 인증하려면 [암호 및 IAM 데이터베이스 인증(Password and IAM database authentication)]을 선택합니다. 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요. PostgreSQL의 경우: • IAM 사용자 및 역할을 통해 데이터베이스 암호 및 사용자 자격 증명으로 데이터베이스 사용자	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 modify-db-cluster를 실행하고 다음 옵션을 설정합니다. • IAM 인증의 경우 `--enable-iam-database-authentication|--no-enable-iam-database-authentication` 옵션을 설정합니다. • Kerberos 인증의 경우 `--domain` 및 `--domain-iam-role-name` 옵션을 설정합니다. RDS API를 사용하여 ModifyDBCluster를 호출하고 다음 파라미터를 설정합니다. • IAM 인증의 경우 `EnableIAMDatabaseAuthentica`	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
를 인증하려면 IAM database authentication(IAM 데이터베이스 인증)을 선택합니다. 자세한 내용은 [IAM 데이터베이스 인증](#) 단원을 참조하십시오. • Kerberos 인증을 사용하여 데이터베이스 암호 및 사용자 자격 증명을 인증하려면 [Kerberos authentication(Kerberos 인증)]을 선택합니다. 자세한 내용은 [Aurora PostgreSQL과 함께 Kerberos 인증 사용](#) 섹션을 참조하세요.	`tion` 파라미터를 설정합니다. • Kerberos 인증의 경우 `Domain` 및 `DomainIAMRoleName` 파라미터를 설정합니다.		

설정 및 설명	방법	범위	가동 중지 참고 사항
데이터베이스 포트 DB 클러스터에 액세스하는 데 사용할 포트.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--port` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `Port` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단됩니다. DB 클러스터의 모든 DB 인스턴스는 즉시 재부팅됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 클러스터 식별자 DB 클러스터 식별자입니다. 이 값은 소문자 문자열로 저장됩니다. DB 클러스터 식별자를 변경하면 DB 클러스터 엔드포인트가 변경됩니다. DB 클러스터에 있는 DB 인스턴스의 엔드포인트는 변경되지 않습니다.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster` 를 실행하고 `--new-db-cluster-identifier` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `NewDBClusterIdentifier` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 클러스터 파라미터 그룹 DB 클러스터와 연결할 DB 클러스터 파라미터 그룹. 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--db-cluster-parameter-group-name` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `DBClusterParameterGroupName` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다. 파라미터 그룹을 변경하는 경우 일부 파라미터에 대한 변경 내용은 재부팅 없이 DB 클러스터의 DB 인스턴스에 즉시 적용됩니다. 다른 파라미터에 대한 변경 내용은 DB 클러스터의 DB 인스턴스를 재부팅한 후에만 적용됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 인스턴스 클래스 사용할 DB 인스턴스 클래스. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--db-instance-class` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `DBInstanceClass` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 인스턴스 식별자 DB 인스턴스 식별자 이 값은 소문자 문자열로 저장됩니다.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--new-db-instance-identifier` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `NewDBInstanceIdentifier` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중에는 가동 중지 시간이 발생합니다. RDS가 DB 인스턴스를 다시 시작하여 다음을 업데이트합니다. • Aurora MySQL - `information_schema.replica_host_status` 테이블에 있는 `SERVER_ID` 열 • Aurora PostgreSQL - `aurora_replica_status()` 함수의 `server_id` 열

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 파라미터 그룹 DB 인스턴스와 연결하려는 DB 파라미터 그룹. 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--db-parameter-group-name` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `DBParameterGroupName` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다. 새 DB 파라미터 그룹을 DB 인스턴스와 연결하면 수정된 정적 파라미터 및 동적 파라미터는 DB 인스턴스가 재부팅된 후에만 적용됩니다. 그러나 DB 파라미터 그룹을 DB 인스턴스에 연결한 후 DB 파라미터 그룹에서 동적 파라미터를 수정하면 이러한 변경 사항이 재부팅 없이 즉시 적용됩니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹 및 Amazon Aurora DB 클러스터 또는 Amazon Aurora DB 인스턴스 재부팅 단원을 참조하세요.

설정 및 설명	방법	범위	가동 중지 참고 사항
삭제 방지 DB 클러스터가 삭제되지 않도록 방지하기 위한 삭제 방지 활성화 자세한 내용은 Aurora 클러스터의 삭제 방지 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--deletion-protection|--no-deletion-protection` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `DeletionProtection` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
엔진 버전 사용하려는 DB 엔진의 버전입니다. 프로덕션 DB 클러스터스를 업그레이드하려면 먼저 테스트 DB 클러스터에서 업그레이드 프로세스를 테스트하여 지속 시간을 확인하고 애플리케이션의 유효성을 확인하는 것이 좋습니다.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--engine-version` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `EngineVersion` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
확장 모니터링 DB 인스턴스가 실행되는 운영 체제에 대한 실시간 측정치 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--monitoring-role-arn` 및 `--monitoring-interval` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `MonitoringRoleArn` 및 `MonitoringInterval` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
로그 내보내기 Amazon CloudWatch Logs에 게시할 로그 유형을 선택합니다. 자세한 내용은 Aurora MySQL 데이터베이스 로그 파일 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--cloudwatch-logs-export-configuration` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `CloudwatchLogsExportConfiguration` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
유지보수 윈도우 시스템 유지 관리를 실행하는 기간. 시스템 유지 관리는 업그레이드를 포함합니다(해당할 경우). 유지 관리 기간은 국제 표준시(UTC)의 시작 시간과 시간 단위의 지속 기간으로 구성됩니다. 이 기간을 현재 시간으로 설정하려면 대기 중인 변경 사항이 모두 적용될 수 있도록 현재 시간과 기간 종료 시간 사이에 최소 30분 이상 필요합니다. 유지 관리 기간을 DB 클러스터 및 DB 클러스터의 각 DB 인스턴스에 대해 독자적으로 설정할 수 있습니다. 수정 범위가 전체 DB 클러스터이면 수정은 DB 클러스터 유지 관리 기간 중에 수행됩니다. 수정 범위가 전체 DB 인스턴스이면 수정은 이 DB 인스턴스의 유지 관리 기간 중에 수행됩니다.	AWS Management Console을 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정. AWS Management Console을 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 DB 클러스터에서 DB 인스턴스 수정. AWS CLI를 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 `modify-db-cluster`를 실행하고 `--preferred-maintenance-window` 옵션을 설정합니다. AWS CLI를 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 `modify-db-instance` 를 실행하고 `--preferred-maintenance-window` 옵션을 설정합니다.	전체 DB 클러스터 또는 단일 DB 인스턴스	인스턴스가 중단될 수 있는 작업이 하나 이상 대기 중이고, 유지 관리 기간이 현재 시간을 포함하여 변경된 경우 대기 중인 작업들이 즉시 적용되고 인스턴스가 중단됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
DB 클러스터에 대한 유지 관리 기간 및 백업 기간은 겹칠 수 없습니다. 자세한 내용은 [Amazon RDS 유지 관리 기간](#) 섹션을 참조하세요.	RDS API를 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 [`ModifyDBCluster`](#) 를 호출하고 `PreferredMaintenanceWindow` 파라미터를 설정합니다. RDS API를 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 [`ModifyDBInstance`](#) 를 호출하고 `PreferredMaintenanceWindow` 파라미터를 설정합니다.		

설정 및 설명	방법	범위	가동 중지 참고 사항
AWS Secrets Manager에서 마스터 자격 증명 관리 Secrets Manager에서 보안 암호의 마스터 사용자 암호를 관리하려면 AWS Secrets Manager에서 마스터 자격 증명 관리를 선택합니다. 원하는 경우 보안 암호를 보호하는 데 사용할 KMS 키를 선택합니다. 자신의 계정에서 KMS 키를 선택하거나 다른 계정의 키를 입력할 수 있습니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 단원을 참조하십시오. Aurora에서 이미 DB 클러스터의 마스터 사용자 암호를 관리하고 있는 경우 보안 암호 즉시 교체를 선택하여 마스터 사용자 암호를 교체할 수 있습니다.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--manage-master-user-password	--no-manage-master-user-password` 및 `--master-user-secret-kms-key-id` 옵션을 설정합니다. 마스터 사용자 암호를 즉시 교체하려면 `--rotate-master-user-password` 옵션을 설정합니다.  RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `ManageMasterUserPassword` 및 `MasterUserSecretKmsKeyId` 파라미터를 설정합니다. 마스터 사용자 암호를 즉시 교	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
자세한 내용은 [Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리](#) 단원을 참조하십시오.	체하려면 `RotateMasterUserPassword` 파라미터를 `true`로 설정합니다.		

설정 및 설명	방법	범위	가동 중지 참고 사항
네트워크 유형 DB 클러스터에서 지원하는 IP 주소 지정 프로토콜. 리소스가 IPv4 주소 지정 프로토콜을 통해서만 DB 클러스터와 통신할 수 있도록 지정하는 IPv4. 리소스가 IPv4, IPv6 또는 모두를 통해 DB 클러스터와 통신할 수 있도록 지정하는 듀얼 스택 모드. IPv6 주소 지정 프로토콜을 통해 DB 클러스터와 통신해야 하는 리소스가 있는 경우 이중 스택 모드를 사용합니다. 듀얼 스택 모드를 사용하려면 두 개의 가용 영역에 걸쳐 IPv4 및 IPv6 네트워크 프로토콜을 모두 지원하는 서브넷이 두 개 이상 있어야 합니다. 또한 IPv6 CIDR 블록을 지정한 DB 서브넷 그룹의 서브넷과 연결해야 합니다. 자세한 내용은 Amazon Aurora IP 주	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--network-type` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `NetworkType` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
소 지정 단원을 참조하십시오.			
새 마스터 암호 마스터 사용자의 암호. • Aurora MySQL의 경우, 암호는 8~41자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. • Aurora PostgreSQL의 경우, 암호는 8~99자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. • /, ", @ 또는 공백을 포함할 수 없습니다.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--master-user-password` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `MasterUserPassword` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
성능 개선 도우미 데이터베이스 성능을 분석하고 문제를 해결할 수 있도록 DB 인스턴스 로드를 모니터링하는 도구인 성능 개선 도우미 활성화 여부. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--enable-performance-insights|--no-enable-performance-insights` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `EnablePerformanceInsights` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
성능 개선 도우미 AWS KMS key 성능 개선 도우미 데이터의 암호화를 위한 AWS KMS key 식별자입니다. KMS 키 식별자는 KMS 키의 Amazon 리소스 이름(ARN), 키 식별자 또는 키 별칭입니다. 자세한 내용은 Aurora 의 성능 개선 도우미 설정 및 해제 단원을 참조하십시오.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--performance-insights-kms-key-id` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `PerformanceInsightsKMSKeyId` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
성능 개선 도우미 보관 기간 성능 개선 도우미 데이터를 보관하는 시간(일). 프리 티어의 보존 설정은 기본값(7일)입니다. 성능 데이터를 더 오래 보존하려면 1~24개월을 지정하십시오. 보존 기간에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 단원을 참조하십시오. 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 단원을 참조하십시오.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--performance-insights-retention-period` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `PerformanceInsightsRetentionPeriod` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
프로모션 티어 기존 기본 인스턴스에 장애가 발생한 후 Aurora 복제본을 기본 인스턴스로 승격할 순서를 지정하는 값입니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 단원을 참조하십시오.	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--promotion-tier` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `PromotionTier` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
공개 액세스(Public access) DB 인스턴스에 퍼블릭 IP 주소를 부여하려면(즉 VPC 외부에서 액세스할 수 있음) 공개적으로 액세스할 수 있음(Publicly accessible)을 선택합니다. 공개적으로 액세스가 가능하려면 DB 인스턴스도 VPC의 퍼블릭 서브넷에 있어야 합니다. VPC 내부에서만 DB 인스턴스에 액세스할 수 있게 하려면 공개적으로 액세스할 수 없음(Not publicly accessible)을 선택합니다. 자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 섹션을 참조하세요. Amazon VPC 외부에서 DB 인스턴스에 연결하려면 DB 인스턴스에 공개적으로 액세스할 수 있어야 하고, DB 인스턴스 보안 그룹의 인바운드 규칙을 사용하여 액세스 권한을 부여해야 하며, 기타 요	AWS Management Console, DB 클러스터에서 DB 인스턴스 수정 사용. AWS CLI를 사용하여 `modify-db-instance` 를 실행하고 `--publicly-accessible|--no-publicly-accessible` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBInstance` 를 호출하고 `PubliclyAccessible` 파라미터를 설정합니다.	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
구 사항을 충족해야 합니다. 자세한 내용은 [Amazon RDS DB 인스턴스에 연결할 수 없음](#) 섹션을 참조하세요. DB 인스턴스에 공개적으로 액세스할 수 없는 경우 AWS Site-to-Site VPN 연결 또는 AWS Direct Connect 연결을 사용하여 프라이빗 네트워크에서 액세스할 수도 있습니다. 자세한 내용은 [인터네트워크 트래픽 개인 정보](#) 섹션을 참조하세요.			

설정 및 설명	방법	범위	가동 중지 참고 사항
Serverless v2 용량 설정 Aurora 용량 유닛(ACU)으로 측정한 Aurora Serverless v2 DB 클러스터의 데이터베이스 용량입니다. 자세한 내용은 클러스터의 Aurora Serverless v2 용량 설정 단원을 참조하십시오.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--serverless-v2-scaling-configuration` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `ServerlessV2ScalingConfiguration` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다. 변경 사항이 즉시 적용됩니다. 이 설정은 즉시 적용 설정을 무시합니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
보안 그룹 DB 클러스터와 연결할 보안 그룹. 자세한 내용은 보안 그룹을 통한 액세스 제어 섹션을 참조하세요.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--vpc-security-group-ids` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster` 를 호출하고 `VpcSecurityGroupIds` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
대상 역추적 기간 DB 클러스터를 역추적 가능한 시간(초). 이 설정은 Aurora MySQL에서만 사용 가능하며 역추적을 활성화한 상태에서 DB 클러스터가 생성된 경우에만 사용 가능합니다.	AWS Management Console, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 사용. AWS CLI를 사용하여 `modify-db-cluster`를 실행하고 `--backtrack-window` 옵션을 설정합니다. RDS API를 사용하여 `ModifyDBCluster`를 호출하고 `BacktrackWindow` 파라미터를 설정합니다.	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

Amazon Aurora DB 클러스터에 적용되지 않는 설정

다음과 같은 AWS CLI 명령 `modify-db-cluster` 및 RDS API 작업 `ModifyDBCluster`의 설정은 Amazon Aurora DB 클러스터에 적용되지 않습니다.

> **Note**
>
> Aurora DB 클러스터에 대한 이러한 설정을 수정하기 위해 AWS Management Console을 사용할 수 없습니다.

AWS CLI 설정	RDS API 설정
`--allocated-storage`	`AllocatedStorage`

AWS CLI 설정	RDS API 설정
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

DB 인스턴스용 Amazon Aurora에 적용되지 않는 설정

다음과 같은 AWS CLI 명령 [`modify-db-instance`] 및 RDS API 작업 [`ModifyDBInstance`]의 설정은 Amazon Aurora DB 인스턴스에 적용되지 않습니다.

> **Note**
>
> Aurora DB 인스턴스에 대한 이러한 설정을 수정하기 위해 AWS Management Console을 사용할 수 없습니다.

AWS CLI 설정	RDS API 설정
`--allocated-storage`	`AllocatedStorage`

AWS CLI 설정	RDS API 설정
--allow-major-version-upgrade|--no-allow-major-version-upgrade	AllowMajorVersionUpgrade
--copy-tags-to-snapshot|--no-copy-tags-to-snapshot	CopyTagsToSnapshot
--domain	Domain
--db-security-groups	DBSecurityGroups
--db-subnet-group-name	DBSubnetGroupName
--domain-iam-role-name	DomainIAMRoleName
--multi-az|--no-multi-az	MultiAZ
--iops	Iops
--license-model	LicenseModel
--network-type	NetworkType
--option-group-name	OptionGroupName
--processor-features	ProcessorFeatures
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features|--no-use-default-processor-features	UseDefaultProcessorFeatures

DB 클러스터에 Aurora 복제본 추가

복제를 사용하는 Aurora DB 클러스터에는 기본 DB 인스턴스 1개와 최대 15개의 Aurora 복제본이 있습니다. 기본 DB 인스턴스는 읽기 및 쓰기 작업을 지원하고, 클러스터 볼륨의 모든 데이터 수정 작업을 수행합니다. Aurora 복제본은 기본 DB 인스턴스와 동일한 스토리지 볼륨에 연결되지만 읽기 작업만 지원합니다. Aurora 복제본을 사용하여 기본 DB 인스턴스에서 읽기 워크로드를 오프로드합니다. 자세한 내용은 Aurora 복제본 단원을 참조하십시오.

Amazon Aurora 복제본에는 다음과 같은 제약이 포함되어 있습니다.

- Aurora Serverless v1 DB 클러스터용 Aurora 복제본은 생성할 수 없습니다. Aurora Serverless v1에는 모든 데이터베이스 읽기 및 쓰기 작업을 지원하기 위해 자동으로 확장 및 축소되는 단일 DB 인스턴스가 있습니다.

  그러나 Aurora Serverless v2 DB 클러스터에 리더 인스턴스를 추가할 수 있습니다. 자세한 내용은 Aurora Serverless v2 리더 추가 단원을 참조하십시오.

DB 클러스터의 가용성을 높이려면 DB 클러스터의 여러 가용 영역에 걸쳐 Aurora DB 클러스터의 기본 인스턴스 및 Aurora 복제본을 분배하는 것이 좋습니다. 자세한 내용은 리전 가용성 섹션을 참조하세요.

Aurora DB 클러스터에서 Aurora 복제본을 제거하려면 Aurora DB 클러스터에서 DB 인스턴스 삭제 의 다음 지침에 따라 Aurora 복제본을 삭제하십시오.

> **Note**
> 또한 Amazon Aurora은 RDS DB 인스턴스 등의 외부 데이터베이스 복제도 지원합니다. RDS DB 인스턴스는 Amazon Aurora와 같은 AWS 리전에 있어야 합니다. 자세한 내용은 Amazon Aurora를 사용한 복제 단원을 참조하십시오.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터에 Aurora 복제본을 추가할 수 있습니다.

콘솔

DB 클러스터에 Aurora 복제본을 추가하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 데이터베이스를 선택한 다음 새로운 DB 인스턴스를 추가할 DB 클러스터를 선택합니다.
3. 클러스터와 기본 인스턴스 모두 사용 가능 상태인지 확인합니다. DB 클러스터 또는 기본 인스턴스가 생성 중 같은 전환 상태인 경우에는 복제본을 추가할 수 없습니다.

   클러스터에 기본 인스턴스가 없는 경우 create-db-instance AWS CLI 명령을 사용하여 인스턴스를 생성합니다. 이 상황은 CLI를 사용하여 DB 클러스터 스냅샷을 복구한 다음 AWS Management Console에서 클러스터를 보는 경우에 발생할 수 있습니다.
4. 작업에서 Add reader(리더 추가)를 선택합니다.

   Add reader(리더 추가) 페이지가 나타납니다.
5. Add reader(리더 추가) 페이지에서 Aurora 복제본에 대한 옵션을 지정합니다. 다음 표에서는 Aurora 복제본 설정을 보여 줍니다.

옵션	조치
가용 영역	특정 가용 영역의 지정 여부를 결정합니다. 목록에는 DB 클러스터를 생성할 때 선택한 DB 서브넷 그룹에 매핑된 가용 영역만 포함됩니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 단원을 참조하십시오.
공개적으로 액세스할 수 있음(Public ly accessible)	Aurora 복제본에 퍼블릭 IP 주소를 할당하려면 Yes를 선택하고, 그렇지 않으면 No를 선택합니다. 모든 사용자의 액세스에서 Aurora 복제본을 숨기는 방법에 대한 자세한 정보는 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 단원을 참조하십시오.
암호화(Encryption)	이 Aurora 복제본에 대해 암호화를 활성화하려면 `Enable encryption` 을 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
DB 인스턴스 클래스	Aurora 복제본의 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.

옵션	조치
Aurora 복제본 소스	Aurora 복제본을 생성할 기본 인스턴스의 식별자를 선택합니다.
DB 인스턴스 식별자	선택한 AWS 리전에서 사용자의 계정에 대해 고유한 인스턴스의 이름을 입력합니다. 선택한 AWS 리전 및 DB 엔진을 포함(예: **aurora-read-instance1**)하는 등 이름에 지능적 요소를 추가할 수도 있습니다.
우선 순위	인스턴스의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스의 결함으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.
데이터베이스 포트	Aurora 복제본 포트는 DB 클러스터 포트와 동일합니다.
DB 파라미터 그룹	파라미터 그룹을 선택합니다. Aurora에 사용할 수 있는 기본 파라미터 그룹이 포함되어 있거나, 직접 파라미터 그룹을 생성할 수 있습니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.
성능 개선 도우미	Performance Insights 활성화(Turn on Performance Insights) 확인란은 기본적으로 선택되어 있습니다. 이 값은 라이터 인스턴스에서 상속되지 않습니다. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하십시오.
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.

옵션	조치
역할 모니터링	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. Amazon RDS가 Amazon CloudWatch Logs와 통신할 수 있도록 생성한 IAM 역할을 선택하거나 RDS가 `rds-monitoring-role` 라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 [Enhanced Monitoring을 사용하여 OS 지표 모니터링] 섹션을 참조하세요.
세부 수준	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
마이너 버전 자동 업그레이드	Aurora DB 클러스터를 활성화하여 DB 엔진의 마이너 버전 업그레이드를 자동으로 수신할 수 있도록 하려면 Enable auto minor version upgrade(자동 마이너 버전 업그레이드 활성화)를 선택하십시오. 마이너 버전 자동 업그레이드 설정은 Aurora PostgreSQL 및 Aurora MySQL DB 클러스터 모두에 적용됩니다. Aurora MySQL 2.x 클러스터의 경우 이 설정은 클러스터를 최대 버전 2.07.2로 업그레이드합니다. Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하세요. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

6. Add reader(리더 추가)를 선택하여 Aurora 복제본을 생성합니다.

AWS CLI

DB 클러스터에 Aurora 복제본을 생성하려면 create-db-instance AWS CLI 명령을 실행합니다. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션으로 포함하십시오. 다음 예제와 같이 `--`

`availability-zone` 파라미터를 사용하여 Aurora 복제본에 가용 영역을 선택적으로 지정할 수 있습니다.

예를 들어, 다음 명령을 사용하면 이름이 `sample-instance-us-west-2a`인 새 MySQL 5.7–호환 Aurora 복제본이 생성됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

다음 명령을 사용하면 이름이 `sample-instance-us-west-2a`인 새 MySQL 5.7 호환 Aurora 복제본이 생성됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

다음 명령을 사용하면 이름이 `sample-instance-us-west-2a`인 PostgreSQL 호환 Aurora 복제본이 생성됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

RDS API

DB 클러스터에 Aurora 복제본을 생성하려면 CreateDBInstance 작업을 호출하십시오. DB 클러스터의 이름을 `DBClusterIdentifier` 파라미터로 포함하십시오. 선택에 따라 `AvailabilityZone` 파라미터를 사용하여 Aurora 복제본의 가용 영역을 지정할 수 있습니다.

Aurora 복제본을 사용한 Amazon Aurora Auto Scaling에 대한 자세한 내용은 다음 섹션을 참조하세요.

주제

Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling

연결 및 워크로드 요구 사항을 충족하기 위해 Aurora Auto Scaling은 Aurora DB 클러스터에 대해 프로비저닝된 Aurora 복제본(리더 DB 인스턴스) 수를 동적으로 조정합니다. Aurora Auto Scaling은 Aurora MySQL 및 Aurora PostgreSQL에 모두 사용 가능합니다. Aurora Auto Scaling은 Aurora DB 클러스터를 활성화하여 연결 또는 워크로드의 갑작스러운 증가를 처리합니다. 연결 또는 워크로드가 감소하면 사용하지 않는 프로비저닝된 DB 인스턴스에 대해 요금을 지불하지 않도록 Aurora Auto Scaling이 불필요한 Aurora 복제본을 제거합니다.

고객은 조정 정책을 정의하고 Aurora DB 클러스터에 적용합니다. 조정 정책은 Aurora Auto Scaling에서 관리할 수 있는 최소 및 최대 Aurora 복제본 수를 정의합니다. 정책을 기반으로 Amazon CloudWatch 지표와 대상 값을 사용하여 결정된 실제 워크로드에 따라 Aurora Auto Scaling이 Aurora 복제본 수를 늘리거나 줄여 조정합니다.

> **Note**
>
> Aurora Auto Scaling은 라이터 DB 인스턴스의 워크로드에는 적용되지 않습니다. Aurora Auto Scaling은 리더 인스턴스의 워크로드만 처리합니다.

AWS Management Console을 사용하여 미리 정의된 지표를 기반으로 조정 정책을 적용할 수 있습니다. 또는 미리 정의된 지표나 사용자 지정 지표를 기반으로 AWS CLI 또는 Aurora Auto Scaling API를 사용하여 크기 조정 정책을 적용할 수도 있습니다.

주제

- 시작하기 전에
- Aurora Auto Scaling 정책
- DB 인스턴스 ID 및 태그 지정
- Aurora Auto Scaling 및 성능 개선 도우미

시작하기 전에

Aurora Auto Scaling을 Aurora DB 클러스터에 사용하려면 먼저 기본(라이터) DB 인스턴스가 있는 Aurora DB 클러스터를 생성해야 합니다. Aurora DB 클러스터 생성에 대한 자세한 정보는 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

DB 클러스터가 복제본이 사용 가능한 상태일 경우에만 Aurora Auto Scaling이 DB 클러스터의 크기를 조정합니다.

Aurora Auto Scaling이 새로운 Aurora 복제본을 추가할 때 새로운 Aurora 복제본은 기본 인스턴스에 사용되는 것과 동일한 DB 인스턴스 클래스입니다. DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하십시오. 또한 새 Aurora 복제본을 위한 승격 티어는 우선 순위가 마지막인 기본값 15로 설정됩니다. 즉 장애 조치가 이루어지는 동안 수동으로 생성된 것과 같이 우선 순위가 더 높은 복제본이 먼저 승격됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.

Aurora Auto Scaling은 자체에서 생성한 Aurora 복제본만 제거합니다.

Aurora Auto Scaling의 이점을 활용하려면 애플리케이션에서 새로운 Aurora 복제본과의 연결을 지원해야 합니다. 이렇게 하려면 Aurora 리더 엔드포인트를 사용하는 것이 좋습니다. AWS JDBC 드라이버와 같은 드라이버를 사용할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하십시오.

> **Note**
>
> Aurora 글로벌 데이터베이스는 현재 세컨더리 데이터베이스 클러스터에 대해 Aurora Auto Scaling을 지원하지 않습니다.

Aurora Auto Scaling 정책

Aurora Auto Scaling에서는 조정 정책을 사용하여 Aurora DB 클러스터의 Aurora 복제본 수를 조정합니다. Aurora Auto Scaling의 구성 요소는 다음과 같습니다.

- 서비스 연결 역할
- 대상 지표
- 최소 및 최대 용량
- 휴지 기간

주제

- Auto Scaling 정책 추가, 편집 또는 삭제

서비스 연결 역할

Aurora Auto Scaling은 `AWSServiceRoleForApplicationAutoScaling_RDSCluster` 서비스 연결 역할을 사용합니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 Application Auto Scaling 서비스 연결 역할을 참조하십시오.

대상 지표

이 유형의 정책에서는 미리 정의된 지표나 사용자 지정 지표 및 지표의 대상 값이 대상 추적 조정 정책 구성에 지정됩니다. Aurora Auto Scaling은 조정 정책을 트리거하는 CloudWatch 경보를 생성 및 관리하고 지표와 대상 값을 기준으로 조정 조절을 계산합니다. 조정 정책은 필요에 따라 Aurora 복제본을 추가하거나 제거하여 지표를 지정한 대상 값으로 또는 대상 값에 가깝게 유지합니다. 대상 추적 조정 정책은 지표를 대상 값에 가깝게 유지하는 것 외에도 워크로드 변화로 인한 지표의 변동에 따라 조정되기도 합니다. 이 정책은 DB 클러스터의 사용 가능한 Aurora 복제본 수의 급격한 변동을 최소하기도 합니다.

미리 정의된 평균 CPU 사용률 지표가 사용되는 조정 정책을 예로 든다면, 그러한 정책이 CPU 사용률을 40%의 지정된 사용률(퍼센트)로 또는 그에 가깝게 유지할 수 있습니다.

> **Note**
>
> Aurora DB 클러스터마다 대상 지표에 대해 Auto Scaling 정책을 하나씩만 생성할 수 있습니다.

Aurora Auto Scaling을 구성하면 대상 지표 값이 클러스터에 있는 모든 리더 인스턴스의 평균으로 계산됩니다. 이 계산은 다음과 같이 이루어집니다.

- Auto Scaling에서 관리되든 수동으로 추가되든 관계없이 Aurora 클러스터의 모든 리더 인스턴스를 포함합니다.
- 사용자 지정 엔드포인트와 연결된 인스턴스를 포함합니다. 사용자 지정 엔드포인트는 대상 지표 계산에 영향을 주지 않습니다.
- 클러스터의 라이터 인스턴스는 포함하지 않습니다.

이 지표는 다음 차원을 사용하여 CloudWatch에서 파생됩니다.

- `DBClusterIdentifier`

- `Role=READER`

예를 들어 다음 설정을 사용하는 Aurora MySQL 클러스터를 생각해 보세요.

- 수동 인스턴스(Auto Scaling으로 제어되지 않음):
  - CPU 사용률이 50%인 라이터
  - CPU 사용률이 90%인 리더 1(사용자 지정 엔드포인트: `custom-reader-1`)
  - CPU 사용률이 90%인 리더 2(사용자 지정 엔드포인트: `custom-reader-2`)
- Auto Scaling 인스턴스:
  - CPU 사용률이 10%인 리더 3(Auto Scaling을 사용하여 추가됨)

이 시나리오에서는 Auto Scaling 정책의 대상 지표가 다음과 같이 계산됩니다.

```
Target metric = (CPU utilization of reader 1 + reader 2 + reader 3) / total number of
 readers

Target metric = (90 + 90 + 10) / 3 = 63.33%
```

Auto Scaling 정책은 이 값을 사용하여 정의된 임계값에 따라 스케일 인할지 스케일 아웃할지를 평가합니다.

다음을 고려하세요.

- 사용자 지정 엔드포인트는 트래픽이 특정 리더로 라우팅되는 방법을 결정하지만 지표 계산에서 리더를 제외하지는 않습니다.
- 수동 인스턴스는 항상 대상 지표 계산에 포함됩니다.
- 예기치 않은 규모 조정 동작을 방지하려면 Auto Scaling 구성이 클러스터의 모든 리더 인스턴스를 고려하는지 확인합니다.
- 클러스터에 리더가 없는 경우 지표가 계산되지 않으며 Auto Scaling 정책 경보는 비활성 상태로 유지됩니다. Auto Scaling 정책이 효과적으로 작동하려면 항상 리더가 하나 이상 있어야 합니다.

최소 및 최대 용량

Application Auto Scaling에서 관리할 최대 Aurora 복제본 수를 지정할 수 있습니다. 이 값은 0–15로 설정되어야 하며 최소 Aurora 복제본 수에 대해 지정된 값과 같거나 커야 합니다.

Application Auto Scaling에서 관리할 최소 Aurora 복제본 수를 지정할 수도 있습니다. 이 값은 0–15로 설정되어야 하며 최대 Aurora 복제본 수에 대해 지정된 값과 같거나 작아야 합니다.

Aurora Auto Scaling이 작동하려면 리더 DB 인스턴스가 하나 이상 있어야 합니다. DB 클러스터에 리더 인스턴스가 없고 최소 용량을 0으로 설정하면 Aurora Auto Scaling이 작동하지 않습니다.

> **Note**
>
> Aurora DB 클러스터에 대해 최소 및 최대 용량이 설정됩니다. 지정된 값은 해당 Aurora DB 클러스터와 연관된 모든 정책에 적용됩니다.

휴지 기간

Aurora DB 클러스터의 축소 및 확장에 영향을 미치는 휴지 기간을 추가하여 대상 추적 조정 정책의 응답성을 조정할 수 있습니다. 휴지 기간은 기간이 만료될 때까지 후속 스케일 인 또는 스케일 아웃 요청을 차단합니다. 이 차단으로 인해 축소 요청에 대한 Aurora DB 클러스터의 Aurora 복제본 차단과 확장 요청에 대한 Aurora 복제본 생성 속도가 느려집니다.

다음과 같은 휴지 기간을 지정할 수 있습니다.

- 축소 활동은 Aurora DB 클러스터에 있는 Aurora 복제본 수를 줄입니다. 스케일 인 휴지 기간은 스케일 인 활동이 완료되고 다른 스케일 인 활동이 시작되기 전의 시간을 초 단위로 지정합니다.
- 확장 활동은 Aurora DB 클러스터에 있는 Aurora 복제본 수를 늘립니다. 스케일 아웃 휴지 기간은 스케일 아웃 활동이 완료되고 다른 스케일 아웃 활동이 시작되기 전의 시간을 초 단위로 지정합니다.

> **Note**
>
> 후속 스케일 아웃 요청이 첫 번째 요청보다 많은 Aurora Replica에 대한 요청인 경우 스케일 아웃 휴지 기간은 무시됩니다.

스케일 인 또는 스케일 아웃 휴지 기간을 지정하지 않은 경우 기본값은 각각 300초입니다.

스케일 인 활동 활성화 또는 비활성화

정책의 스케일 인 활동을 활성화하거나 비활성화할 수 있습니다. 축소 활동을 활성화하면 조정 정책을 통해 Aurora 복제본을 삭제할 수 있습니다. 스케일 인 활동이 활성화되면 조정 정책의 스케일 인 휴지 기간이 스케일 인 활동에 적용됩니다. 축소 활동을 비활성화하면 스케일 정책을 통해 Aurora 복제본을 삭제할 수 없습니다.

> **Note**
> 조정 정책이 필요에 따라 Aurora 복제본을 생성할 수 있도록 확장 활동이 항상 활성화됩니다.

Auto Scaling 정책 추가, 편집 또는 삭제

AWS Management Console, AWS CLI 또는 Application Auto Scaling API를 사용하여 Auto Scaling 정책을 추가, 편집 또는 삭제할 수 있습니다. Auto Scaling 정책을 추가, 편집 또는 삭제하는 방법에 대한 자세한 내용은 다음 섹션을 참조하세요.

- Amazon Aurora DB 클러스터에 Auto Scaling 정책 추가
- Amazon Aurora DB 클러스터의 Auto Scaling 정책 편집
- Amazon Aurora DB 클러스터에서 Auto Scaling 정책 삭제

DB 인스턴스 ID 및 태그 지정

Aurora Auto Scaling으로 복제본이 추가되면 DB 인스턴스 ID에 `application-autoscaling-` 접두어가 붙습니다 (예: `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`).

다음 태그는 DB 인스턴스에 자동으로 추가됩니다. DB 인스턴스 세부 정보 페이지의 Tags 탭에서 확인할 수 있습니다.

Tag	값
application-autoscaling:resourceId	cluster:mynewcluster-cluster

Amazon RDS 리소스 태그에 관한 자세한 내용은 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 단원을 참조하십시오.

Aurora Auto Scaling 및 성능 개선 도우미

성능 개선 도우미를 사용하면 다른 Aurora 리더 DB 인스턴스와 마찬가지로 Aurora Auto Scaling에서 추가한 복제본을 모니터링할 수 있습니다.

성능 개선 도우미를 사용하여 Aurora DB 클러스터를 모니터링하는 방법에 대한 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하세요.

Amazon Aurora DB 클러스터에 Auto Scaling 정책 추가

AWS Management Console, AWS CLI 또는 Application Auto Scaling API를 사용하여 크기 조정 정책을 추가할 수 있습니다.

> **Note**
>
> AWS CloudFormation을 사용하여 크기 조정 정책을 추가하는 예는 AWS CloudFormation 사용 설명서의 Aurora DB 클러스터의 크기 조정 정책 선언을 참조하세요.

콘솔

AWS Management Console을 사용하여 크기 조정 정책을 Aurora DB 클러스터에 추가할 수 있습니다.

Aurora DB 클러스터에 Auto Scaling 정책을 추가하려면

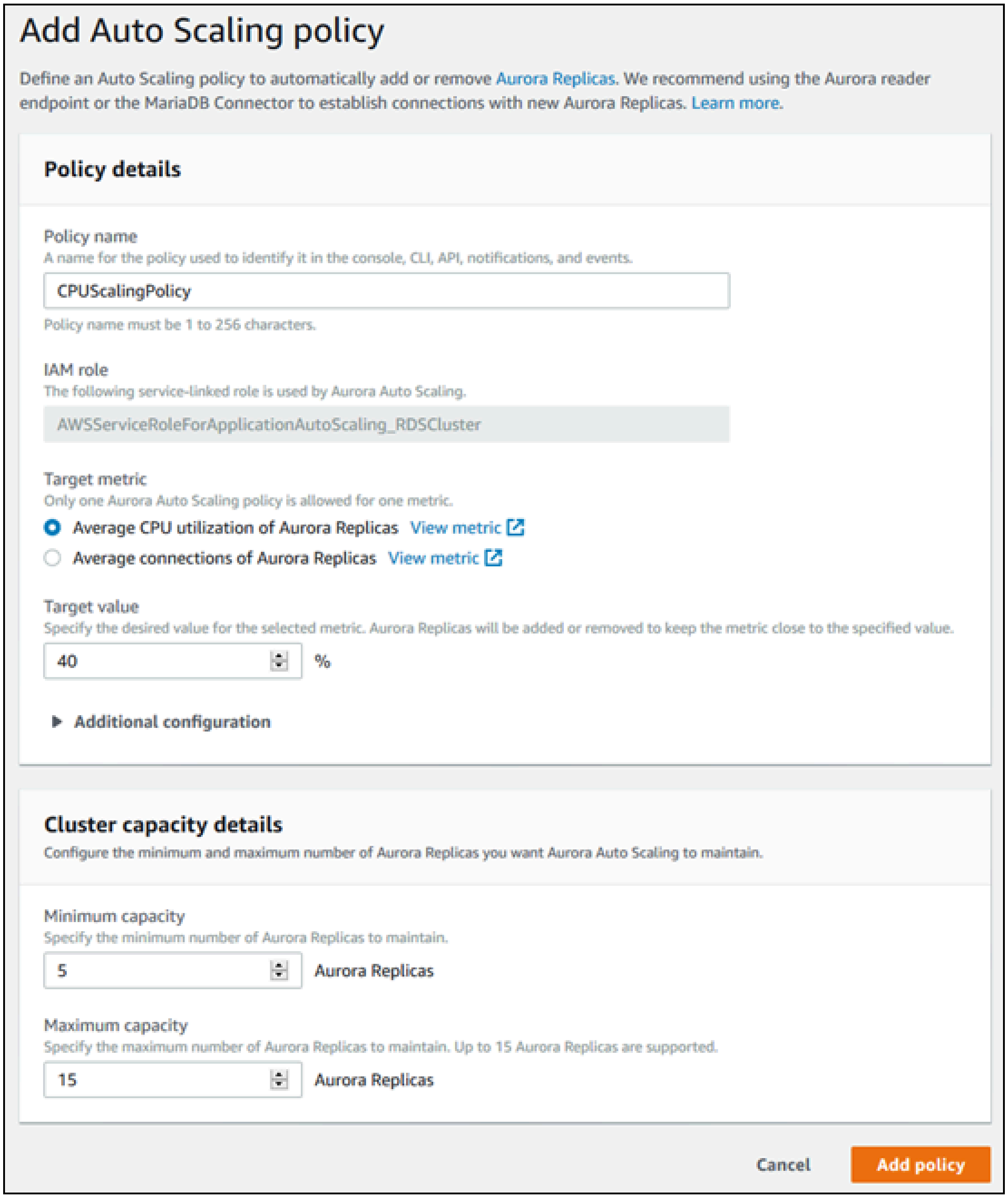
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 정책을 추가할 Aurora DB 클러스터를 선택하십시오.
4. 로그 및 이벤트 탭을 선택합니다.
5. Auto scaling policies(Auto Scaling 정책) 섹션에서 추가를 선택합니다.

   [Add Auto Scaling policy] 대화 상자가 나타납니다.
6. Policy Name(정책 이름)에서 정책의 이름을 입력합니다.
7. 대상 지표로 다음 중 하나를 선택합니다.
   - 평균 CPU 사용률을 기반으로 정책을 생성하려면 Aurora 복제본의 평균 CPU 사용률.
   - Aurora 복제본에 대한 평균 연결 수를 기반으로 정책을 생성하려면 Aurora 복제본의 평균 연결 수.
8. 대상 값으로 다음 중 하나를 입력합니다.
   - 이전 단계에서 Aurora 복제본의 평균 CPU 사용률을 선택한 경우 Aurora 복제본에 유지하려는 CPU 사용률(퍼센트)을 입력하십시오.
   - 이전 단계에서 Aurora 복제본의 평균 연결을 선택한 경우 유지하려는 연결 수를 입력하십시오.

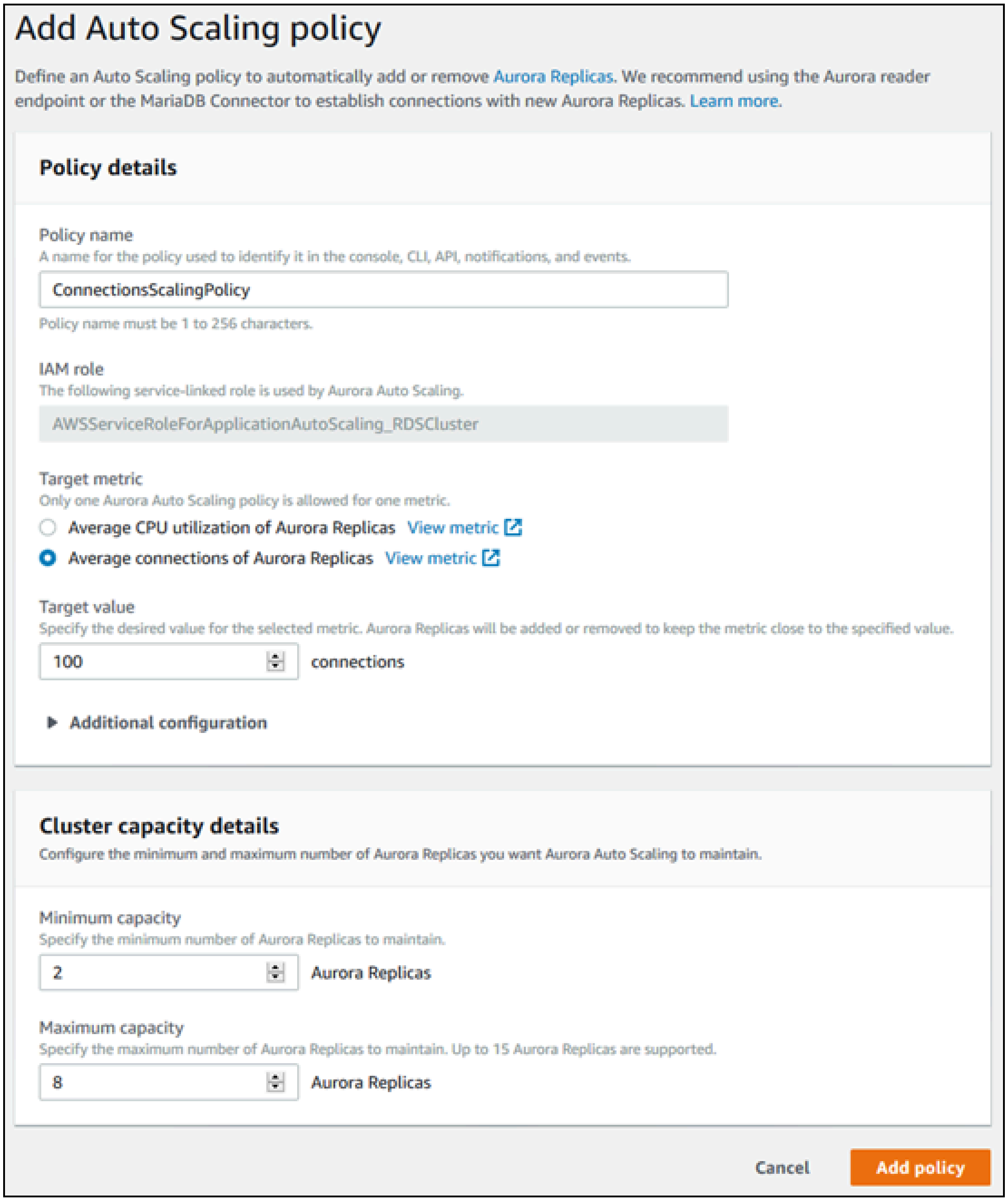
Aurora 복제본이 추가되거나 제거되어 지정한 값에 가깝게 지표가 유지됩니다.

9. (선택 사항) 추가 구성을 열어 스케일 인 또는 스케일 아웃 휴지 기간을 생성합니다.
10. 최소 용량에 Aurora Auto Scaling 정책에 따라 유지해야 할 최소 Aurora 복제본 수를 입력하십시오.
11. 최대 용량에 Aurora Auto Scaling 정책에 따라 유지해야 할 최대 Aurora 복제본 수를 입력하십시오.
12. [Add policy]를 선택합니다.

다음 대화 상자에서 평균 CPU 사용률 40%를 기반으로 Auto Scaling 정책을 생성합니다. 정책은 최소 5개의 Aurora 복제본과 최대 15개의 Aurora 복제본을 지정합니다.



다음 대화 상자는 평균 연결 수 100을 기반으로 Auto Scaling 정책을 생성합니다. 정책은 최소 2개의 Aurora 복제본과 최대 8개의 Aurora 복제본을 지정합니다.



AWS CLI 또는 Application Auto Scaling API

미리 정의된 지표나 사용자 지정 지표를 기반으로 조정 정책을 적용할 수 있습니다. 이를 위해 AWS CLI 또는 Application Auto Scaling API를 사용할 수 있습니다. 첫 단계는 Application Auto Scaling 사용하여 Aurora DB 클러스터를 등록해야 합니다.

Aurora DB 클러스터 등록

Aurora DB 클러스터에 Aurora Auto Scaling을 사용하려면 먼저 Application Auto Scaling을 사용하여 Aurora DB 클러스터를 등록합니다. 그렇게 하려면 해당 클러스터에 적용할 크기 조정 차원 및 한계를 정의합니다. Application Auto Scaling은 Aurora 복제본 수를 나타내는 `rds:cluster:ReadReplicaCount` 크기 조정 가능한 차원을 따라 Aurora DB 클러스터를 동적으로 크기 조정합니다.

Aurora DB 클러스터를 등록하려면 AWS CLI 또는 Application Auto Scaling API를 사용할 수 있습니다.

AWS CLI

Aurora DB 클러스터를 등록하려면 다음 파라미터와 함께 [`register-scalable-target`] AWS CLI 명령을 사용하세요.

- `--service-namespace` 이 값을 로 설정하십시오.`rds`
- `--resource-id` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `--scalable-dimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`
- `--min-capacity` – Application Auto Scaling에서 관리하는 최소 리더 DB 인스턴스 수 `--min-capacity`, `--max-capacity` 및 클러스터의 DB 인스턴스 수 간의 관계에 대한 자세한 내용은 [최소 및 최대 용량] 단원을 참조하십시오.
- `--max-capacity` – Application Auto Scaling에서 관리하는 최소 리더 DB 인스턴스 수 `--min-capacity`, `--max-capacity` 및 클러스터의 DB 인스턴스 수 간의 관계에 대한 자세한 내용은 [최소 및 최대 용량] 단원을 참조하십시오.

Example

다음 예제에서는 이름이 `myscalablecluster`인 Aurora DB 클러스터를 등록합니다. 등록은 1개에서 8개까지 Aurora 복제본을 포함하도록 DB 클러스터 크기를 동적으로 조정해야 함을 나타냅니다.

대상 LinuxmacOS, 또는Unix:

```
aws application-autoscaling register-scalable-target \
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
```

```
    --max-capacity 8 \
```

Windows의 경우:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

Application Auto Scaling API

Application Auto Scaling로 Aurora DB 클러스터를 등록하려면 다음 파라미터와 함께 `RegisterScalableTarget` Application Auto Scaling API 작업을 사용하십시오.

- `ServiceNamespace` 이 값을 로 설정하십시오.`rds`
- `ResourceID` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `ScalableDimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`
- `MinCapacity` – Application Auto Scaling에서 관리하는 최소 리더 DB 인스턴스 수 `MinCapacity`, `MaxCapacity` 및 클러스터의 DB 인스턴스 수 간의 관계에 대한 자세한 내용은 최소 및 최대 용량 단원을 참조하십시오.
- `MaxCapacity` – Application Auto Scaling에서 관리하는 최소 리더 DB 인스턴스 수 `MinCapacity`, `MaxCapacity` 및 클러스터의 DB 인스턴스 수 간의 관계에 대한 자세한 내용은 최소 및 최대 용량 단원을 참조하십시오.

Example

다음 예제에서는 Application Auto Scaling API를 사용하여 이름이 `myscalablecluster`인 Aurora DB 클러스터를 등록합니다. 이 등록은 1개에서 8개까지 Aurora 복제본을 포함하도록 DB 클러스터 크기를 동적으로 조정해야 함을 나타냅니다.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
```

```
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Aurora DB 클러스터에 대한 조정 정책 정의

대상 추적 조정 정책 구성은 지표와 대상 값이 정의되어 있는 JSON 블록으로 나타냅니다. 텍스트 파일에 JSON 블록으로 조정 정책 구성을 저장할 수 있습니다. AWS CLI 또는 Application Auto Scaling API를 호출할 때 이 텍스트 파일을 사용합니다. 정책 구성 구문에 대한 자세한 정보는 Application Auto Scaling API 참조의 `TargetTrackingScalingPolicyConfiguration`을 참조하세요.

대상 추적 조정 정책 구성을 정의하기 위해 다음과 같은 옵션을 사용할 수 있습니다.

주제

미리 정의된 지표 사용

미리 정의된 지표를 사용하여 Aurora Auto Scaling에서 대상 추적과 동적 조정 둘 다에 원활하게 사용할 수 있는 대상 추적 조정 정책을 Aurora DB 클러스터에 대해 신속하게 정의할 수 있습니다.

현재 Aurora은 Aurora Auto Scaling에서 다음과 같이 미리 정의된 지표를 지원합니다.

- RDSReaderAverageCPUUtilization – Aurora DB 클러스터에 있는 모든 Aurora 복제본에서 CloudWatch에 있는 `CPUUtilization` 지표의 평균 값.

- RDSReaderAverageDatabaseConnections – Aurora DB 클러스터에 있는 모든 Aurora 복제본에서 CloudWatch에 있는 `DatabaseConnections` 지표의 평균 값.

`CPUUtilization` 및 `DatabaseConnections` 지표에 대한 자세한 정보는 [Amazon Aurora에 대한 Amazon CloudWatch 지표](#) 단원을 참조하십시오.

조정 정책에서 미리 정의된 지표를 사용하려면 조정 정책을 위한 대상 추적 구성을 생성합니다. 미리 정의된 지표의 `PredefinedMetricSpecification` 및 해당 지표의 대상 값에 대한 `TargetValue`를 이 구성에 포함해야 합니다.

Example

다음 예제에서는 Aurora DB 클러스터의 대상 추적 조정을 위한 일반적인 정책 구성을 설명합니다. 이 구성에서 `RDSReaderAverageCPUUtilization` 미리 정의된 지표는 모든 Aurora 복제본에 대해 평균 CPU 사용률 40%를 기반으로 Aurora DB 클러스터를 조정합니다.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

사용자 지정 지표 사용

사용자 지정 지표를 사용하여 사용자 지정 요구 사항에 맞는 대상 추적 조정 정책을 정의할 수 있습니다. 조정에 따라 변경되는 모든 Aurora 지표를 기반으로 사용자 지정 지표를 정의할 수 있습니다.

모든 Aurora 지표를 대상 추적에 사용할 수 있는 것은 아닙니다. 측정치는 유효한 사용량 수치로서 인스턴스의 사용량을 설명해야 합니다. Aurora DB 클러스터에 있는 Aurora 복제본 수에 따라 지표 값이 증가하거나 줄어들어야 합니다. 지표 데이터를 사용하여 Aurora 복제본 수를 비례적으로 확장 또는 축소하려면 이 비례적인 증가나 감소가 필요합니다.

Example

다음 예제에서는 조정 정책의 대상 추적 구성을 설명합니다. 이 구성에서 사용자 지정 지표는 `my-db-cluster`라는 Aurora DB 클러스터의 모든 Aurora 복제본에 대해 평균 CPU 사용률 50%를 기반으로 Aurora DB 클러스터를 조정합니다.

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

휴지 기간 사용

`ScaleOutCooldown`에 초 단위로 값을 지정하여 Aurora DB 클러스터를 확장하기 위한 휴지 기간을 추가할 수 있습니다. 마찬가지로 `ScaleInCooldown`에 초 단위로 값을 추가하여 Aurora DB 클러스터를 축소하기 위한 휴지 기간을 추가할 수 있습니다. `ScaleInCooldown` 및 `ScaleOutCooldown`에 대한 자세한 정보는 Application Auto Scaling API 참조의 [`TargetTrackingScalingPolicyConfiguration`]을 참조하십시오.

Example

다음 예제에서는 조정 정책의 대상 추적 구성을 설명합니다. 이 구성에서 `RDSReaderAverageCPUUtilization` 사전 정의 지표는 해당 Aurora DB 클러스터에 있는 모든 Aurora 복제본에 대해 평균 CPU 사용률 40%를 기반으로 Aurora DB 클러스터를 조정합니다. 구성에서는 스케일 인 휴지 기간 10분과 스케일 아웃 휴지 기간 5분을 제공합니다.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

스케일 인 활동 비활성화

축소 활동을 비활성화하여 대상 추적 조정 정책 구성에서 Aurora DB 클러스터를 축소하지 않도록 할 수 있습니다. 축소 활동을 비활성화하면 조정 정책에서 필요에 따라 Aurora 복제본을 생성할 수 있지만 삭제할 수는 없습니다.

`DisableScaleIn`에 부울 값을 지정하여 Aurora DB 클러스터에 대한 축소 활동을 활성화하거나 비활성화할 수 있습니다. `DisableScaleIn`에 대한 자세한 정보는 Application Auto Scaling API 참조의 [`TargetTrackingScalingPolicyConfiguration`]을 참조하십시오.

Example

다음 예제에서는 조정 정책의 대상 추적 구성을 설명합니다. 이 구성에서 `RDSReaderAverageCPUUtilization` 미리 정의된 지표는 해당 Aurora DB 클러스터에 있는 모든 Aurora 복제본에 대해 평균 CPU 사용률 40%를 기반으로 Aurora DB 클러스터를 조정합니다. 구성에서는 조정 정책의 스케일 인 활동을 비활성화합니다.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

Aurora DB 클러스터에 조정 정책 적용

Application Auto Scaling으로 Aurora DB 클러스터를 등록하고 조정 정책을 삭제한 후 등록된 Aurora DB 클러스터에 조정 정책을 적용합니다. 크기 조정 정책을 Aurora DB 클러스터에 적용하려면 AWS CLI 또는 Application Auto Scaling API를 사용할 수 있습니다.

AWS CLI

크기 조정 정책을 Aurora DB 클러스터에 적용하려면 [`put-scaling-policy`]AWS CLI 명령을 다음 파라미터와 함께 사용합니다.

- `--policy-name` – 조정 정책의 이름입니다.
- `--policy-type` 이 값을 로 설정하십시오.`TargetTrackingScaling`

- `--resource-id` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `--service-namespace` 이 값을 로 설정하십시오.`rds`
- `--scalable-dimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`
- `--target-tracking-scaling-policy-configuration` – Aurora DB 클러스터에 사용할 대상 추적 조정 정책 구성입니다.

Example

다음 예제에서는 Application Auto Scaling를 사용하여 `myscalablepolicy`라는 대상 추적 조정 정책을 `myscalablecluster`라는 Aurora DB 클러스터에 적용합니다. 이를 위해 `config.json`이라는 파일에 저장된 정책 구성을 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Windows의 경우:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

Application Auto Scaling API

Application Auto Scaling API를 사용하여 Aurora DB 클러스터에 조정 정책을 적용하려면 다음 파라미터와 함께 `PutScalingPolicy` Application Auto Scaling API 작업을 사용하십시오.

- `PolicyName` – 조정 정책의 이름입니다.
- `ServiceNamespace` 이 값을 로 설정하십시오.`rds`
- `ResourceID` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `ScalableDimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`
- `PolicyType` 이 값을 로 설정하십시오.`TargetTrackingScaling`
- `TargetTrackingScalingPolicyConfiguration` – Aurora DB 클러스터에 사용할 대상 추적 조정 정책 구성입니다.

Example

다음 예제에서는 Application Auto Scaling를 사용하여 `myscalablepolicy`라는 대상 추적 조정 정책을 `myscalablecluster`라는 Aurora DB 클러스터에 적용합니다. `RDSReaderAverageCPUUtilization` 사전 정의 지표를 기반으로 하는 정책 구성을 사용합니다.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
```

```
    }
}
```

Amazon Aurora DB 클러스터의 Auto Scaling 정책 편집

AWS Management Console, AWS CLI 또는 Application Auto Scaling API를 사용하여 크기 조정 정책을 편집할 수 있습니다.

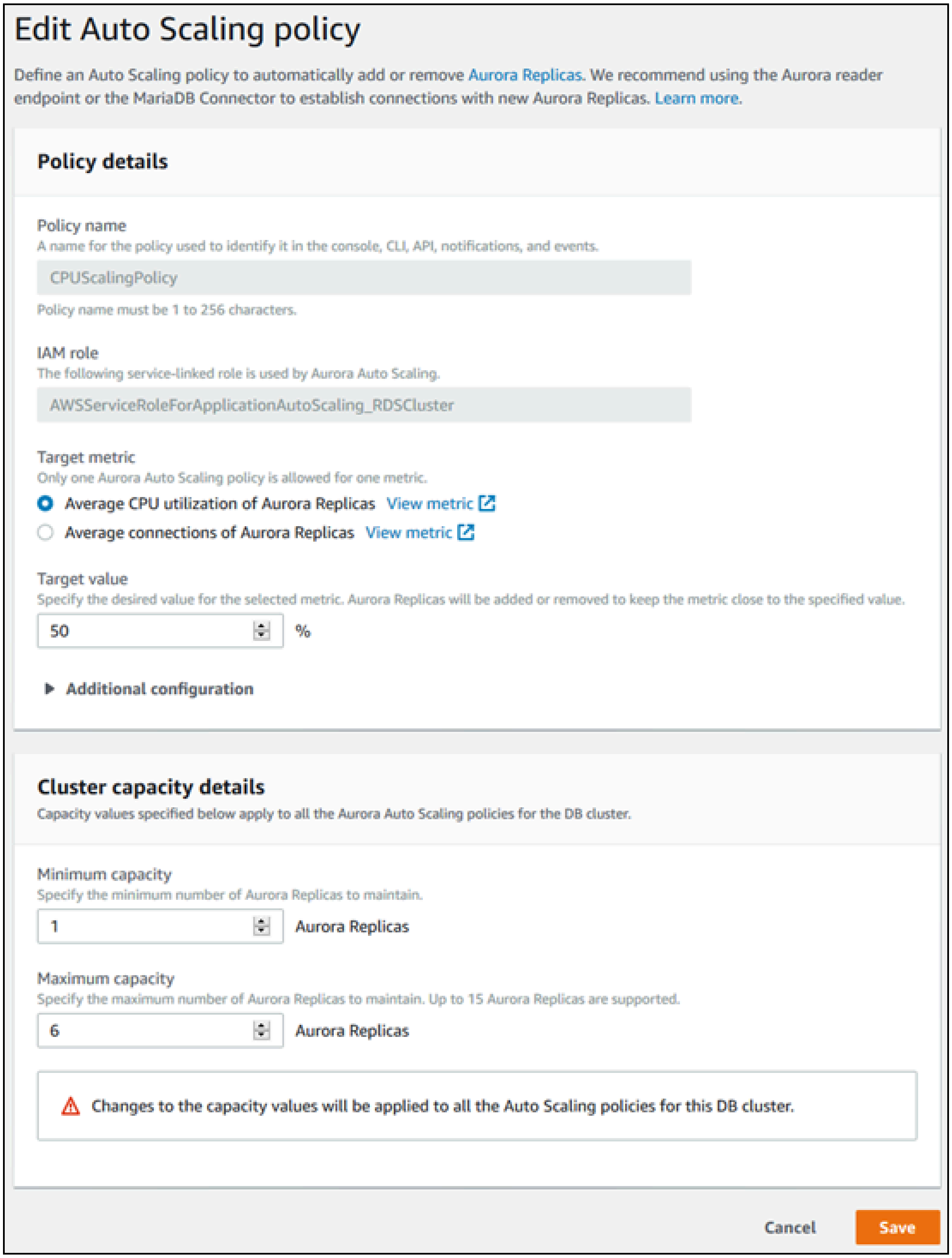
콘솔

AWS Management Console을 사용하여 조정 정책을 편집할 수 있습니다.

Aurora DB 클러스터의 Auto Scaling 정책을 편집하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Auto Scaling 정책을 편집할 Aurora DB 클러스터를 선택합니다.
4. 로그 및 이벤트 탭을 선택합니다.
5. Auto scaling policies(Auto Scaling 정책) 섹션에서 Auto Scaling 정책을 선택한 후 편집을 선택합니다.
6. 정책을 변경합니다.
7. 저장을 선택합니다.

[Edit Auto Scaling policy] 대화 상자 샘플은 다음과 같습니다.



AWS CLI 또는 Application Auto Scaling API

크기 조정 정책을 적용하는 것과 같은 방식으로 AWS CLI 또는 Application Auto Scaling API를 사용하여 크기 조정 정책을 편집할 수 있습니다.

- AWS CLI를 사용할 때는 `--policy-name` 파라미터에서 편집할 정책의 이름을 지정하세요. 변경할 파라미터의 새로운 값을 지정합니다.
- Application Auto Scaling API를 사용할 때는 `PolicyName` 파라미터에서 편집하려는 정책의 이름을 지정하세요. 변경할 파라미터의 새로운 값을 지정합니다.

자세한 내용은 Aurora DB 클러스터에 조정 정책 적용 단원을 참조하십시오.

Amazon Aurora DB 클러스터에서 Auto Scaling 정책 삭제

AWS Management Console, AWS CLI 또는 Application Auto Scaling API를 사용하여 크기 조정 정책을 삭제할 수 있습니다.

콘솔

AWS Management Console을 사용하여 조정 정책을 삭제할 수 있습니다.

Aurora DB 클러스터의 Auto Scaling 정책을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Auto Scaling 정책을 삭제할 Aurora DB 클러스터를 선택합니다.
4. 로그 및 이벤트 탭을 선택합니다.
5. Auto scaling policies(Auto Scaling 정책) 섹션에서 Auto Scaling 정책을 선택한 후 삭제를 선택합니다.

AWS CLI

Aurora DB 클러스터에서 크기 조정 정책을 삭제하려면 다음 파라미터와 함께 `delete-scaling-policy` AWS CLI 명령을 사용하세요.

- `--policy-name` – 조정 정책의 이름입니다.
- `--resource-id` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `--service-namespace` 이 값을 로 설정하십시오.`rds`

- `--scalable-dimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`

Example

다음 예제에서는 `myscalablepolicy`라는 대상 추적 조정 정책을 `myscalablecluster`라는 Aurora DB 클러스터에서 삭제합니다.

대상 LinuxmacOS, 또는Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

Windows의 경우:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

Application Auto Scaling API

Aurora DB 클러스터에서 조정 정책을 삭제하려면 다음 파라미터와 함께 `DeleteScalingPolicy` Application Auto Scaling API 작업을 사용하십시오.

- `PolicyName` – 조정 정책의 이름입니다.
- `ServiceNamespace` 이 값을 로 설정하십시오.`rds`
- `ResourceID` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `ScalableDimension` 이 값을 로 설정하십시오.`rds:cluster:ReadReplicaCount`

Example

다음 예제에서는 Application Auto Scaling API를 사용하여 `myscalablepolicy`라는 대상 추적 조정 정책을 `myscalablecluster`라는 Aurora DB 클러스터에서 삭제합니다.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

Aurora DB 클러스터의 성능 및 확장 관리

다음 옵션을 사용해 Aurora DB 클러스터 및 DB 인스턴스의 성능 및 확장을 관리할 수 있습니다.

주제

스토리지 조정

Aurora 스토리지는 클러스터 볼륨에 저장된 데이터에 따라 자동 조정됩니다. 데이터 증가에 따라 클러스터 볼륨 스토리지는 최대 128테비바이트(TiB) 또는 64TiB까지 확장됩니다. 최대 크기는 DB 엔진 버전에 따라 다릅니다. 클러스터 볼륨에 포함되는 데이터 종류를 알아보려면 Amazon Aurora 스토리지 단원을 참조하세요. 특정 버전의 최대 크기에 대한 자세한 내용은 Amazon Aurora 크기 제한 섹션을 참조하세요.

스토리지 비용을 결정하기 위해 클러스터 볼륨 크기가 한 시간 단위로 평가됩니다. 요금에 대한 자세한 내용은 Aurora 요금 페이지를 참조하십시오.

Aurora 클러스터 볼륨의 크기는 수테비바이트까지 확장할 수 있지만 요금은 볼륨에서 사용한 공간에 대해서만 청구됩니다. 요금이 청구되는 스토리지 공간을 결정하는 메커니즘은 Aurora 클러스터 버전에 따라 다릅니다.

- 클러스터 볼륨에서 Aurora 데이터가 제거되면 비슷한 양만큼 요금이 청구되는 전체 공간이 감소합니다. 이 동적 크기 조정 동작은 기본 테이블스페이스가 삭제되거나 공간이 더 적게 필요하도록 재구성할 때 발생합니다. 따라서 더 이상 필요하지 않은 테이블과 데이터베이스를 삭제하여 스토리지 비용을 줄일 수 있습니다. 동적 크기 조정은 특정 Aurora 버전에 적용됩니다. 다음은 데이터를 제거할 때 클러스터 볼륨의 크기가 동적으로 조정되는 Aurora 버전입니다.

데이터베이스 엔진	동적 크기 조정 기능이 있는 버전
Aurora MySQL	• 버전 3(MySQL 8.0과 호환): 지원되는 모든 버전

데이터베이스 엔진	동적 크기 조정 기능이 있는 버전
	• 버전 2 (MySQL 5.7과 호환): 2.11 이상
Aurora PostgreSQL	지원되는 모든 버전
Aurora Serverless v2	지원되는 모든 버전
Aurora Serverless v1	지원되는 모든 버전

- 위 목록에 있는 Aurora 버전보다 낮은 버전에서는 데이터를 제거할 때 복원된 공간을 클러스터 볼륨이 다시 사용할 수 있지만 볼륨 자체의 크기는 줄어들지 않습니다.
- 이 기능은 Aurora를 사용할 수 있는 AWS 리전에 단계적으로 배포되고 있습니다. 클러스터가 있는 리전에 따라 이 기능을 아직 사용하지 못할 수도 있습니다.

동적 크기 조정은 클러스터 볼륨 내의 테이블스페이스를 물리적으로 제거하거나 크기를 조정하는 작업에 적용됩니다. 따라서 `DROP TABLE`, `DROP DATABASE`, `TRUNCATE TABLE` 및 `ALTER TABLE ... DROP PARTITION`과 같은 SQL 문에 적용됩니다. `DELETE` 문을 사용하여 행을 삭제하는 경우에는 적용되지 않습니다. 테이블에서 많은 수의 행을 삭제하는 경우 이후에 Aurora MySQL `OPTIMIZE TABLE` 확장이나 Aurora PostgreSQL `pg_repack` 문을 실행해 테이블을 재구성하여 클러스터 볼륨의 크기를 동적으로 조정할 수 있습니다.

Aurora MySQL의 경우 다음 고려 사항이 적용됩니다.

- 동적 크기 조정을 지원하는 DB 엔진 버전으로 DB 클러스터를 업그레이드한 후 해당 특정 AWS 리전에서 이 기능이 활성화되면 나중에 `DROP TABLE`과 같은 특정 SQL 문에 의해 해제되는 모든 공간을 회수할 수 있습니다.

  특정 AWS 리전에서 이 기능이 명시적으로 비활성화되어 있으면 동적 크기 조정을 지원하는 버전에서도 공간을 재사용할 수만 있고 회수할 수 없습니다.

  이 기능은 2020년 11월부터 2022년 3월까지 특정 DB 엔진 버전(1.23.0-1.23.4, 2.09.0-2.09.3 및 2.10.0)에서 활성화되었으며, 이후 버전에서는 기본적으로 활성화됩니다.
- 테이블은 내부적으로 다양한 크기의 연속된 하나 이상의 조각으로 저장됩니다. 테이블 `TRUNCATE TABLE` 작업을 실행하는 동안 첫 번째 조각에 해당하는 공간은 재사용할 수 있고 회수할 수 없습니다. 다른 조각은 회수할 수 있습니다. `DROP TABLE` 작업 중에는 전체 테이블스페이스에 해당하는 공간을 회수할 수 있습니다.

- innodb_file_per_table 파라미터는 테이블 스토리지가 구성되는 방식에 영향을 미칩니다. 테이블이 시스템 테이블스페이스의 일부인 경우 테이블을 삭제해도 시스템 테이블스페이스의 크기가 줄어들지 않습니다. 따라서 동적 크기 조정을 최대한 활용하려면 Aurora MySQL DB 클러스터에서 innodb_file_per_table을 1로 설정해야 합니다.
- 버전 2.11 이상의 경우 InnoDB 임시 테이블스페이스가 삭제되고 재시작 시 다시 생성됩니다. 임시 테이블스페이스가 차지하는 공간이 시스템에 릴리스되고 클러스터 볼륨의 크기가 조정됩니다. 동적 크기 조정 기능을 최대한 활용하려면 DB 클러스터를 버전 2.11 이상으로 업그레이드하는 것이 좋습니다.

> **Note**
>
> 동적 크기 조정 기능은 테이블스페이스의 테이블이 삭제될 때 즉시 공간을 회수하는 것이 아니라 하루에 약 10TB의 속도로 공간을 점진적으로 확보합니다. 시스템 테이블스페이스는 절대 제거되지 않으므로 시스템 테이블스페이스의 공간은 회수되지 않습니다. 테이블스페이스에서 회수되지 않은 여유 공간은 작업에서 해당 테이블스페이스에 공간이 필요할 때 재사용됩니다. 동적 규모 조정 기능은 클러스터가 사용 가능한 상태일 때만 스토리지 공간을 회수할 수 있습니다.

CloudWatch에서 VolumeBytesUsed 지표를 모니터링하여 클러스터가 사용 중인 스토리지 공간의 양을 확인할 수 있습니다. 스토리지 요금 청구에 대한 자세한 정보는 [Aurora 데이터 스토리지 요금이 청구되는 방법]() 섹션을 참조하세요.

- AWS Management Console에서는 클러스터에 대한 세부 정보 페이지의 Monitoring 탭에서 차트로 이 그림을 볼 수 있습니다.
- AWS CLI를 사용하는 경우 다음 Linux 예제와 유사한 명령을 실행하면 됩니다. 이때 시작 및 종료 시간과 클러스터 이름을 해당되는 고유한 값으로 대체합니다.

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

이 명령을 실행하면 다음과 비슷한 출력이 생성됩니다.

```
{
```

```
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

다음 예제에서는 Linux 시스템에서 AWS CLI 명령을 사용하여 시간 경과에 따른 Aurora 클러스터의 스토리지 사용량을 추적하는 방법을 보여줍니다. `--start-time` 및 `--end-time` 파라미터는 전체 시간 간격을 하루로 정의합니다. `--period` 파라미터는 한 시간 간격으로 측정값을 요청합니다. 지표가 연속적으로 수집되지 않고 일정 간격으로 수집되기 때문에 작은 `--period` 값을 선택하는 것은 적절하지 않습니다. 또한 해당 SQL 문이 완료된 후에도 Aurora 스토리지 작업이 백그라운드에서 일정 시간 동안 계속되는 경우가 있습니다.

첫 번째 예제는 출력을 기본 JSON 형식으로 반환합니다. 데이터 포인트는 타임스탬프를 기준으로 정렬되지 않고 임의의 순서로 반환됩니다. 이 JSON 데이터를 차트 도구로 가져와 정렬 및 시각화를 수행할 수 있습니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
```

```
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

이 예제는 이전 예제와 동일한 데이터를 반환합니다. `--output` 파라미터는 데이터를 압축 일반 텍스트 형식으로 나타냅니다. `aws cloudwatch` 명령은 해당 출력을 `sort` 명령에 파이프합니다. `-k` 명령의 `sort` 파라미터는 UTC(협정 세계시) 형식의 타임스탬프인 세 번째 필드를 기준으로 출력을 정렬합니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
```

```
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

정렬된 출력에는 모니터링 기간의 시작 및 종료 시 사용된 스토리지 양이 표시됩니다. 또한 Aurora가 클러스터에 더 많은 스토리지를 할당한 기간 동안의 데이터 포인트도 찾을 수 있습니다. 다음 예제에서는 Linux 명령을 사용하여 시작 및 종료 `VolumeBytesUsed` 값을 기가바이트(GB)와 기비바이트(GiB) 형식으로 변경합니다. 기가바이트는 10의 제곱으로 측정된 단위를 나타내며 회전식 하드 드라이브의 스토리지를 논할 때 일반적으로 사용됩니다. 기비바이트는 2의 제곱으로 측정된 단위를 나타냅니다. Aurora 스토리지 측정 및 제한은 일반적으로 기비바이트와 테비바이트와 같은 2의 제곱 단위로 명시됩니다.

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

`VolumeBytesUsed` 지표는 클러스터에서 요금이 발생하고 있는 스토리지 양을 알려줍니다. 따라서 가능하면 이 수치를 최소화하는 것이 좋습니다. 그러나 이 지표에는 Aurora가 클러스터에서 내부적으로 사용하여 요금을 부과하지 않는 일부 스토리지가 포함되어 있습니다. 클러스터가 스토리지 제한에 근접하여 공간이 부족해질 수 있는 경우 `AuroraVolumeBytesLeftTotal` 지표를 모니터링하고 이 수치를 최대화하는 것이 더 도움이 됩니다. 다음 예제에서는 `AuroraVolumeBytesLeftTotal` 대신 `VolumeBytesUsed`에 대해 이전 계산과 비슷한 계산을 실행합니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

Aurora MySQL 버전 2.09 이상이나 Aurora PostgreSQL을 실행하는 클러스터의 경우 데이터가 추가되면 VolumeBytesUsed에서 보고되는 사용 가능한 크기가 증가하고 데이터가 제거되면 사용 가능한 크기가 감소합니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다. 이 보고서는 임시 데이터가 있는 테이블이 생성되고 삭제될 때 15분 간격으로 클러스터의 최대 및 최소 스토리지 크기를 표시합니다. 이 보고서에는 최소 값보다 최대 값이 먼저 나열됩니다. 따라서 15분 간격 내에 스토리지 사용량이 어떻게 변경되었는지 이해하려면 오른쪽에서 왼쪽으로 수치를 해석해야 합니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

다음 예시에서는 Aurora MySQL 버전 2.09 이상이나 Aurora PostgreSQL을 실행하는 클러스터의 경우 AuroraVolumeBytesLeftTotal에서 보고하는 사용 가능한 크기에 128TiB 크기 제한이 어떻게 반영되는지 보여줍니다.

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
```

```
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
127 TiB remaining for this cluster
```

인스턴스 조정

필요에 따라 DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora DB 클러스터의 크기를 조정할 수 있습니다. Aurora는 데이터베이스 엔진 호환성에 따라 Aurora에 최적화된 여러 DB 인스턴스 클래스를 지원합니다.

데이터베이스 엔진	인스턴스 조정
Amazon Aurora MySQL	Aurora MySQL DB 인스턴스 조정섹션을 참조하십시오.
Amazon Aurora PostgreSQL	Aurora PostgreSQL DB 인스턴스 조정섹션을 참조하십시오.

읽기 확장

DB 클러스터에서 Aurora 복제본을 최대 15개까지 생성하여 Aurora DB 클러스터에 대한 읽기 조정을 수행할 수 있습니다. 각 Aurora 복제본은 복제본 지연 시간—을 최소화하여 클러스터 볼륨에서 동일한 데이터를 반환합니다(일반적으로 이 지연 시간은 기본 인스턴스가 업데이트를 적용한 후 100밀리초 미만입니다). 읽기 트래픽이 증가하면 Aurora 복제본을 추가 생성하여 직접 연결함으로써 DB 클러스터의 읽기 부하를 분산시키는 것도 가능합니다. Aurora 복제본의 DB 인스턴스 클래스가 기본 인스턴스의 DB 인스턴스 클래스와 같을 필요는 없습니다.

DB 클러스터에 Aurora 복제본을 추가하는 방법에 대한 자세한 내용은 DB 클러스터에 Aurora 복제본 추가 단원을 참조하십시오.

연결 관리

Aurora DB 인스턴스에 대해 허용되는 최대 연결 수는 DB 인스턴스의 인스턴스 수준 파라미터 그룹의 `max_connections` 파라미터로 결정됩니다. 파라미터 기본값은 DB 인스턴스에 사용되는 DB 인스턴스 클래스와 데이터베이스 엔진 호환성에 따라 달라집니다.

데이터베이스 엔진	max_connections 기본값
Amazon Aurora MySQL	Aurora MySQL DB 인스턴스에 대한 최대 연결섹션을 참조하십시오.
Amazon Aurora PostgreSQL	Aurora PostgreSQL DB 인스턴스에 대한 최대 연결 섹션을 참조하세요.

> **Tip**
> 애플리케이션이 연결을 자주 열고 닫거나, 수명이 긴 연결을 많이 열어 두는 경우, Amazon RDS 프록시를 사용하는 것이 좋습니다. RDS 프록시는 데이터베이스 연결을 효율적으로 안전하게 공유하기 위해 연결 풀링을 사용하는 완전관리형 고가용성 데이터베이스 프록시입니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

쿼리 실행 계획 관리

Aurora PostgreSQL용 쿼리 계획 관리 기능을 사용하면 최적화 프로그램에서 실행할 계획을 제어할 수 있습니다. 자세한 내용은 Aurora PostgreSQL용 쿼리 실행 계획 관리 섹션을 참조하세요.

Aurora DB 클러스터에 대한 볼륨 복제

Aurora 복제를 사용하면 처음에는 원본과 동일한 데이터 페이지를 공유하지만 별도의 독립적인 볼륨인 새 클러스터를 생성할 수 있습니다. 이 프로세스는 빠르고 비용 효율적으로 진행되도록 설계되었습니다. 연결된 데이터 볼륨이 있는 새 클러스터를 복제본이라고 합니다. 복제본 생성은 스냅샷 복원과 같은 다른 기술을 사용하여 데이터를 물리적으로 복사하는 것보다 빠르고 공간 효율적입니다.

주제

Aurora 복제 개요

Aurora는 기록 중 복사(Copy-on-Write) 프로토콜을 사용하여 복제본을 생성합니다. 이 메커니즘은 최소한의 추가 공간을 사용하여 초기 복제를 만듭니다. 복제가 처음 생성되면 Aurora는 소스 Aurora DB 클러스터와 새로운(복제된) Aurora DB 클러스터에서 사용하는 데이터의 단일 복사본을 유지합니다. 소스 Aurora DB 클러스터 또는 Aurora DB 클러스터 복제가 Aurora 스토리지 볼륨의 데이터를 변경한 경우에만 추가 스토리지가 할당됩니다. 기록 중 복사(Copy-on-Write) 프로토콜에 대한 자세한 내용은 Aurora 복제 작동 방식 섹션을 참조하세요.

Aurora 복제 작업은 데이터 손상 위험 없이 프로덕션 데이터를 사용하여 테스트 환경을 신속하게 설정하는 데 특히 유용합니다. 다음과 같은 여러 유형의 애플리케이션에 복제본을 사용할 수 있습니다.

- 잠재적 변경 사항(예: 스키마 변경 및 파라미터 그룹 변경)을 실험하여 모든 영향을 평가합니다.
- 데이터 내보내기 또는 복제본에서 분석 쿼리 실행과 같은 워크로드 집약적인 작업을 수행하는 경우
- 개발, 테스트 또는 기타 용도로 프로덕션 DB 클러스터의 복사본을 생성합니다.

동일한 Aurora DB 클러스터에서 둘 이상의 복제본을 생성할 수 있습니다. 다른 복제본에서 여러 복제본을 생성할 수도 있습니다.

Aurora 복제본을 생성한 후 Aurora DB 인스턴스를 소스 Aurora DB 클러스터와 다르게 구성할 수 있습니다. 예를 들어 소스 프로덕션 Aurora DB 클러스터와 동일한 고가용성 요구 사항을 충족하기 위해 개

발 목적으로 복제본이 필요하지 않을 수 있습니다. 이 경우 Aurora DB 클러스터에서 사용하는 여러 DB 인스턴스가 아닌 단일 Aurora DB 인스턴스로 복제본을 구성할 수 있습니다.

소스와 다른 배포 구성을 사용하여 복제를 생성하면 소스 Aurora DB 엔진의 최신 마이너 버전을 사용하여 복제본이 생성됩니다.

Aurora DB 클러스터에서 복제를 생성하면 소스 Aurora DB 클러스터를 소유하는 계정과 동일한 AWS 계정에서 복제본이 생성됩니다. 그러나 Aurora Serverless v2 및 프로비저닝된 Aurora DB 클러스터와 복제본을 다른 AWS 계정에 공유할 수도 있습니다. 자세한 내용은 AWS RAM과 Amazon Aurora에서 교차 계정 복제 단원을 참조하십시오.

복제본을 테스트, 개발 또는 다른 용도로 사용한 후 삭제할 수 있습니다.

Aurora 복제의 제한 사항

Aurora 복제는 다음과 같은 제한 사항이 있습니다.

- AWS 리전에서 허용하는 최대 DB 클러스터 개수까지 원하는 만큼 복제본을 생성할 수 있습니다.
- copy-on-write 프로토콜을 사용하여 최대 15개의 복제본을 만들 수 있습니다. 그러나 15개의 복제본을 생성한 후에는 다음에 생성하는 복제본이 전체 복제본이 됩니다. 전체 복사 프로토콜은 특정 시점 복구와 같은 역할을 합니다.
- 소스 Aurora DB 클러스터에서 다른 AWS 리전에 복제본을 생성할 수 없습니다.
- 병렬 쿼리 기능이 없는 Aurora DB 클러스터에서 병렬 쿼리를 사용하는 클러스터로의 복제본을 생성할 수 없습니다. 병렬 쿼리를 사용하는 클러스터에 데이터를 가져오려면 원본 클러스터의 스냅샷을 생성하여 병렬 쿼리 기능을 사용하는 클러스터에 복원합니다.
- DB 인스턴스가 없는 Aurora DB 클러스터에서 복제본을 생성할 수 없습니다. 하나 이상의 DB 인스턴스가 있는 Aurora DB 클러스터만 복제할 수 있습니다.
- 복제본을 Aurora DB 클러스터의 Virtual Private Cloud(VPC)와 다른 Virtual Private Cloud(VPC)에 생성할 수 있습니다. 이렇게 하면 VPC의 서브넷을 동일한 가용 영역에 매핑해야 합니다.
- 프로비저닝된 Aurora DB 클러스터에서 Aurora 프로비저닝된 복제본을 생성할 수 있습니다.
- Aurora Serverless v2 인스턴스가 있는 클러스터는 프로비저닝된 클러스터와 동일한 규칙을 따릅니다.
- Aurora Serverless v1의 경우:
  - Aurora Serverless v1 DB 클러스터에서 프로비저닝된 복제본을 생성할 수 있습니다.
  - Aurora Serverless v1 또는 프로비저닝된 DB 클러스터에서 Aurora Serverless v1 복제본을 생성할 수 있습니다.

- 암호화되지 않은 프로비저닝된 Aurora DB 클러스터에서는 Aurora Serverless v1 복제본을 생성할 수 없습니다.
- 교차 계정 복제는 현재 Aurora Serverless v1 DB 클러스터 복제를 지원하지 않습니다. 자세한 내용은 계정 간 복제 제한 사항 단원을 참조하십시오.
- 복제된 Aurora Serverless v1 DB 클러스터는 Aurora Serverless v1 DB 클러스터와 동일한 동작과 제한 사항이 있습니다. 자세한 내용은 Amazon Aurora Serverless v1 사용 단원을 참조하십시오.
- Aurora Serverless v1 DB 클러스터는 항상 암호화됩니다. Aurora Serverless v1 DB 클러스터를 프로비저닝된 Aurora DB 클러스터로 복제하는 경우 프로비저닝된 Aurora DB 클러스터가 암호화됩니다. 암호화 키를 선택할 수 있지만 암호화를 비활성화할 수는 없습니다. 프로비저닝된 Aurora DB 클러스터에서 Aurora Serverless v1로 복제하려면 암호화되고 프로비저닝된 Aurora DB 클러스터로 시작해야 합니다.

Aurora 복제 작동 방식

Aurora 복제는 Aurora DB 클러스터의 스토리지 계층에서 작동합니다. 이는 Aurora 스토리지 볼륨을 지원하는 기본 내구 미디어 측면에서 빠르고 공간 효율적인 기록 중 복사(copy-on-write) 프로토콜을 사용합니다. Amazon Aurora 스토리지 개요에서 Aurora 클러스터 볼륨에 대해 자세히 알아보세요.

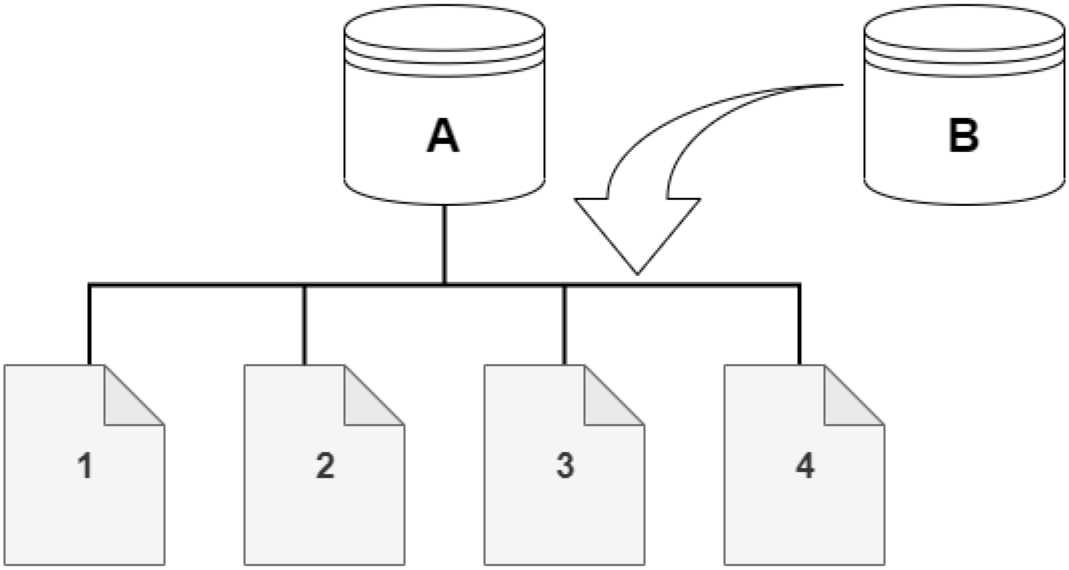
주제

- 기록 중 복사(copy-on-write) 이해
- 원본 클러스터 볼륨 삭제

기록 중 복사(copy-on-write) 이해

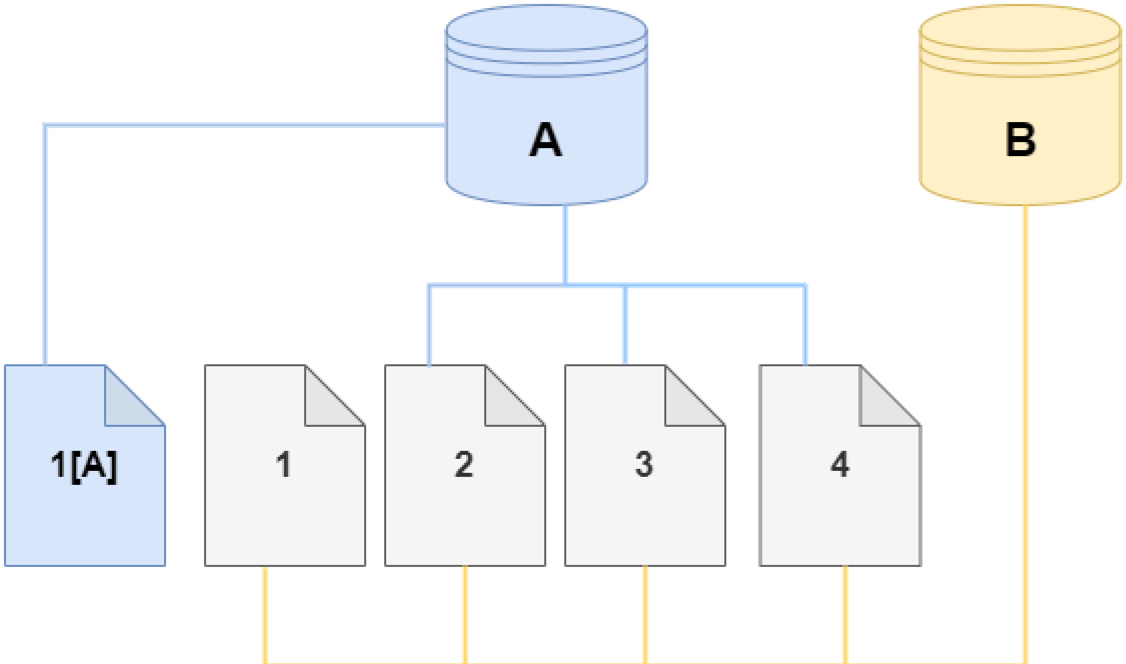
Aurora DB 클러스터는 기본 Aurora 스토리지 볼륨의 페이지에 데이터를 저장합니다.

예를 들어, 다음 다이어그램에서 데이터 페이지(1, 2, 3, 4)가 네 개인 Aurora DB 클러스터(A)를 찾을 수 있습니다. 복제본 B가 Aurora DB 클러스터에서 생성된다고 가정해 보겠습니다. 복제본이 생성되면 데이터가 복사되지 않습니다. 대신 복제본은 소스 Aurora DB 클러스터와 동일한 페이지 집합을 가리킵니다.

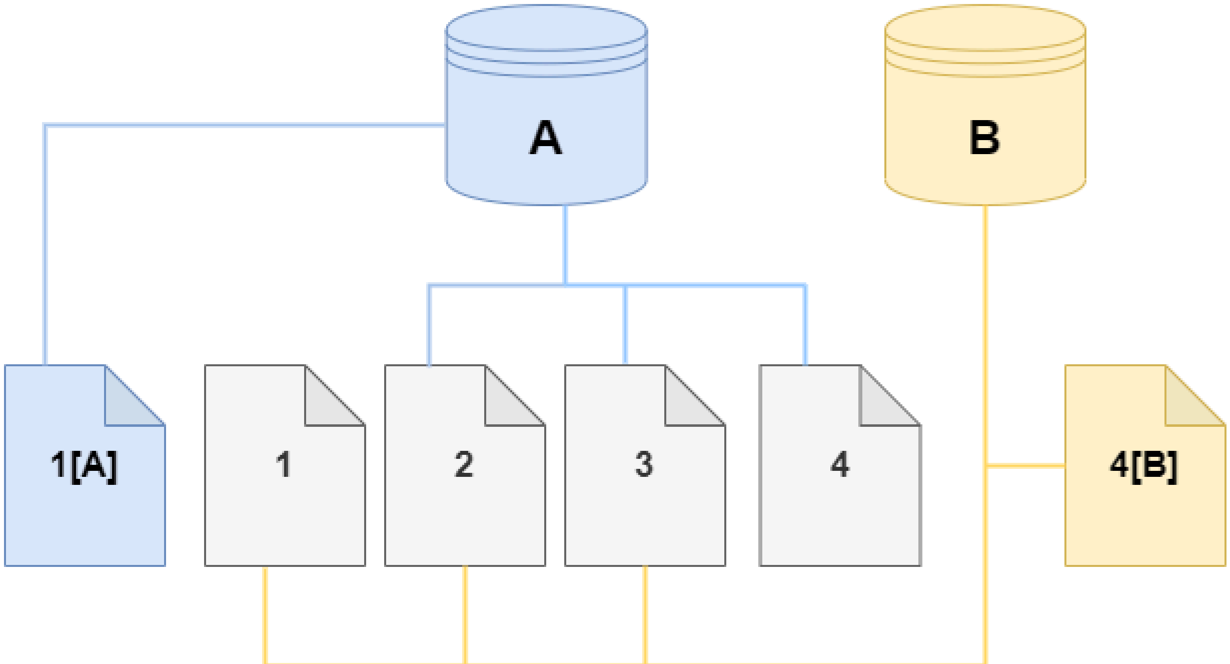


복제본이 생성되면 일반적으로 추가 스토리지가 필요하지 않습니다. 기록 중 복사(copy-on-write) 프로토콜은 물리적 스토리지 미디어에서 소스 세그먼트와 동일한 세그먼트를 사용합니다. 소스 세그먼트의 용량이 전체 복제본 세그먼트에 충분하지 않은 경우에만 추가 스토리지가 필요합니다. 이 경우 소스 세그먼트는 다른 물리적 디바이스로 복사됩니다.

다음 다이어그램에서는 앞에서와 같이 동일한 클러스터 A와 해당 복제본 B를 사용하여 작동하는 기록 중 복사(copy-on-write) 프로토콜의 예를 찾을 수 있습니다. Aurora DB 클러스터(A)를 변경하여 페이지 1에 보관된 데이터가 변경된다고 가정해 보겠습니다. Aurora는 원본 페이지 1에 기록하는 대신 새 페이지 1[A]을 생성합니다. 클러스터(A)에 대한 Aurora DB 클러스터 볼륨은 이제 페이지 1[A], 2, 3, 4를 가리키고 복제본(B)은 여전히 원본 페이지를 참조합니다.



복제본에서 스토리지 볼륨의 페이지 4가 변경됩니다. 원본 페이지 4에 기록하는 대신 Aurora는 새 페이지 4[B]를 생성합니다. 이제 복제본은 페이지 1, 2, 3 및 페이지 4[B]를 가리키고 클러스터(A)는 계속해서 1[A], 2, 3 및 4를 가리킵니다.



시간이 경과하여 소스 Aurora DB 클러스터 볼륨과 복제본 모두가 변경될 경우 변경 사항을 캡처하고 저장하기 위해 점점 더 많은 스토리지가 필요합니다.

원본 클러스터 볼륨 삭제

처음에 복제 볼륨은 복제가 생성된 원본 볼륨과 동일한 데이터 페이지를 공유합니다. 원본 볼륨이 존재하는 한 복제 볼륨은 복제본이 생성하거나 수정한 페이지의 소유자로만 간주됩니다. 따라서 복제 볼륨의 `VolumeBytesUsed` 지표는 작게 시작해서 데이터가 원본 클러스터와 복제본 간에 분산될 때만 커집니다. 소스 볼륨과 복제 간에 동일한 페이지가 있는 경우 스토리지 요금은 원본 클러스터에만 적용됩니다. `VolumeBytesUsed` 지표에 대한 자세한 내용은 [Amazon Aurora에 대한 클러스터 수준 지표]를 참고하세요.

하나 이상의 복제본이 연결된 소스 클러스터 볼륨을 삭제해도 복제본의 클러스터 볼륨에 있는 데이터는 변경되지 않습니다. Aurora는 이전에 소스 클러스터 볼륨이 소유하던 페이지를 보존합니다. Aurora는 삭제된 클러스터가 소유한 페이지에 대한 스토리지 요금을 재분배합니다. 예를 들어 원본 클러스터에 두 개의 복제본이 있었는데 원본 클러스터가 삭제되었다고 가정해 보겠습니다. 이제 원본 클러스터

가 소유한 데이터 페이지 중 절반을 하나의 복제본이 소유하게 됩니다. 페이지의 나머지 절반은 다른 복제본이 소유하게 됩니다.

원본 클러스터를 삭제한 다음 복제본을 더 생성하거나 삭제하면 Aurora는 동일한 페이지를 공유하는 모든 복제본에 데이터 페이지의 소유권을 계속 재분배합니다. 따라서 복제본의 클러스터 볼륨에 따라 `VolumeBytesUsed` 지표 값이 변경되는 것을 확인할 수 있습니다. 더 많은 복제본이 생성되고 페이지 소유권이 더 많은 클러스터에 분산됨에 따라 지표 값이 감소할 수 있습니다. 복제본이 삭제되고 페이지 소유권이 더 적은 수의 클러스터에 할당됨에 따라 지표 값이 증가할 수도 있습니다. 쓰기 작업이 복제 볼륨의 데이터 페이지에 미치는 영향에 대한 자세한 내용은 기록 중 복사(copy-on-write) 이해 섹션을 참조하세요.

원본 클러스터와 복제본을 동일한 AWS 계정에서 소유한 경우 해당 클러스터에 대한 모든 스토리지 요금이 동일한 AWS 계정에 적용됩니다. 클러스터 중 일부가 크로스 계정 복제본인 경우 원본 클러스터를 삭제하면 크로스 계정 복제본을 소유한 AWS 계정에 추가 스토리지 요금이 부과될 수 있습니다.

예를 들어 복제본을 생성하기 전에 먼저 클러스터 볼륨에 1,000개의 사용된 데이터 페이지가 있다고 가정해 보겠습니다. 해당 클러스터를 복제하면 처음에는 복제 볼륨의 사용된 페이지가 0으로 나타납니다. 복제본이 100개의 데이터 페이지를 수정하면 해당 100페이지만 복제본 볼륨에 저장되고 사용된 것으로 표시됩니다. 상위 볼륨의 변경되지 않은 나머지 900개 페이지는 두 클러스터에서 모두 공유됩니다. 이 경우 상위 클러스터의 스토리지 요금은 1,000페이지, 복제본 볼륨은 100페이지에 대한 스토리지 요금이 부과됩니다.

소스 볼륨을 삭제하는 경우 복제본에 대한 스토리지 요금에는 변경된 100페이지와 원본 볼륨의 공유 페이지 900개(총 1,000페이지)가 포함됩니다.

Amazon Aurora 복제 생성

소스 Aurora DB 클러스터와 같은 AWS 계정에서 복제본을 생성합니다. 이를 위해 AWS Management Console 또는 AWS CLI 및 다음 절차를 사용할 수 있습니다.

다른 AWS 계정에서 복제본을 생성하거나 다른 AWS 계정과 복제본을 공유하도록 허용하려면 AWS RAM과 Amazon Aurora에서 교차 계정 복제의 절차를 사용합니다.

콘솔

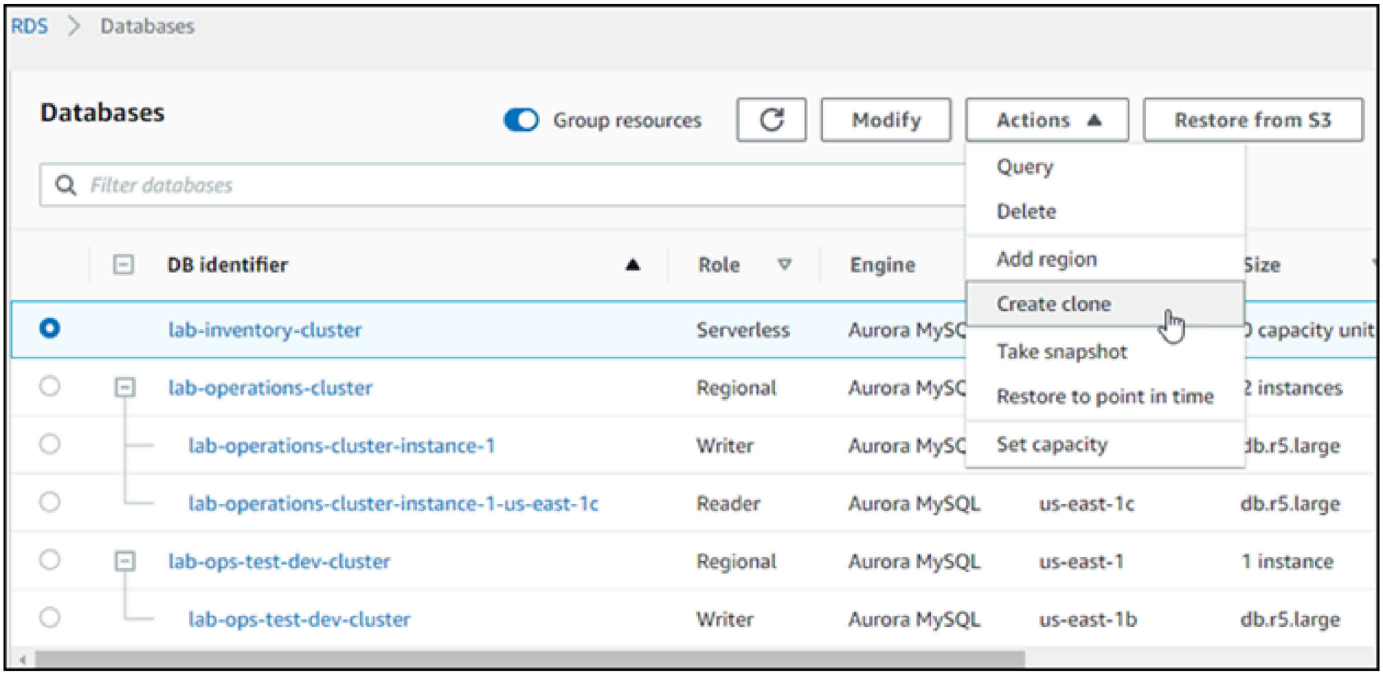
다음 프로시저에서는 AWS Management Console을 사용하여 Aurora DB 클러스터를 복제하는 방법에 대해 설명합니다.

AWS Management Console을 사용하여 복제본을 생성하면 하나의 Aurora DB 인스턴스가 있는 하나의 Aurora DB 클러스터가 생성됩니다.

이번 지침은 복제본을 생성하는 것과 동일한 AWS 계정에서 소유하고 있는 DB 클러스터에 적용됩니다. DB 클러스터를 다른 AWS 계정에서 소유하고 있다면 AWS RAM과 Amazon Aurora에서 교차 계정 복제 섹션을 참조하세요.

AWS을 사용해 AWS Management Console 계정에서 소유하는 DB 클러스터 복제본을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 목록에서 Aurora DB 클러스터를 선택하고 [작업(Actions)]에서 [복제본 생성(Create clone)]을 선택합니다.



설정, 연결성 및 Aurora DB 클러스터 복제본에 대한 기타 옵션을 구성할 수 있는 복제본 생성 페이지가 열립니다.

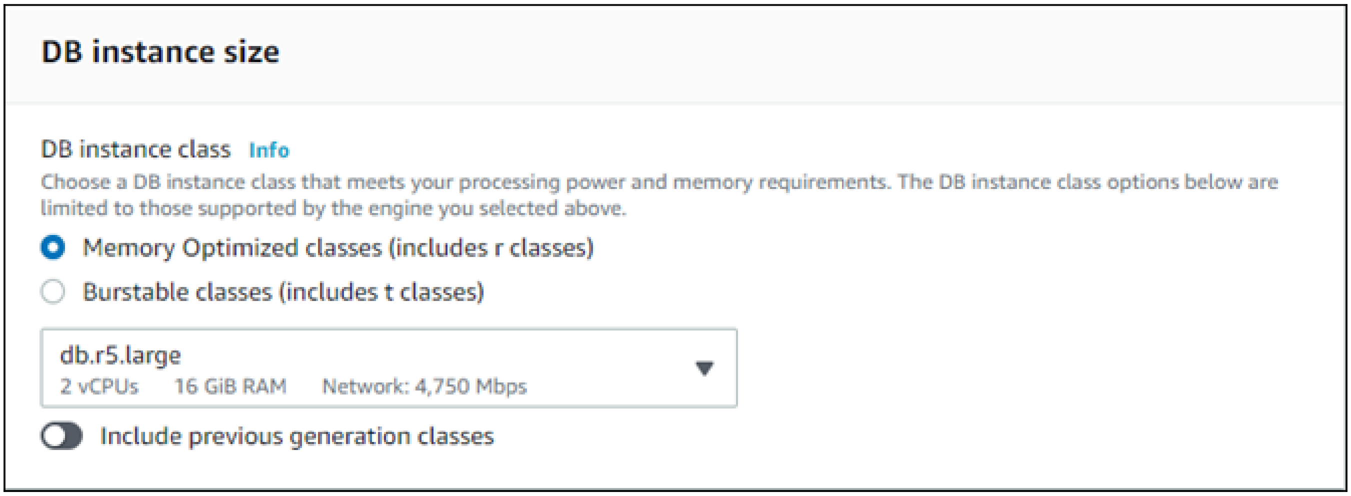
4. DB 인스턴스 식별자에 복제된 Aurora DB 클러스터에 부여할 이름을 입력합니다.
5. Aurora Serverless v1 DB 클러스터의 경우 용량 유형에서 프로비저닝됨 또는 서버리스를 선택합니다.

   소스 Aurora DB 클러스터가 Aurora Serverless v1 DB 클러스터이거나 암호화되고 프로비저닝된 Aurora DB 클러스터인 경우 [서버리스(Serverless)]를 선택합니다.
6. Aurora Serverless v2 또는 프로비저닝된 DB 클러스터의 경우 클러스터 스토리지 구성에서 Aurora I/O-Optimized 또는 Aurora Standard 중 하나를 선택합니다.

자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 단원을 참조하십시오.

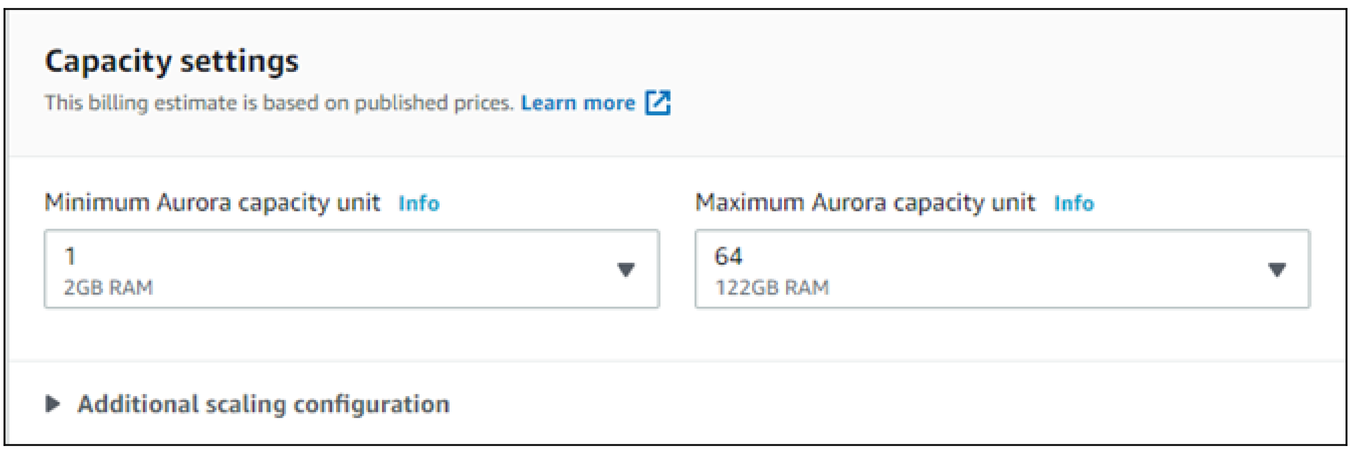
7. DB 인스턴스 크기 또는 DB 클러스터 용량을 선택합니다.

   • 프로비저닝된 복제본의 경우 DB 인스턴스 클래스를 선택합니다.



   제공된 설정을 수락하거나 복제본에 다른 DB 인스턴스 클래스를 사용할 수 있습니다.

   • Aurora Serverless v1 또는 Aurora Serverless v2 복제본의 경우 용량 설정을 선택합니다.



   제공된 설정을 수락하거나 복제본에 맞게 설정을 변경할 수 있습니다.

8. 복제본에 필요에 따라 다른 설정을 선택하세요. Aurora DB 클러스터 및 인스턴스 설정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 섹션에서 참조하세요.

9. 복제본 생성을 선택합니다.

복제본이 생성되면 복제본은 콘솔의 [데이터베이스(Databases)] 섹션에 다른 Aurora DB 클러스터와 함께 나열되고 현재 상태가 표시됩니다. 상태가 [사용 가능(Available)]이면 복제본을 사용할 준비가 된 것입니다.

AWS CLI

AWS CLI를 사용하여 Aurora DB 클러스터를 복제하려면 클론 클러스터를 만들고 여기에 하나 이상의 DB 인스턴스를 추가하는 별도의 단계가 필요합니다.

`restore-db-cluster-to-point-in-time` AWS CLI 명령을 사용하면 원래 클러스터와 동일한 스토리지 데이터를 가진 Aurora DB 클러스터가 생성되지만 Aurora DB 인스턴스는 생성되지 않습니다. 클론이 사용 가능해지면 별도로 DB 인스턴스를 생성합니다. DB 인스턴스와 해당 인스턴스 클래스의 수를 선택하여 클론에 원래 클러스터보다 더 많거나 적은 컴퓨팅 파워를 제공할 수 있습니다. 프로세스의 단계는 다음과 같습니다.

1. restore-db-cluster-to-point-in-time CLI 명령을 사용하여 복제본을 생성합니다.
2. create-db-instance CLI 명령을 사용하여 클론에 대한 라이터 DB 인스턴스를 생성합니다.
3. (선택 사항) 추가 create-db-instance CLI 명령을 실행하여 하나 이상의 리더 인스턴스를 클론 클러스터에 추가합니다. 리더 인스턴스를 사용하면 클론의 고가용성 및 읽기 확장성 측면을 개선하는 데 도움이 됩니다. 개발 및 테스트용으로만 클론을 사용하려는 경우에는 이 단계를 건너뛰어도 됩니다.

주제

- 복제본 생성
- 상태 확인 및 복제본 세부 정보 가져오기
- 복제본에 대한 Aurora DB 인스턴스 생성
- 복제본에 사용할 파라미터

복제본 생성

`restore-db-cluster-to-point-in-time` CLI 명령을 사용하여 초기 클론 클러스터를 생성합니다.

소스 Aurora DB 클러스터로부터 클론을 생성하는 방법

- `restore-db-cluster-to-point-in-time` CLI 명령을 사용합니다. 다음 파라미터의 값을 지정합니다. 이 일반적인 경우에는 클론이 원래 클러스터와 동일한 엔진 모드(프로비저닝된 모드 또는 Aurora Serverless v1)를 사용합니다.

- `--db-cluster-identifier` - 복제본에 대해 의미 있는 이름을 선택합니다. 복제본의 이름을 지정할 때 restore-db-cluster-to-point-in-time CLI 명령을 사용합니다. 그런 다음 create-db-instance CLI 명령에 복제본 이름을 전달합니다.
- `--restore-type` – `copy-on-write`을 사용하여 소스 DB 클러스터의 복제본을 생성합니다. 이 파라미터가 없으면 `restore-db-cluster-to-point-in-time`은 복제본을 생성하는 대신 Aurora DB 클러스터를 복원합니다.
- `--source-db-cluster-identifier` - 복제할 소스 Aurora DB 클러스터의 이름을 사용합니다.
- `--use-latest-restorable-time` - 이 값은 소스 DB 클러스터에 대해 복원 가능한 최신 볼륨 데이터를 가리킵니다. 이를 사용하여 클론을 생성할 수 있습니다.

다음 예제에서는 `my-source-cluster`라는 이름의 클러스터에서 `my-clone`라는 이름의 복제본을 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

이 명령은 복제본의 세부 사항을 포함하는 JSON 객체를 반환합니다. 클론에 대한 DB 인스턴스를 생성하기 전에 복제된 DB 클러스터를 사용할 수 있는지 확인합니다. 자세한 내용은 상태 확인 및 복제본 세부 정보 가져오기 단원을 참조하십시오.

예를 들어 복제하고자 하는 `tpch100g`라는 이름의 클러스터가 있다고 가정합니다. 다음 Linux 예제는 `tpch100g-clone`이라는 복제된 클러스터, `tpch100g-clone-instance`라는 Aurora Serverless v2 라이터 인스턴스 및 새 클러스터를 위한 `tpch100g-clone-instance-2`라는 프로비저닝된 리더 인스턴스를 만듭니다.

`--master-username` 및 `--master-user-password`와 같은 일부 파라미터를 제공할 필요가 없습니다. Aurora는 원본 클러스터에서 파라미터를 자동으로 결정합니다. 사용할 DB 엔진을 지정해야 합니다. 따라서 이 예제는 새 클러스터를 테스트하여 `--engine` 파라미터에 사용할 올바른 값을 결정합니다.

이 예에는 클론 클러스터를 생성할 때의 `--serverless-v2-scaling-configuration` 옵션도 포함되어 있습니다. 이렇게 하면 원래 클러스터에서 Aurora Serverless v2를 사용하지 않았더라도 클론에 Aurora Serverless v2 인스턴스를 추가할 수 있습니다.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.serverless \
  --engine aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance-2 \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r6g.2xlarge \
  --engine aurora-mysql
```

소스 Aurora DB 클러스터와 다른 엔진 모드를 사용하여 복제본을 생성하는 방법

- 이 절차는 Aurora Serverless v1을 지원하는 이전 엔진 버전에만 적용됩니다. Aurora Serverless v1 클러스터가 있고 프로비저닝된 클러스터인 클론을 생성하려고 한다고 가정해 보겠습니다. 이 경우 [`restore-db-cluster-to-point-in-time`]() CLI 명령을 사용하여 이전 예제와 유사한 파라미터 값과 함께 다음 추가 파라미터를 지정합니다.

- `--engine-mode` - 소스 Aurora DB 클러스터와 다른 엔진 모드를 사용하는 클론을 생성하는 경우에만 이 파라미터를 사용합니다. 이 파라미터는 Aurora Serverless v1을 지원하는 이전 엔진 버전에만 적용됩니다. 다음과 같이 `--engine-mode`를 사용하여 전달할 값을 선택합니다.
  - `--engine-mode provisioned`을 사용하여 Aurora Serverless DB 클러스터에서 프로비저닝된 Aurora DB 클러스터 복제본을 생성합니다.

    > **Note**
    >
    > Aurora Serverless v1에서 복제된 클러스터와 함께 Aurora Serverless v2를 사용하려는 경우에도 클론의 엔진 모드를 `provisioned`로 지정해야 합니다. 그런 다음 이후에 추가 업그레이드 및 마이그레이션 단계를 수행합니다.

  - `--engine-mode serverless`를 사용하여 프로비저닝된 Aurora DB 클러스터에서 Aurora Serverless v1 클론을 생성합니다. `serverless` 엔진 모드를 지정할 때 `--scaling-configuration`을 선택할 수도 있습니다.
- `--scaling-configuration` - (선택 사항) `--engine-mode serverless`를 사용하여 Aurora Serverless v1 복제본의 최소 및 최대 용량을 구성합니다. 이 파라미터를 사용하지 않는 경우 Aurora가 DB 엔진의 기본 Aurora Serverless v1 용량 값을 사용하여 Aurora Serverless v1 클론을 생성합니다.

다음 예제에서는 `my-source-cluster`라는 이름의 Aurora Serverless v1 DB 클러스터에서 `my-clone`이라는 이름의 프로비저닝된 클론을 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode provisioned \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode provisioned ^
```

```
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

이 명령은 DB 인스턴스를 생성하는 데 필요한 복제본의 세부 정보가 포함된 JSON 객체를 반환합니다. 복제본 상태(빈 Aurora DB 클러스터)가 사용 가능(Available) 상태가 될 때까지는 작업을 수행할 수 있습니다.

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 명령은 해당 DB 클러스터의 DB 인스턴스가 아닌 DB 클러스터만 복원합니다. 복원된 DB 클러스터에 대한 DB 인스턴스를 생성하려면 create-db-instance 명령을 실행합니다. 해당 명령으로 복원된 DB 클러스터의 식별자를 `--db-cluster-identifier` 파라미터로 지정합니다. `restore-db-cluster-to-point-in-time` 명령이 완료되고 DB 클러스터를 사용 가능할 때만 DB 인스턴스를 생성할 수 있습니다. Aurora Serverless v1 클러스터로 시작해서 Aurora Serverless v2 클러스터로 마이그레이션하려고 한다고 가정해 보겠습니다. 마이그레이션의 초기 단계로 Aurora Serverless v1 클러스터의 프로비저닝된 클론을 생성합니다. 필요한 버전 업그레이드를 포함한 전체 절차는 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드 단원을 참조하세요.

상태 확인 및 복제본 세부 정보 가져오기

다음 명령을 사용하여 새로 생성된 클론 클러스터의 상태를 확인할 수 있습니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

또는 다음 AWS CLI 쿼리를 사용하여 복제본에 대한 DB 인스턴스를 생성하는 데 필요한 다른 값 및 상태를 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

Windows의 경우:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
```

```
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

이 쿼리는 다음과 비슷한 출력을 반환합니다.

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

복제본에 대한 Aurora DB 인스턴스 생성

create-db-instance CLI 명령을 사용하여 Aurora Serverless v2 또는 프로비저닝된 복제본에 대한 DB 인스턴스를 생성합니다. Aurora Serverless v1 클론을 위한 DB 인스턴스는 만들지 않습니다.

DB 인스턴스는 소스 DB 클러스터에서 `--master-username` 및 `--master-user-password` 속성을 상속합니다.

다음은 프로비저닝된 복제본에 대한 DB 인스턴스를 생성하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.r6g.2xlarge \
    --engine aurora-mysql
```

Windows의 경우:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r6g.2xlarge ^
    --engine aurora-mysql
```

다음 예제에서는 Aurora Serverless v2를 지원하는 엔진 버전을 사용하는 클론을 위해 Aurora Serverless v2 DB 인스턴스를 만듭니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.serverless \
  --engine aurora-postgresql
```

Windows의 경우:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.serverless ^
    --engine aurora-mysql
```

복제본에 사용할 파라미터

다음 표에는 `restore-db-cluster-to-point-in-time`에서 Aurora DB 클러스터를 복제하는 데 사용되는 다양한 파라미터가 요약되어 있습니다.

파라미터	설명
`--source-db-cluster-identifier`	복제할 소스 Aurora DB 클러스터의 이름을 사용합니다.
`--db-cluster-identifier`	`restore-db-cluster-to-point-in-time` 명령을 사용하여 복제본을 생성할 때는 의미가 있는 이름을 지정하세요. 그런 다음 이 이름을 `create-db-instance` 명령으로 전달합니다.
`--restore-type`	`copy-on-write` 를 `--restore-type` 로 지정하여 소스 Aurora DB 클러스터를 복원하는 대신 소스 DB 클러스터의 복제본을 생성합니다.
`--use-latest-restorable-time`	이 값은 소스 DB 클러스터에 대해 복원 가능한 최신 볼륨 데이터를 가리킵니다. 이를 사용하여 클론을 생성할 수 있습니다.

파라미터	설명
`--serverless-v2-scaling-configuration`	(Aurora Serverless v2를 지원하는 최신 버전) 이 파라미터를 사용하여 Aurora Serverless v2 클론의 최소 및 최대 용량을 구성합니다. 이 파라미터를 지정하지 않으면 클러스터를 수정하여 이 속성을 추가하기 전까지는 클론 클러스터에 Aurora Serverless v2 인스턴스를 만들 수 없습니다.
`--engine-mode`	(Aurora Serverless v1만 지원하는 이전 버전) 이 파라미터를 사용하여 소스 Aurora DB 클러스터와 다른 유형의 클론을 생성합니다. 클론에는 다음 값 중 하나가 포함되어야 합니다. • `provisioned` 를 사용하여 Aurora Serverless v1 DB 클러스터에서 프로비저닝된 클론을 생성합니다. • `serverless` 를 사용하여 프로비저닝된 DB 클러스터 또는 Aurora Serverless v2 DB 클러스터에서 Aurora Serverless v1 복제본을 생성합니다. `serverless` 엔진 모드를 지정할 때 `--scaling-configuration` 을 선택할 수도 있습니다.
`--scaling-configuration`	(Aurora Serverless v1만 지원하는 이전 버전) 이 파라미터를 사용하여 Aurora Serverless v1 클론의 최소 및 최대 용량을 구성합니다. 이 파라미터를 지정하지 않는 경우 Aurora가 DB 엔진의 기본 용량 값을 사용하여 복제본을 생성합니다.

VPC 간 및 계정 간 복제에 대한 자세한 내용은 다음 섹션을 참조하세요.

주제

- Amazon Aurora를 사용한 VPC 간 복제
- AWS RAM과 Amazon Aurora에서 교차 계정 복제

Amazon Aurora를 사용한 VPC 간 복제

원본 클러스터와 복제본에 서로 다른 네트워크 액세스 제어를 적용하고 싶다고 가정해 보겠습니다. 예를 들어 복제를 사용하여 개발 및 테스트를 위해 다른 VPC에 프로덕션 Aurora 클러스터의 복사본을 만들 수 있습니다. 또는 데이터베이스 보안을 강화하기 위해 퍼블릭 서브넷에서 프라이빗 서브넷으로의 마이그레이션의 일부로 복제본을 만들 수도 있습니다.

다음 섹션에서는 원본 클러스터와 복제본이 서로 다른 서브넷이나 다른 VPC에서도 동일한 Aurora 스토리지 노드에 액세스할 수 있도록 복제본의 네트워크 구성을 설정하는 방법을 설명합니다. 네트워크 리소스를 미리 확인하면 복제 중 진단하기 어려운 오류를 방지할 수 있습니다.

Aurora가 VPC, 서브넷 및 DB 서브넷 그룹과 상호 작용하는 방식에 익숙하지 않은 경우 먼저 [Amazon VPC 및 Amazon Aurora] 섹션을 참조하세요. 해당 섹션의 자습서를 통해 AWS에서 이러한 종류의 리소스를 생성하고 이러한 리소스가 서로 어떻게 결합되는지 이해할 수 있습니다.

이 단계에는 RDS와 EC2 서비스 간 전환이 포함되므로, 예제에서는 이러한 작업을 자동화하고 출력을 저장하는 방법을 이해하는 데 도움이 되도록 AWS CLI 명령을 사용합니다.

주제

시작하기 전 준비 사항

VPC 간 복제 설정을 시작하기 전에 다음 리소스가 준비되어 있는지 확인하세요.

- 복제를 위한 소스로 사용할 Aurora DB 클러스터. Aurora DB 클러스터를 처음 만드는 경우, Amazon Aurora 시작하기의 자습서를 참조하여 MySQL 또는 PostgreSQL 데이터베이스 엔진을 사용하여 클러스터를 설정하세요.
- VPC 간 복제본을 생성하려는 경우 두 번째 VPC. 복제본에 사용할 VPC가 없는 경우에는 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 또는 자습서: DB 클러스터(듀얼 스택 모드)에 사용할 VPC 생성를 참조하세요.

네트워크 환경에 대한 정보 수집

VPC 간 복제를 사용하면 네트워크 환경이 원본 클러스터와 복제본 간에 크게 다를 수 있습니다. 복제본을 만들기 전에 원본 클러스터에서 사용된 VPC, 서브넷, DB 서브넷 그룹 및 AZ에 대한 정보를 수집하여 기록하세요. 이렇게 하면 문제 발생 가능성을 최소화할 수 있습니다. 네트워크 문제가 발생하는 경우 진단 정보를 검색하기 위해 문제 해결 활동을 중단할 필요가 없습니다. 다음 섹션에서는 이러한

유형의 정보를 수집하는 CLI 예제를 보여줍니다. 복제본을 만들고 문제 해결을 수행하는 동안 참조하기 편리한 형식으로 세부 정보를 저장할 수 있습니다.

- 1단계: 원본 클러스터의 가용 영역 확인
- 2단계: 원본 클러스터의 DB 서브넷 그룹 확인
- 3단계: 원본 클러스터의 서브넷 확인
- 4단계: 원본 클러스터에서 DB 인스턴스의 가용 영역 확인
- 5단계: 복제본에 사용할 수 있는 VPC 확인

1단계: 원본 클러스터의 가용 영역 확인

복제본을 생성하기 전에 원본 클러스터가 스토리지에 사용하는 AZ를 확인하세요. Amazon Aurora 스토리지에 설명된 대로, 각 Aurora 클러스터의 스토리지는 정확히 3개의 AZ와 연결되어 있습니다. Amazon Aurora DB 클러스터는 컴퓨팅과 스토리지의 분리를 활용하기 때문에 이 규칙은 클러스터에 있는 인스턴스 수에 관계없이 적용됩니다.

예를 들어, 다음과 같은 CLI 명령을 실행하여 *my_cluster*를 자신의 클러스터 이름으로 대체합니다. 다음 예제는 AZ 이름에 따라 알파벳순으로 정렬된 목록을 생성합니다.

```
aws rds describe-db-clusters \
  --db-cluster-identifier my_cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' \
  --output text
```

다음 예제는 앞의 `describe-db-clusters` 명령의 샘플 출력을 보여줍니다. 이 그림은 Aurora 클러스터의 스토리지가 항상 3개의 AZ를 사용한다는 것을 보여줍니다.

```
us-east-1c
us-east-1d
us-east-1e
```

이러한 AZ에 연결할 수 있는 모든 리소스가 준비되지 않은 네트워크 환경에서 복제본을 만들려면 해당 AZ 중 2개 이상과 연결된 서브넷을 만든 다음 해당 2~3개의 서브넷을 포함하는 DB 서브넷 그룹을 만들어야 합니다. 다음 예제에서는 그 방법을 보여줍니다.

2단계: 원본 클러스터의 DB 서브넷 그룹 확인

원본 클러스터와 동일한 수의 서브넷을 복제본에 사용하려면 원본 클러스터의 DB 서브넷 그룹에서 서브넷 수를 확인할 수 있습니다. Aurora DB 서브넷 그룹에는 각각 다른 AZ와 연결된 서브넷이 2개 이상 포함되어 있습니다. 서브넷이 어떤 AZ와 연결되어 있는지 기록해 두세요.

다음 예제는 원본 클러스터의 DB 서브넷 그룹을 찾은 다음 해당 AZ로 거꾸로 작업하는 방법을 보여줍니다. 첫 번째 명령의 *my_cluster*를 사용자 클러스터의 이름으로 대체합니다. 두 번째 명령에서 DB 서브넷 그룹의 이름을 *my_subnet*으로 대체합니다.

```
aws rds describe-db-clusters --db-cluster-identifier my_cluster \
  --query '*[].DBSubnetGroup' --output text

aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetAvailabilityZone.Name]' --output text
```

두 개의 서브넷을 포함하는 DB 서브넷 그룹이 있는 클러스터의 경우 샘플 출력은 다음과 유사하게 보일 수 있습니다. 이 경우 `two-subnets`은 DB 서브넷 그룹을 생성할 때 지정한 이름입니다.

```
two-subnets

us-east-1d
us-east-1c
```

DB 서브넷 그룹에 3개의 서브넷이 포함된 클러스터의 경우, 출력은 다음과 비슷하게 보일 수 있습니다.

```
three-subnets

us-east-1f
us-east-1d
us-east-1c
```

3단계: 원본 클러스터의 서브넷 확인

원본 클러스터의 서브넷에 대한 자세한 정보가 필요한 경우 다음과 유사한 AWS CLI 명령을 실행하세요. P 주소 범위, 소유자 등과 같은 서브넷 속성을 확인할 수 있습니다. 이렇게 하면 동일한 VPC에서 다른 서브넷을 사용할지, 아니면 다른 VPC에 비슷한 특성을 가진 서브넷을 만들지 결정할 수 있습니다.

VPC에서 사용할 수 있는 모든 서브넷의 서브넷 ID를 찾습니다.

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc \
  --query '*[].[SubnetId]' --output text
```

DB 서브넷 그룹에서 사용되는 정확한 서브넷을 찾습니다.

```
aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetIdentifier]' --output text
```

그리고 나서 다음 명령어와 같이 조사할 서브넷을 목록으로 지정합니다. 서브넷의 이름을 *my_subnet_1* 등으로 대체합니다.

```
aws ec2 describe-subnets \
  --subnet-ids '["my_subnet_1","my_subnet2","my_subnet3"]'
```

다음 예제는 이러한 `describe-subnets` 명령의 일부 출력을 보여줍니다. 출력에는 각 서브넷에 대해 볼 수 있는 몇 가지 중요한 속성(예: 연결된 AZ 및 해당 서브넷이 속한 VPC)이 표시됩니다.

```
{
    'Subnets': [
        {
            'AvailabilityZone': 'us-east-1d',
            'AvailableIpAddressCount': 54,
            'CidrBlock': '10.0.0.64/26',
            'State': 'available',
            'SubnetId': 'subnet-000a0bca00e0b0000',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
        },
        {
            'AvailabilityZone': 'us-east-1c',
            'AvailableIpAddressCount': 55,
            'CidrBlock': '10.0.0.0/26',
            'State': 'available',
            'SubnetId': 'subnet-4b4dbfe4d4a4fd4c4',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
```

4단계: 원본 클러스터에서 DB 인스턴스의 가용 영역 확인

이 절차를 사용하여 원본 클러스터의 DB 인스턴스에 사용된 AZ를 파악할 수 있습니다. 이렇게 하면 복제본의 DB 인스턴스에 대해 정확히 동일한 AZ를 설정할 수 있습니다. 또한 복제본을 프로덕션, 개발, 테스트 등에 사용하는지 여부에 따라 복제본에 더 많은 또는 더 적은 수의 DB 인스턴스를 사용할 수 있습니다.

원본 클러스터의 각 인스턴스에 대해 다음과 같은 명령을 실행합니다. 먼저 인스턴스 생성이 완료되어 `Available` 상태가 되었는지 확인합니다. 인스턴스 식별자를 *my_instance*로 대체합니다.

```
aws rds describe-db-instances --db-instance-identifier my_instance \
  --query '*[].AvailabilityZone' --output text
```

다음 예제는 앞의 describe-db-instances 명령을 실행한 결과를 보여줍니다. Aurora 클러스터에는 4개의 데이터베이스 인스턴스가 있습니다. 따라서 매번 다른 DB 인스턴스 식별자로 대체하여 명령을 네 번 실행합니다. 출력은 이러한 DB 인스턴스가 최대 3개의 AZ에 어떻게 분산되어 있는지 보여줍니다.

```
us-east-1a
us-east-1c
us-east-1d
us-east-1a
```

복제본이 생성되고 DB 인스턴스를 추가하는 경우, create-db-instance 명령에서 동일한 AZ 이름을 지정할 수 있습니다. 이렇게 하면 원본 클러스터와 정확히 동일한 AZ로 구성된 새 클러스터에 DB 인스턴스를 설정할 수 있습니다.

5단계: 복제본에 사용할 수 있는 VPC 확인

원본과 다른 VPC에서 복제본을 만들려는 경우, 계정에 사용할 수 있는 VPC ID 목록을 확인할 수 있습니다. 원본 클러스터와 동일한 VPC에 추가 서브넷을 만들어야 하는 경우에도 이 단계를 수행할 수 있습니다. 서브넷을 만드는 명령을 실행할 때 파라미터로 VPC ID를 지정합니다.

계정의 모든 VPC를 나열하려면 다음 CLI 명령을 실행합니다.

```
aws ec2 describe-vpcs --query '*[].[VpcId]' --output text
```

다음 예제는 앞의 describe-vpcs 명령의 샘플 출력을 보여줍니다. 출력은 현재 AWS 계정에 VPC 간 복제를 위한 소스 또는 대상으로 사용할 수 있는 4개의 VPC가 있음을 보여줍니다.

```
vpc-fd111111
vpc-2222e2cd2a222f22e
vpc-33333333a33333d33
vpc-4ae4d4de4a4444dad
```

복제 대상과 동일한 VPC를 사용하거나 다른 VPC를 사용할 수 있습니다. 원본 클러스터와 복제본이 동일한 VPC에 있는 경우, 복제본에 동일한 DB 서브넷 그룹을 재사용할 수 있습니다. 다른 DB 서브넷 그룹을 생성할 수도 있습니다. 예를 들어, 새 DB 서브넷 그룹은 프라이빗 서브넷을 사용하고 원본 클러스

터의 DB 서브넷 그룹은 퍼블릭 서브넷을 사용할 수 있습니다. 다른 VPC에서 복제본을 생성하는 경우, 새 VPC에 서브넷이 충분한지, 서브넷이 원본 클러스터의 올바른 AZ와 연결되어 있는지 확인하세요.

복제본을 위한 네트워크 리소스 생성

네트워크 정보를 수집하는 동안 복제본에 추가 네트워크 리소스가 필요하다는 것을 발견한 경우, 복제본을 설정하기 전에 해당 리소스를 만들 수 있습니다. 예를 들어, 더 많은 서브넷, 특정 AZ와 연결된 서브넷 또는 새 DB 서브넷 그룹을 만들어야 할 수 있습니다.

- 1단계: 복제본의 서브넷 생성
- 2단계: 복제본에 대한 DB 서브넷 그룹 생성

1단계: 복제본의 서브넷 생성

복제본에 대한 새 서브넷을 만들어야 하는 경우 다음과 유사한 명령을 실행합니다. 다른 VPC에서 복제본을 만들거나 퍼블릭 서브넷 대신 프라이빗 서브넷을 사용하는 등 다른 네트워크를 변경할 때 이 작업을 수행해야 할 수 있습니다.

AWS는 서브넷의 ID를 자동으로 생성합니다. 복제본의 VPC 이름을 *my_vpc*로 대체합니다. `--cidr-block` 옵션의 주소 범위를 선택하여 해당 범위에서 최소 16개의 IP 주소를 허용합니다. 지정하려는 다른 속성을 포함할 수 있습니다. `aws ec2 create-subnet help` 명령을 실행하여 모든 선택 사항을 확인합니다.

```
aws ec2 create-subnet --vpc-id my_vpc \
  --availability-zone AZ_name --cidr-block IP_range
```

다음 예제는 새로 생성된 서브넷의 몇 가지 중요한 속성을 보여줍니다.

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1b',
        'AvailableIpAddressCount': 59,
        'CidrBlock': '10.0.0.64/26',
        'State': 'available',
        'SubnetId': 'subnet-44b4a44f4e44db444',
        'VpcId': 'vpc-555fc5df555e555dc',
        ...
        }
}
```

2단계: 복제본에 대한 DB 서브넷 그룹 생성

다른 VPC에 복제본을 만들거나 동일한 VPC 내의 다른 서브넷 집합에 복제본을 만드는 경우 새 DB 서브넷 그룹을 만들고 복제본을 만들 때 지정합니다.

다음 세부 사항을 모두 알고 있는지 확인하세요. 앞선 예제의 출력에서 이 모든 것을 확인할 수 있습니다.

1. 원본 클러스터의 VPC입니다. 자세한 내용은 3단계: 원본 클러스터의 서브넷 확인 섹션을 참조하세요.
2. 다른 VPC에서 복제본을 만드는 경우 복제본의 VPC입니다. 지침은 5단계: 복제본에 사용할 수 있는 VPC 확인 단원을 참조하십시오.
3. 원본 클러스터의 Aurora 스토리지와 연결된 3개의 AZ입니다. 지침은 1단계: 원본 클러스터의 가용 영역 확인 단원을 참조하십시오.
4. 원본 클러스터의 DB 서브넷 그룹과 연결된 두세 개의 AZ입니다. 지침은 2단계: 원본 클러스터의 DB 서브넷 그룹 확인 단원을 참조하십시오.
5. 복제본에 사용하려는 VPC에 있는 모든 서브넷의 서브넷 ID 및 연결된 AZ입니다. 3단계: 원본 클러스터의 서브넷 확인에서와 동일한 `describe-subnets` 명령을 사용하여 원본 클러스터의 서브넷을 확인하고 대상 VPC의 VPC ID를 대체합니다.

원본 클러스터의 스토리지와 연결되어 있고 대상 VPC의 서브넷과 연결되어 있는 AZ의 수를 확인합니다. 복제본을 성공적으로 생성하려면 두 개 또는 세 개의 AZ가 공통으로 있어야 합니다. 공통 AZ가 2개 미만인 경우 1단계: 복제본의 서브넷 생성으로 돌아가세요. 원본 클러스터의 스토리지와 연결된 AZ와 연결된 새 서브넷을 하나, 둘 또는 세 개 만듭니다.

대상 VPC에서 원본 클러스터의 Aurora 스토리지와 동일한 AZ와 연결된 서브넷을 선택합니다. 세 개의 AZ를 선택하는 것이 이상적입니다. 이렇게 하면 복제본의 DB 인스턴스를 여러 AZ에 분산하여 컴퓨팅 리소스의 고가용성을 가장 유연하게 확보할 수 있습니다.

다음과 유사한 명령을 실행하여 새 DB 서브넷 그룹을 생성합니다. 목록에 있는 서브넷의 ID를 대체합니다. 경 변수를 사용하여 서브넷 ID를 지정하는 경우, ID 주위에 큰따옴표가 유지되도록 `--subnet-ids` 파라미터 목록을 인용하도록 주의하세요.

```
aws rds create-db-subnet-group --db-subnet-group-name my_subnet_group \
  --subnet-ids '["my_subnet_1","my_subnet_2","my_subnet3"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets for clone'
```

다음 예제에서는 `create-db-subnet-group` 명령의 일부 출력을 보여줍니다

```
{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'my_subnet_group',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets for clone',
        'VpcId': 'vpc-555fc5df555e555dc',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'my_subnet_1',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1c'
                },
                'SubnetStatus': 'Active'
            },
            {
                'SubnetIdentifier': 'my_subnet_2',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1d'
                },
                'SubnetStatus': 'Active'
            }
            ...
        ],
        'SupportedNetworkTypes': [
            'IPV4'
        ]
    }
}
```

이 시점에서는 아직 복제본을 실제로 만들지 않았습니다. 복제본을 만들 때 `restore-db-cluster-to-point-in-time` 및 `create-db-instance` 명령에 적절한 파라미터를 지정할 수 있도록 관련 VPC 및 서브넷 리소스를 모두 만들었습니다.

새 네트워크 설정으로 Aurora 복제본 생성

새 클러스터에서 사용할 VPC, 서브넷, AZ 및 서브넷 그룹의 올바른 구성을 확인했으면 실제 복제 작업을 수행할 수 있습니다. 다음 CLI 예제에서는 복제 및 해당 DB 인스턴스를 설정하는 명령에 지정하는 `--db-subnet-group-name`, `--availability-zone`, `--vpc-security-group-ids` 등의 옵션을 중점적으로 설명합니다.

- 1단계: 복제본의 DB 서브넷 그룹 지정
- 2단계: 복제본의 인스턴스에 대한 네트워크 설정 지정

- 3단계: 클라이언트 시스템에서 복제본으로 연결 설정

1단계: 복제본의 DB 서브넷 그룹 지정

복제본을 생성할 때 DB 서브넷 그룹을 지정하여 올바른 VPC, 서브넷 및 AZ 설정을 모두 구성할 수 있습니다. 앞의 예제에 있는 명령을 사용하여 DB 서브넷 그룹에 들어가는 모든 관계와 매핑을 확인합니다.

예를 들어, 다음 명령은 원본 클러스터를 복제본으로 복제하는 것을 보여줍니다. 첫 번째 예제에서 원본 클러스터는 2개의 서브넷과 연결되고 복제본은 3개의 서브넷과 연결됩니다. 두 번째 예제에서는 서브넷이 3개인 클러스터에서 서브넷이 2개인 클러스터로 복제본을 복제하는 반대 사례를 보여줍니다.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-3-subnets \
  --db-cluster-identifier cluster-cloned-to-2-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name two-subnets
```

복제본에서 Aurora 서버리스 v2 인스턴스를 사용하려는 경우, 그림과 같이 복제본을 생성할 때 `--serverless-v2-scaling-configuration` 옵션을 포함하세요. 이렇게 하면 복제본에서 DB 인스턴스를 생성할 때 `db.serverless` 클래스를 사용할 수 있습니다. 나중에 복제본을 수정하여 이 크기 조정 구성 속성을 추가할 수도 있습니다. 이 예제의 용량 숫자는 클러스터의 각 서버리스 v2 인스턴스를 2~32개의 Aurora 용량 단위(ACU)로 확장할 수 있도록 합니다. Aurora 서버리스 v2 기능 및 용량 범위를 선택하는 방법에 대한 자세한 내용은 Aurora Serverless v2 사용하기 섹션을 참조하세요.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-2-subnets \
  --db-cluster-identifier cluster-cloned-to-3-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name three-subnets \
  --serverless-v2-scaling-configuration 'MinCapacity=2,MaxCapacity=32'
```

DB 인스턴스에서 사용하는 서브넷 수에 관계없이 소스 클러스터 및 복제본에 대한 Aurora 스토리지는 3개의 AZ와 연결됩니다. 음 예제에는 앞의 예제에서 `restore-db-cluster-to-point-in-time` 명령 모두에 대해 원본 클러스터 및 복제본과 연결된 AZ가 나열되어 있습니다.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-with-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text
```

```
us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-with-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

원본 클러스터와 복제본 클러스터의 각 쌍 간에 적어도 두 개의 AZ가 겹치기 때문에 두 클러스터 모두 동일한 기본 Aurora 스토리지에 액세스할 수 있습니다.

2단계: 복제본의 인스턴스에 대한 네트워크 설정 지정

복제본에 DB 인스턴스를 생성하면 기본적으로 클러스터 자체에서 DB 서브넷 그룹을 상속받습니다. 이렇게 하면 Aurora는 각 인스턴스를 특정 서브넷에 자동으로 할당하고 서브넷과 연결된 AZ에 인스턴스를 만듭니다. 특히 개발 및 테스트 시스템의 경우, 복제본에 새 인스턴스를 추가하는 동안 서브넷 ID나 AZ를 추적할 필요가 없으므로 이 선택이 편리합니다.

다른 방법으로, 복제본에 대한 Aurora DB 인스턴스를 생성할 때 AZ를 지정할 수 있습니다. 지정하는 AZ는 복제본과 연결된 AZ 세트에서 가져와야 합니다. 복제본에 사용하는 DB 서브넷 그룹에 서브넷이 2개만 포함된 경우, 이 두 서브넷과 연결된 AZ 중에서만 선택할 수 있습니다. 이렇게 선택하면 쓰기 인스턴스와 대기 리더 인스턴스가 서로 다른 AZ에 있도록 할 수 있으므로 고가용성 시스템을 위한 유연성과 복원력을 제공합니다. 또는 클러스터에 리더를 추가하는 경우 리더가 3개의 AZ에 분산되도록 할 수 있습니다. 이렇게 하면 드물게 AZ에 장애가 발생하는 경우에도 다른 두 개의 AZ에 여전히 쓰기 인스턴스와 다른 리더 인스턴스가 있습니다.

다음 예제에서는 사용자 정의 DB 서브넷 그룹을 사용하는 프로비저닝된 DB 인스턴스를 복제본 Aurora PostgreSQL 클러스터에 추가합니다.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_postgresql_clone \
  --db-instance-identifier my_postgres_instance \
  --db-subnet-group-name my_new_subnet \
  --engine aurora-postgresql \
  --db-instance-class db.t4g.medium
```

다음 예제는 이러한 명령의 일부 출력을 보여줍니다.

```
{
  'DBInstanceIdentifier': 'my_postgres_instance',
  'DBClusterIdentifier': 'my_aurora_postgresql_clone',
  'DBInstanceClass': 'db.t4g.medium',
  'DBInstanceStatus': 'creating'
  ...
}
```

다음 예제에서는 사용자 정의 DB 서브넷 그룹을 사용하는 Aurora MySQL 복제본에 Aurora 서버리스 v2 DB 인스턴스를 추가합니다. 서버리스 v2 인스턴스를 사용하려면 앞의 예제에서와 같이 `restore-db-cluster-to-point-in-time` 명령에 `--serverless-v2-scaling-configuration` 옵션을 지정해야 합니다.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_mysql_clone \
  --db-instance-identifier my_mysql_instance \
  --db-subnet-group-name my_other_new_subnet \
  --engine aurora-mysql \
  --db-instance-class db.serverless
```

다음 예제는 이러한 명령의 일부 출력을 보여줍니다.

```
{
  'DBInstanceIdentifier': 'my_mysql_instance',
  'DBClusterIdentifier': 'my_aurora_mysql_clone',
  'DBInstanceClass': 'db.serverless',
  'DBInstanceStatus': 'creating'
  ...
}
```

3단계: 클라이언트 시스템에서 복제본으로 연결 설정

이미 클라이언트 시스템에서 Aurora 클러스터에 연결하고 있는 경우, 새 복제본에 동일한 유형의 연결을 허용하고 싶을 수 있습니다. 예를 들어 Amazon Cloud9 인스턴스 또는 EC2 인스턴스에서 원본 클러스터에 연결할 수 있습니다. 동일한 클라이언트 시스템 또는 대상 VPC에서 생성한 새 클라이언트 시스템의 연결을 허용하려면 VPC에서와 동일한 DB 서브넷 그룹 및 VPC 보안 그룹을 설정합니다. 그런 다음 복제본을 생성할 때 서브넷 그룹과 보안 그룹을 지정합니다.

다음 예제에서는 Aurora Serverless v2 복제본을 설정합니다. 이 구성은 DB 클러스터를 생성할 때 `--engine-mode provisioned`와 `--serverless-v2-scaling-configuration`을 조합한 다음 클러스터의 각 DB 인스턴스를 생성할 때 `--db-instance-class db.serverless`를 조합한 것을 기반으로 합니다. `provisioned` 엔진 모드가 기본값이므로 원하는 경우 해당 옵션을 생략할 수 있습니다.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier serverless-sql-postgres\
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-subnet-group-name 'default-vpc-1234' \
  --vpc-security-group-ids 'sg-4567' \
  --serverless-v2-scaling-configuration 'MinCapacity=0.5,MaxCapacity=16' \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

그런 다음 복제본에서 DB 인스턴스를 생성할 때 동일한 `--db-subnet-group-name` 옵션을 지정합니다. 선택적으로 `--availability-zone` 옵션을 포함하여 해당 서브넷 그룹의 서브넷과 연결된 AZ 중 하나를 지정할 수 있습니다. 해당 AZ는 원본 클러스터와 연결된 AZ 중 하나이기도 해야 합니다.

```
aws rds create-db-instance \
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-instance-identifier serverless-sql-postgres-clone-instance \
  --db-instance-class db.serverless \
  --db-subnet-group-name 'default-vpc-987zyx654' \
  --availability-zone 'us-east-1c' \
  --engine aurora-postgresql
```

퍼블릭 서브넷에서 프라이빗 서브넷으로 클러스터 이동

복제본을 사용하여 퍼블릭 서브넷과 프라이빗 서브넷 간에 클러스터를 마이그레이션할 수 있습니다. 애플리케이션을 프로덕션에 배포하기 전에 애플리케이션에 보안 계층을 추가할 때 이 작업을 수행할

수 있습니다. 이 예제에서는 Aurora로 복제본 프로세스를 시작하기 전에 프라이빗 서브넷과 NAT 게이트웨이가 이미 설정되어 있어야 합니다.

Aurora와 관련된 단계는 앞의 예제에서 네트워크 환경에 대한 정보 수집 및 새 네트워크 설정으로 Aurora 복제본 생성에 대한 일반적인 단계와 동일하게 따를 수 있습니다. 가장 큰 차이점은 원본 클러스터의 모든 AZ에 매핑되는 퍼블릭 서브넷이 있더라도 이제 Aurora 클러스터를 위한 프라이빗 서브넷이 충분한지, 그리고 이러한 서브넷이 원본 클러스터의 Aurora 스토리지에 사용되는 것과 동일한 모든 AZ와 연결되어 있는지 확인해야 한다는 것입니다. 다른 복제 사용 사례와 마찬가지로, 필요한 AZ와 연결된 프라이빗 서브넷을 3개 또는 2개로 복제본에 대한 DB 서브넷 그룹을 만들 수 있습니다. 그러나 DB 서브넷 그룹에 2개의 프라이빗 서브넷을 사용하는 경우, 원본 클러스터의 Aurora 스토리지에 사용되는 세 번째 AZ와 연결된 세 번째 프라이빗 서브넷이 있어야 합니다.

이 체크리스트를 참조하여 이러한 유형의 복제본 작업을 수행하기 위한 모든 요구 사항이 갖추어져 있는지 확인할 수 있습니다.

- 원본 클러스터와 연결된 세 개의 AZ를 기록합니다. 지침은 1단계: 원본 클러스터의 가용 영역 확인 단원을 참조하십시오.
- 원본 클러스터의 DB 서브넷 그룹에 있는 퍼블릭 서브넷과 연결된 AZ를 3개 또는 2개 기록합니다. 지침은 3단계: 원본 클러스터의 서브넷 확인 단원을 참조하십시오.
- 원본 클러스터와 연결된 세 개의 AZ 모두에 매핑되는 프라이빗 서브넷을 만듭니다. 또한 프라이빗 서브넷과 통신할 수 있도록 NAT 게이트웨이 생성 등 기타 네트워킹 설정을 수행합니다. 관련 지침은 Amazon Virtual Private Cloud 사용 설명서의 서브넷 생성을 참조하세요.
- 첫 번째 지점의 AZ와 연결된 프라이빗 서브넷 중 3개 또는 2개가 포함된 새 DB 서브넷 그룹을 생성합니다. 지침은 2단계: 복제본에 대한 DB 서브넷 그룹 생성 단원을 참조하십시오.

모든 필수 요건이 갖추어지면 복제본을 만드는 동안 원본 클러스터의 데이터베이스 활동을 일시 중지하고 애플리케이션을 사용하도록 전환할 수 있습니다. 복제본이 만들어지고 연결할 수 있는지, 애플리케이션 코드를 실행할 수 있는지 등을 확인한 후에는 원본 클러스터의 사용을 중단할 수 있습니다.

VPC 간 복제본 생성의 엔드 투 엔드 예제

원본과 다른 VPC에 복제본을 만드는 것은 앞의 예와 동일한 일반 단계를 사용합니다. VPC ID는 서브넷의 속성이므로 실제로 RDS CLI 명령을 실행할 때 파라미터로 VPC ID를 지정하지 않습니다. 가장 큰 차이점은 새 서브넷, 특정 AZ에 매핑된 새 서브넷, VPC 보안 그룹 및 새 DB 서브넷 그룹을 만들어야 할 가능성이 더 높다는 것입니다. 특히 해당 VPC에서 만드는 첫 번째 Aurora 클러스터인 경우 더욱 그렇습니다.

이 체크리스트를 참조하여 이러한 유형의 복제본 작업을 수행하기 위한 모든 요구 사항이 갖추어져 있는지 확인할 수 있습니다.

- 원본 클러스터와 연결된 세 개의 AZ를 기록합니다. 지침은 1단계: 원본 클러스터의 가용 영역 확인 단원을 참조하십시오.
- 원본 클러스터의 DB 서브넷 그룹에 있는 서브넷과 연결된 AZ를 3개 또는 2개 기록합니다. 지침은 2단계: 원본 클러스터의 DB 서브넷 그룹 확인 단원을 참조하십시오.
- 원본 클러스터와 연결된 세 개의 AZ 모두에 매핑되는 서브넷을 만듭니다. 지침은 1단계: 복제본의 서브넷 생성 단원을 참조하십시오.
- 클라이언트 시스템, 애플리케이션 서버 등이 복제본의 DB 인스턴스와 통신할 수 있도록 VPC 보안 그룹 설정 등 기타 네트워킹 설정을 수행합니다. 지침은 보안 그룹을 통한 액세스 제어 단원을 참조하십시오.
- 첫 번째 지점의 AZ와 연결된 서브넷 중 3개 또는 2개의 서브넷을 포함하는 새 DB 서브넷 그룹을 만듭니다. 지침은 2단계: 복제본에 대한 DB 서브넷 그룹 생성 단원을 참조하십시오.

모든 필수 요건이 갖추어지면 복제본을 만드는 동안 원본 클러스터의 데이터베이스 활동을 일시 중지하고 애플리케이션을 사용하도록 전환할 수 있습니다. 복제본이 생성되고 연결, 애플리케이션 코드 실행 등이 가능한지 확인한 후에는 원본과 복제본을 모두 계속 실행할지, 아니면 원본 클러스터의 사용을 중단할지 고려할 수 있습니다.

다음 Linux 예제는 한 VPC에서 다른 VPC로 Aurora DB 클러스터를 복제본으로 생성하기 위한 AWS CLI 작업 순서를 보여줍니다. 예시와 관련이 없는 일부 필드는 명령 출력에 표시되지 않습니다.

먼저 소스 및 대상 VPC의 ID를 확인합니다. VPC를 생성할 때 할당하는 설명적인 이름은 VPC 메타데이터에 태그로 표시됩니다.

```
$ aws ec2 describe-vpcs --query '*[].[VpcId,Tags]'
[
    [
        'vpc-0f0c0fc0000b0ffb0',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-source'
            }
        ]
    ],
    [
        'vpc-9e99d9f99a999bd99',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-dest'
```

```
            }
        ]
    ]
]
```

원본 클러스터는 이미 소스 VPC에 존재합니다. Aurora 스토리지에 대해 동일한 AZ 세트를 사용하여 복제본을 설정하려면 원본 클러스터에서 사용하는 AZ를 확인합니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier original-cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

원래 클러스터에서 사용하는 AZ에 해당하는 서브넷이 있는지 확인합니다(`us-east-1c`, `us-east-1d`, `us-east-1f`).

```
$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
  --availability-zone us-east-1c --cidr-block 10.0.0.128/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1c',
        'SubnetId': 'subnet-3333a33be3ef3e333',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1d --cidr-block 10.0.0.160/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1d',
        'SubnetId': 'subnet-4eeb444cd44b4d444',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1f --cidr-block 10.0.0.224/28
{
    'Subnet': {
```

```
        'AvailabilityZone': 'us-east-1f',
        'SubnetId': 'subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}
```

이 예제에서는 대상 VPC에 필요한 AZ에 매핑되는 서브넷이 있는지 확인합니다.

```
aws ec2 describe-subnets --query 'sort_by(*[] | [?VpcId == `vpc-9e99d9f99a999bd99`] |
[].{SubnetId:SubnetId,VpcId:VpcId,AvailabilityZone:AvailabilityZone},
 &AvailabilityZone)' --output table

---------------------------------------------------------------------------
|                              DescribeSubnets                            |
+------------------+----------------------------+-------------------------+
| AvailabilityZone |          SubnetId          |          VpcId          |
+------------------+----------------------------+-------------------------+
|  us-east-1a      |  subnet-000ff0e00000c0aea  |  vpc-9e99d9f99a999bd99  |
|  us-east-1b      |  subnet-1111d111111ca11b1  |  vpc-9e99d9f99a999bd99  |
|  us-east-1c      | subnet-3333a33be3ef3e333   |  vpc-9e99d9f99a999bd99  |
|  us-east-1d      | subnet-4eeb444cd44b4d444   |  vpc-9e99d9f99a999bd99  |
|  us-east-1f      | subnet-66eea6666fb66d66c   |  vpc-9e99d9f99a999bd99  |
+------------------+----------------------------+-------------------------+
```

VPC에서 Aurora DB 클러스터를 생성하기 전에 Aurora 스토리지에 사용되는 AZ에 매핑되는 서브넷이 있는 DB 서브넷 그룹이 있어야 합니다. 일반 클러스터를 생성할 때는 3개의 AZ 중 원하는 세트를 사용할 수 있습니다. 기존 클러스터를 복제본으로 생성하는 경우, 서브넷 그룹은 Aurora 스토리지에 사용하는 3개의 AZ 중 2개 이상과 일치해야 합니다.

```
$ aws rds create-db-subnet-group \
  --db-subnet-group-name subnet-group-in-other-vpc \
  --subnet-ids
 '["subnet-3333a33be3ef3e333","subnet-4eeb444cd44b4d444","subnet-66eea6666fb66d66c"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c'

{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'subnet-group-in-other-vpc',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
```

```
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'subnet-4eeb444cd44b4d444',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1d' }
            },
            {
                'SubnetIdentifier': 'subnet-3333a33be3ef3e333',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1c' }
            },
            {
                'SubnetIdentifier': 'subnet-66eea6666fb66d66c',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1f' }
            }
        ]
    }
}
```

이제 서브넷과 DB 서브넷 그룹이 준비되었습니다. 다음 예제에서는 클러스터를 복제본으로 복원하는 `restore-db-cluster-to-point-in-time`을 보여줍니다. `--db-subnet-group-name` 옵션은 원본 클러스터의 올바른 AZ 세트에 매핑되는 올바른 서브넷 세트와 복제본을 연결합니다.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier original-cluster \
  --db-cluster-identifier clone-in-other-vpc \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name subnet-group-in-other-vpc

{
  'DBClusterIdentifier': 'clone-in-other-vpc',
  'DBSubnetGroup': 'subnet-group-in-other-vpc',
  'Engine': 'aurora-postgresql',
  'EngineVersion': '15.4',
  'Status': 'creating',
  'Endpoint': 'clone-in-other-vpc.cluster-c0abcdef.us-east-1.rds.amazonaws.com'
}
```

다음 예제에서는 복제본의 Aurora 스토리지가 원본 클러스터와 동일한 AZ 세트를 사용하는 것을 확인합니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier clone-in-other-vpc \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text
```

```
us-east-1c
us-east-1d
us-east-1f
```

이 시점에서 복제본에 대한 DB 인스턴스를 생성할 수 있습니다. 각 인스턴스와 연결된 VPC 보안 그룹이 대상 VPC에 있는 EC2 인스턴스, 애플리케이션 서버 등에 사용하는 IP 주소 범위의 연결을 허용하는지 확인합니다.

AWS RAM과 Amazon Aurora에서 교차 계정 복제

Amazon Aurora에서 AWS Resource Access Manager(AWS RAM)를 사용하여 사용자 AWS 계정에 속한 Aurora DB 클러스터와 복제본을 다른 AWS 계정 또는 조직과 공유할 수 있습니다. 이러한 교차 계정 복제는 데이터베이스 스냅샷을 생성하여 복원하는 것보다 훨씬 빠릅니다. Aurora DB 클러스터 중 하나의 복제본을 생성하고 복제본을 공유할 수 있습니다. 또는 Aurora DB 클러스터를 다른 AWS 계정과 공유하고 계정 소유자가 복제본을 생성하도록 합니다. 선택한 접근 방식은 사용 사례에 따라 다릅니다.

예를 들어 재무 데이터베이스의 복제본을 조직의 내부 감사 팀과 정기적으로 공유해야 할 수 있습니다. 이 경우 감사 팀은 사용하는 애플리케이션에 대한 자체 AWS 계정을 갖게 됩니다. 감사 팀의 AWS 계정에 Aurora DB 클러스터에 액세스하고 필요에 따라 복제할 수 있는 권한을 부여할 수 있습니다.

반면 외부 공급업체가 재무 데이터를 감사하는 경우 복제본을 직접 생성하는 것이 좋습니다. 그런 다음 외부 공급업체에게 복제본에 대한 액세스 권한만 부여합니다.

교차 계정 복제를 사용하여 개발 및 테스트와 같은 동일한 AWS 계정 내의 복제에 대해 동일한 사용 사례를 지원할 수 있습니다. 예를 들어 조직에서 프로덕션, 개발 및 테스트 등에 대해 다른 AWS 계정을 사용할 수 있습니다. 자세한 내용은 Aurora 복제 개요 섹션을 참조하세요.

따라서 복제본을 다른 AWS 계정과 공유하거나 다른 AWS 계정에서 Aurora DB 클러스터의 복제본을 만들 수 있도록 허용할 수 있습니다. 두 경우 모두 AWS RAM을 사용하여 공유 객체를 생성합니다. AWS 계정 간의 AWS 리소스 공유에 대한 전체 내용은 AWS RAM 사용 설명서를 참조하세요.

계정 간 복제본을 만들려면 원본 클러스터를 소유하고 있는 AWS 계정과 복제본을 생성하는 AWS 계정에서 몇 가지 작업이 필요합니다. 먼저 원본 클러스터 소유자가 클러스터를 수정하여 하나 이상의 계정에서 클러스터를 복제할 수 있도록 합니다. 계정 중 하나라도 다른 AWS 조직에 있으면 AWS는 공유 초대를 생성합니다. 다른 계정은 진행하기 전에 초대를 수락해야 합니다. 그러면 승인된 계정에서 클러스터를 복제할 수 있습니다. 이러한 프로세스 전체에서 클러스터는 고유한 Amazon 리소스 이름(ARN)으로 식별합니다.

동일한 AWS 계정 내의 복제와 마찬가지로 소스 또는 복제본에서 데이터를 변경한 경우에만 추가 스토리지 공간이 사용됩니다. 그런 다음 해당 시점에 스토리지 요금이 적용됩니다. 원본 클러스터가 삭제되면 스토리지 비용이 나머지 복제된 클러스터로 동일하게 배분됩니다.

주제

계정 간 복제 제한 사항

Aurora 계정 간 복제는 다음과 같은 제한 사항이 있습니다.

- Aurora Serverless v1 클러스터는 AWS 계정 간 복제가 불가능합니다.
- AWS Management Console을 사용하여 공유 리소스에 대한 초대를 보거나 수락할 수 없습니다. AWS CLI, Amazon RDS API 또는 AWS RAM 콘솔을 사용하여 공유 리소스에 대한 초대를 보고 수락할 수 있습니다.
- 새 복제본은 AWS 계정과 공유 중인 복제본에서만 생성할 수 없습니다.
- AWS 계정과 공유 중인 리소스(복제본 또는 Aurora DB 클러스터)를 공유할 수 없습니다.
- 단일 Aurora DB 클러스터에서 최대 15개의 교차 계정 복제본을 생성할 수 있습니다.
- 이 15개의 교차 계정 복제본은 각각 다른 AWS 계정에서 소유해야 합니다. 즉, AWS 계정 내 클러스터의 교차 계정 복제본을 하나만 생성할 수 있습니다.
- 클러스터를 복제한 후 교차 계정 복제본에 대한 제한을 적용하기 위해 원래 클러스터와 복제본이 동일한 것으로 간주됩니다. 동일한 AWS 계정 내에서 원래 클러스터와 복제된 클러스터의 교차 계정 복제본을 생성할 수 없습니다. 원래 클러스터와 해당 복제본의 총 교차 계정 복제본 수는 15개를 초과할 수 없습니다.
- 클러스터가 `ACTIVE` 상태에 있을 때만 Aurora DB 클러스터를 다른 AWS 계정과 공유할 수 있습니다.
- 다른 AWS 계정과 공유 중인 Aurora DB 클러스터의 이름은 변경할 수 없습니다.
- 클러스터가 기본 RDS 키로 암호화되어 있으면 계정 간 복제본을 생성할 수 없습니다.
- 다른 AWS 계정이 공유한 암호화된 Aurora DB 클러스터에서는 하나의 AWS 계정에 암호화되지 않은 복제본을 만들 수 없습니다. 또한 클러스터 소유자는 소스 클러스터의 AWS KMS key에 액세스할 수 있는 권한을 부여해야 합니다. 하지만 복제본을 생성할 때는 다른 키를 사용해도 됩니다.

다른 AWS 계정에서 클러스터를 복제하도록 허용

다른 AWS 계정에서 자신이 소유한 클러스터를 복제하도록 허용하려면 AWS RAM을 사용해 공유 권한을 설정합니다. 이때 다른 AWS 조직에 속한 나머지 계정 모두에게 초대장을 전송합니다.

AWS RAM 콘솔에서 자신이 소유한 리소스를 공유하는 방법에 대한 자세한 내용은 AWS RAM 사용 설명서에서 [자신이 소유한 리소스 공유]를 참조하세요.

주제

- [다른 AWS 계정에 클러스터를 복제할 수 있는 권한 부여]
- [소유하고 있는 클러스터가 다른 AWS 계정과 공유하는지 확인]

다른 AWS 계정에 클러스터를 복제할 수 있는 권한 부여

공유하고 있는 클러스터가 암호화되어 있으면 해당 클러스터의 AWS KMS key도 공유합니다. 한 AWS 계정의 AWS Identity and Access Management(IAM) 사용자 또는 역할이 다른 계정의 KMS 키를 사용하도록 허용할 수 있습니다.

이를 위해서는 먼저 외부 계정(루트 사용자)을 AWS KMS을(를) 통해 KMS 키의 키 정책에 추가합니다. 개별 사용자 또는 역할이 아니고 사용자 또는 역할을 소유한 외부 계정을 추가하는 것입니다. 또한 사용자가 생성하는 KMS 키만 공유할 수 있으며, 기본 RDS 서비스 키는 공유하지 못합니다. KMS 키 액세스 제어에 대한 자세한 내용은 [AWS KMS에 대한 인증 및 액세스 제어]를 참조하세요.

콘솔

클러스터를 복제할 수 있는 권한을 부여하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 세부 정보 페이지를 볼 수 있도록 공유할 DB 클러스터와 Connectivity & security(연결 및 보안) 탭을 차례대로 선택합니다.
4. 다른 AWS 계정과 DB 클러스터 공유 섹션에서 해당 클러스터의 복제를 허용할 AWS 계정의 숫자 계정 ID를 입력합니다. 동일한 조직의 계정 ID라면 입력란에 입력을 시작한 후 메뉴에서 선택할 수 있습니다.

> **Important**
>
> 경우에 따라 자신의 계정과 다른 AWS 조직에 속하는 계정에게 클러스터 복제를 허용해야 할 수도 있습니다. 이때는 보안을 이유로 콘솔은 해당 계정 ID의 소유자 또는 계정 존재 여부를 보고하지 않습니다.
> 자신의 AWS 계정과 다른 AWS 조직에 속하는 계정 숫자를 입력할 때는 주의해야 합니다. 원하는 계정과의 공유 여부를 바로 확인하십시오.

5. 확인 페이지에서 지정한 계정 ID가 정확한지 확인합니다. 확인 입력란에 `share`를 입력하여 확인합니다.

   [세부 정보(Details)] 페이지에서 [이 DB 클러스터를 공유하는 계정(Accounts that this DB cluster is shared with)] 아래 지정된 AWS 계정 ID를 보여 주는 항목이 나타납니다. 상태 열이 처음에는 대기 중으로 표시됩니다.

6. 해당 AWS 계정 소유자에게 연락하거나, 혹은 두 계정을 모두 소유하고 있다면 해당 계정에 로그인합니다. 해당 계정 소유자에게 공유 초대장을 승인한 후 다음과 같이 DB 클러스터를 복제하도록 지시합니다.

AWS CLI

클러스터를 복제할 수 있는 권한을 부여하려면

1. 필요한 파라미터 정보를 수집합니다. 클러스터 ARN과 다른 AWS 계정의 숫자 ID가 필요합니다.
2. AWS RAM CLI 명령인 [`create-resource-share`]를 실행합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws ram create-resource-share --name descriptive_name \
     --region region \
     --resource-arns cluster_arn \
     --principals other_account_ids
   ```

   Windows의 경우:

   ```
   aws ram create-resource-share --name descriptive_name ^
     --region region ^
     --resource-arns cluster_arn ^
   ```

```
    --principals other_account_ids
```

--principals 파라미터에서 다수의 계정 ID를 추가하려면 공백으로 각 ID를 구분하십시오. 자신의 AWS 조직 외부에 존재하는 계정 ID의 허용 여부를 지정하려면 --allow-external-principals에서 --no-allow-external-principals 또는 create-resource-share 파라미터를 추가합니다.

AWS RAM API

클러스터를 복제할 수 있는 권한을 부여하려면

1. 필요한 파라미터 정보를 수집합니다. 클러스터 ARN과 다른 AWS 계정의 숫자 ID가 필요합니다.
2. AWS RAM API 작업인 CreateResourceShare를 호출한 후 다음 값을 지정합니다.
   - 하나 이상의 AWS 계정에 대한 계정 ID를 principals 파라미터로 지정합니다.
   - 하나 이상의 Aurora DB 클러스터의 ARN을 resourceArns 파라미터로 지정합니다.
   - allowExternalPrincipals 파라미터에 부울 값을 추가하여 자신의 AWS 조직 외부에 존재하는 계정 ID의 허용 여부를 지정합니다.

기본 RDS 키를 사용하는 클러스터 재생성

공유하려는 암호화된 클러스터가 기본 RDS 키를 사용하는 경우 클러스터를 재생성해야 합니다. 이렇게 하려면 DB 클러스터의 수동 스냅샷을 생성하고 AWS KMS key을(를) 사용한 다음 클러스터를 새 클러스터로 복원합니다. 그런 다음 새 클러스터를 공유합니다. 이 프로세스를 수행하려면 다음 단계를 수행합니다.

기본 RDS 키를 사용해 암호화된 클러스터를 재생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.
3. 자신의 스냅샷을 선택합니다.
4. 작업에서 스냅샷 복사와 암호화 활성화를 차례대로 선택합니다.
5. AWS KMS key에서 새롭게 사용할 암호화 키를 선택합니다.
6. 복사된 스냅샷을 복원합니다. 이 작업을 수행하려면 의 절차를 수행합니다DB 클러스터 스냅샷에서 복원 새로운 DB 인스턴스는 새로운 암호화 키를 사용합니다.

7. (선택 사항) 더 이상 필요 없다면 이전 DB 클러스터를 삭제합니다. 이 작업을 수행하려면 의 절차를 수행합니다DB 클러스터 스냅샷 삭제 단, 삭제 전에 새로운 클러스터에 필요한 데이터가 모두 있는지, 그리고 애플리케이션이 새로운 클러스터에 성공적으로 액세스하는지 확인하십시오.

소유하고 있는 클러스터가 다른 AWS 계정과 공유하는지 확인

다른 사용자에게 클러스터 공유 권한이 있는지 확인할 수 있습니다. 따라서 클러스터가 계정 간 최대 복제 수 제한에 접근하는지 파악하는 데 효과적입니다.

AWS RAM 콘솔을 사용하여 리소스를 공유하는 절차는 AWS RAM 사용 설명서에서 자신이 소유한 리소스 공유를 참조하세요.

AWS CLI

소유한 클러스터가 다른 AWS 계정과 공유되는지 확인하려면 다음을 수행하세요.

- 자신의 계정 ID를 리소스 소유자로, 그리고 클러스터 ARN을 리소스 ARN으로 사용하여 AWS RAM CLI 명령인 `list-principals`를 호출합니다. 다음 명령을 사용해 모든 공유 내역을 확인할 수 있습니다. 명령을 실행한 결과에서 클러스터 복제가 허용되는 AWS 계정을 알 수 있습니다.

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

소유한 클러스터가 다른 AWS 계정과 공유되는지 확인하려면 다음을 수행하세요.

- AWS RAM API 작업 ListPrincipals를 호출합니다. 자신의 계정 ID를 리소스 소유자로, 그리고 클러스터 ARN을 리소스 ARN으로 사용합니다.

다른 AWS 계정에서 소유하고 있는 클러스터 복제

다른 AWS 계정에서 소유한 클러스터를 복제하려면 AWS RAM을 사용해 복제 권한을 획득해야 합니다. 필요한 권한을 가져온 후에는 표준 절차에 따라 Aurora 클러스터를 복제합니다.

또한 현재 소유하고 있는 클러스터가 다른 AWS 계정에서 소유하고 있는 클러스터의 복제본인지 알 수도 있습니다.

AWS RAM 콘솔에서 다른 계정이 소유하고 있는 리소스를 사용해 작업하는 방법은 AWS RAM 사용 설명서에서 공유 리소스에 대한 액세스를 참조하세요.

주제

다른 AWS 계정에서 소유하고 있는 클러스터 복제 초대장 보기

다른 AWS 조직의 AWS 계정에서 소유하고 있는 클러스터에 대한 복제 초대장 작업을 할 때는 AWS CLI, AWS RAM 콘솔 또는 AWS RAM API를 사용합니다. 현재 Amazon RDS 콘솔에서는 이러한 절차를 수행할 수 없습니다.

AWS RAM 콘솔에서 초대장 작업에 대한 절차는 AWS RAM 사용 설명서에서 공유 리소스에 대한 액세스를 참조하세요.

AWS CLI

다른 AWS 계정에서 소유하고 있는 클러스터 복제 초대장을 보려면

1. AWS RAM CLI 명령인 `get-resource-share-invitations`를 실행합니다.

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   위 명령을 실행한 결과에서 이전에 승인하거나 거부한 초대장을 포함해 클러스터 복제 초대장을 모두 볼 수 있습니다.

2. (선택 사항) 자신의 작업이 필요한 초대장만 표시되도록 목록을 필터링할 수 있습니다. 파라미터로 `--query 'resourceShareInvitations[?status==`PENDING`]'`를 추가하기만 하면 됩니다.

AWS RAM API

다른 AWS 계정에서 소유하고 있는 클러스터 복제 초대장을 보려면

1. AWS RAM API 작업 `GetResourceShareInvitations`를 호출합니다. 그러면 이전에 승인하거나 거부한 초대장을 포함해 해당하는 초대장이 모두 반환됩니다.

2. (선택 사항) `resourceShareAssociations` 반환 필드에서 `status` 값의 `PENDING` 유무를 확인하여 자신의 작업이 필요한 초대장만 찾아볼 수 있습니다.

다른 AWS 계정에서 소유하고 있는 클러스터에 대한 공유 초대장 승인

다른 AWS 조직의 AWS 계정에서 소유하고 있는 클러스터에 대한 공유 초대장을 승인할 수 있습니다. 해당 초대장 작업은 AWS CLI, AWS RAM 및 RDS API 또는 AWS RAM 콘솔을 사용합니다. 현재 RDS 콘솔에서는 이러한 절차를 수행할 수 없습니다.

AWS RAM 콘솔에서 초대장 작업에 대한 절차는 AWS RAM 사용 설명서에서 공유 리소스에 대한 액세스를 참조하세요.

AWS CLI

다른 AWS 계정에서 클러스터 공유 초대장을 승인하려면

1. 앞에서 한 것처럼 AWS RAM CLI 명령인 `get-resource-share-invitations`를 실행하여 초대장 ARN을 찾습니다.
2. 다음과 같이 AWS RAM CLI 명령인 `accept-resource-share-invitation`을 호출하여 초대장을 승인합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   Windows의 경우:

   ```
   aws ram accept-resource-share-invitation ^
     --resource-share-invitation-arn invitation_arn ^
     --region region
   ```

AWS RAM 및 RDS API

다른 사용자의 클러스터 공유 초대장을 승인하려면

1. 앞에서 한 것처럼 AWS RAM API 작업 `GetResourceShareInvitations`를 호출하여 초대장 ARN을 찾습니다.

2. 해당 ARN을 `resourceShareInvitationArn` 파라미터로 RDS API 작업인 AcceptResourceShareInvitation에게 전달합니다.

다른 AWS 계정에서 소유하고 있는 Aurora 클러스터 복제

위와 같이 DB 클러스터를 소유하고 있는 AWS 계정의 초대장을 승인하였다면 이제 클러스터를 복제할 수 있습니다.

콘솔

다른 AWS 계정에서 소유하고 있는 Aurora 클러스터를 복제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

   데이터베이스 목록 상단에서 역할 값이 `Shared from account #account_id`인 항목이 1개 이상 표시됩니다. 보안을 이유로 직접 확인할 수 있는 원본 클러스터 정보는 제한적입니다. 확인할 수 있는 속성은 복제 클러스터에서 동일해야 하는 데이터베이스 엔진 및 버전 등입니다.
3. 복제할 클러스터를 선택합니다.
4. 작업에서 복제본 생성을 선택합니다.
5. 콘솔 단원의 절차에 따라 복제 클러스터의 설정을 마칩니다.
6. 필요하다면 복제 클러스터의 암호화를 활성화합니다. 복제할 클러스터가 암호화되어 있다면 복제 클러스터에서도 암호화를 활성화해야 합니다. 사용자와 클러스터를 공유한 AWS 계정 역시 클러스터를 암호화하는 데 사용된 KMS 키를 공유해야 합니다. 복제본을 암호화할 때는 동일한 KMS 키를 사용하거나, 사용자 고유의 KMS 키를 사용해도 좋습니다. 클러스터가 기본 KMS 키로 암호화되어 있으면 계정 간 복제본을 생성할 수 없습니다.

   암호화 키를 소유한 계정은 키 정책에 따라 대상 계정에게 키를 사용할 수 있는 권한을 부여해야 합니다. 이러한 프로세스는 대상 계정에게 키 사용 권한을 부여하는 키 정책에 따라 암호화된 스냅샷을 공유하는 방법과 비슷합니다.

AWS CLI

다른 AWS 계정에서 소유하고 있는 Aurora 클러스터를 복제하려면

1. 위와 같이 DB 클러스터를 소유하고 있는 AWS 계정의 초대장을 승인합니다.

2. 다음과 같이 RDS CLI 명령인 `source-db-cluster-identifier`의 `restore-db-cluster-to-point-in-time` 파라미터에서 원본 클러스터의 전체 ARN을 지정하여 클러스터를 복제합니다.

   `source-db-cluster-identifier`로 전달된 ARN을 아직 공유하지 않았다면 마치 지정된 클러스터가 존재하지 않는 것처럼 동일한 오류 메시지가 반환됩니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time
```

   Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time
```

3. 복제할 클러스터가 암호화되어 있으면 `kms-key-id` 파라미터를 추가하여 복제 클러스터도 암호화합니다. `kms-key-id` 값은 원본 DB 클러스터를 암호화할 때 사용한 것과 동일하거나, 혹은 사용자 고유의 KMS 키가 될 수도 있습니다. 단, 암호화 키를 사용할 수 있는 권한이 계정에게 있어야 합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time \
  --kms-key-id=arn:aws:kms:arn_details
```

   Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
```

```
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

암호화 키를 소유한 계정은 키 정책에 따라 대상 계정에게 키를 사용할 수 있는 권한을 부여해야 합니다. 이러한 프로세스는 대상 계정에게 키 사용 권한을 부여하는 키 정책에 따라 암호화된 스냅샷을 공유하는 방법과 비슷합니다. 키 정책 예제는 다음과 같습니다.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
```

```
        "Resource": "*",
        "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
      }
    ]
}
```

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 명령은 해당 DB 클러스터의 DB 인스턴스가 아닌 DB 클러스터만 복원합니다. 복원된 DB 클러스터에 대한 DB 인스턴스를 생성하려면 create-db-instance 명령을 호출합니다. `--db-cluster-identifier`에서 복원된 DB 클러스터의 식별자를 지정합니다. `restore-db-cluster-to-point-in-time` 명령이 완료되고 DB 클러스터를 사용 가능할 때만 DB 인스턴스를 생성할 수 있습니다.

RDS API

다른 AWS 계정에서 소유하고 있는 Aurora 클러스터를 복제하려면

1. 위와 같이 DB 클러스터를 소유하고 있는 AWS 계정의 초대장을 승인합니다.
2. RDS API 작업인 `SourceDBClusterIdentifier`의 `RestoreDBClusterToPointInTime` 파라미터에서 원본 클러스터의 전체 ARN을 지정하여 클러스터를 복제합니다.

   `SourceDBClusterIdentifier`로 전달된 ARN을 아직 공유하지 않았다면 마치 지정된 클러스터가 존재하지 않는 것처럼 동일한 오류 메시지가 반환됩니다.
3. 복제할 클러스터가 암호화되어 있으면 `KmsKeyId` 파라미터를 추가하여 복제 클러스터도 암호화합니다. `kms-key-id` 값은 원본 DB 클러스터를 암호화할 때 사용한 것과 동일하거나, 혹은 사용자 고유의 KMS 키가 될 수도 있습니다. 단, 암호화 키를 사용할 수 있는 권한이 계정에게 있어야 합니다.

   볼륨을 복제할 때는 원본 클러스터를 암호화할 때 사용한 암호화 키를 사용할 수 있는 권한이 대상 계정에게 필요합니다. Aurora는 `KmsKeyId`에 지정된 암호화 키를 사용하여 새로 복제된 클러스터를 암호화합니다.

   암호화 키를 소유한 계정은 키 정책에 따라 대상 계정에게 키를 사용할 수 있는 권한을 부여해야 합니다. 이러한 프로세스는 대상 계정에게 키 사용 권한을 부여하는 키 정책에 따라 암호화된 스냅샷을 공유하는 방법과 비슷합니다. 키 정책 예제는 다음과 같습니다.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

Note

RestoreDBClusterToPointInTime RDS API 작업은 DB 클러스터만 복원하고, 해당 DB 클러스터의 DB 인스턴스는 복원하지 않습니다. RDS API 작업 CreateDBInstance를 호출하여 복원된 DB 클러스터의 기본 인스턴스를 만듭니다. `DBClusterIdentifier`에서 복원된 DB 클러스터의 식별자를 지정합니다. `RestoreDBClusterToPointInTime` 작업이 완료되고 DB 클러스터를 사용할 수 있어야만 DB 인스턴스를 생성할 수 있습니다.

DB 클러스터의 계정 간 복제본 여부 확인

`DBClusters` 객체는 각 클러스터의 계정 간 복제본 여부를 식별합니다. RDS CLI 명령인 `include-shared`를 실행할 때 `describe-db-clusters` 옵션을 사용해 자신에게 복제 권한이 있는 클러스터를 알아볼 수 있습니다. 하지만 해당 클러스터의 구성 세부 정보는 대부분 볼 수 없습니다.

AWS CLI

DB 클러스터가 교차 계정 복제인지 확인하려면

- RDS CLI 명령인 `describe-db-clusters`를 호출합니다.

  다음 예제는 실제 또는 잠재적 계정 간 복제 DB 클러스터가 `describe-db-clusters` 출력 시 어떻게 표시되는지 나타낸 것입니다. AWS 계정에서 소유하고 있는 기존 계정이라면 `CrossAccountClone` 필드에서 클러스터가 다른 AWS 계정에서 소유하고 있는 DB 클러스터의 복제본인지 알 수 있습니다.

  경우에 따라 AWS 필드에 자신과 다른 `DBClusterArn` 계정 번호의 항목이 존재할 수 있습니다. 이러한 경우 해당 항목은 다른 AWS 계정에서 소유하고 있지만 자신이 복제할 수 있는 클러스터라는 것을 의미합니다. 이러한 항목은 `DBClusterArn` 외에 다른 필드가 거의 없습니다. 복제 클러스터를 생성할 때는 `StorageEncrypted`, `Engine` 및 `EngineVersion` 값을 원본 클러스터와 동일하게 지정합니다.

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
```

```
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0
  ]
}
```

RDS API

DB 클러스터가 교차 계정 복제인지 확인하려면

- RDS API 작업인 DescribeDBClusters를 호출합니다.

  AWS 계정에서 소유하고 있는 기존 계정이라면 `CrossAccountClone` 필드에서 클러스터가 다른 AWS 계정에서 소유하고 있는 DB 클러스터의 복제본인지 알 수 있습니다. AWS 필드에서 `DBClusterArn` 계정 번호가 다른 항목은 복제가 가능하지만 다른 AWS 계정에서 소유하고 있는 클러스터라는 것을 의미합니다. 이러한 항목은 `DBClusterArn` 외에 다른 필드가 거의 없습니다. 복제 클러스터를 생성할 때는 `StorageEncrypted`, `Engine` 및 `EngineVersion` 값을 원본 클러스터와 동일하게 지정합니다.

  다음 예제는 실제 복제 클러스터와 잠재적 복제 클러스터를 모두 시연한 반환 값을 나타낸 것입니다.

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
```

```
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
  ]
}
```

Aurora를 다른 AWS 서비스와 통합

Aurora DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora를 다른 AWS 서비스와 통합합니다.

주제

AWS 서비스를 Amazon Aurora MySQL과 통합

Aurora MySQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora MySQL이 다른 AWS 서비스와 통합되었습니다. Aurora MySQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- AWS Lambda 또는 `lambda_sync` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 `lambda_async` 함수를 호출하십시오. 또는 AWS Lambda 프로시저를 사용하여 비동기적으로 `mysql.lambda_async` 함수를 호출하십시오.
- `LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 명령을 사용하여 Amazon S3 버킷에 저장된 텍스트 또는 XML 파일에서 DB 클러스터로 데이터를 로드하십시오.
- `SELECT INTO OUTFILE S3` 명령을 사용하여 DB 클러스터의 데이터를 Amazon S3 버킷에 저장된 텍스트 파일에 저장하십시오.
- Application Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거하십시오. 자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 섹션을 참조하세요.

Aurora MySQL과 다른 AWS 서비스의 통합에 대한 자세한 내용은 Amazon Aurora MySQL을 다른 AWS 서비스와 통합 섹션을 참조하세요.

AWS 서비스를 Amazon Aurora PostgreSQL과 통합

Aurora PostgreSQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora PostgreSQL이 다른 AWS 서비스와 통합되었습니다. Aurora PostgreSQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- Performance Insights에서 관계형 데이터베이스 워크로드를 빠르게 수집하여 살펴보고 성능을 평가합니다.

- Aurora Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거합니다. 자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 섹션을 참조하세요.

Aurora PostgreSQL과 다른 AWS 서비스의 통합에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 다른 AWS 서비스와 통합 섹션을 참조하세요.

Amazon Aurora DB 클러스터 유지 관리

Amazon RDS는 Amazon RDS 리소스를 정기적으로 유지 관리합니다.

주제

- DB 클러스터 유지 관리 업데이트 개요
- 보류 중인 유지 관리 업데이트 보기
- DB 클러스터에 업데이트 적용
- Amazon RDS 유지 관리 기간
- 기본 DB 클러스터 유지 관리 기간 조정
- Aurora DB 클러스터 마이너 버전 자동 업그레이드
- Aurora MySQL 유지 관리 업데이트 빈도 선택

DB 클러스터 유지 관리 업데이트 개요

유지 관리에는 주로 DB 클러스터의 다음 리소스에 대한 업데이트가 포함됩니다.

- 기본 하드웨어
- 기본 운영 체제(OS)
- 데이터베이스 엔진 버전

운영 체제 업데이트는 보안상 가장 빈번하게 발생하며 가능한 한 빨리 수행하는 것이 좋습니다. 운영 체제 업데이트에 대한 자세한 내용은 DB 클러스터에 업데이트 적용 섹션을 참조하세요.

주제

- 유지 관리 업데이트 중 오프라인 리소스
- 연기된 DB 인스턴스 및 DB 클러스터 수정 사항
- DescribePendingMaintenanceActions API의 최종 일관성

유지 관리 업데이트 중 오프라인 리소스

일부 유지 관리 항목을 사용하려면 Amazon RDS에서 DB 클러스터를 잠시 동안 오프라인 상태로 전환해야 합니다. 리소스가 오프라인 상태에 있어야 하는 유지 관리 항목에는 필수 운영 체제 또는 데이터베이스 패칭이 포함됩니다. 이때 보안 및 인스턴스 안정성과 관련된 패치에 한해 필수 패치 작업으로

자동 예약됩니다. 이러한 패치 작업은 드물게 발생하며 일반적인 빈도는 몇 개월에 한 번입니다. 대부분 유지 관리 기간의 일부만 필요합니다.

연기된 DB 인스턴스 및 DB 클러스터 수정 사항

즉시 적용되지 않도록 연기된 DB 클러스터 및 인스턴스 수정 사항은 유지 관리 기간에 적용됩니다. 예를 들어 유지 관리 기간에 DB 인스턴스 클래스나 클러스터 또는 DB 파라미터 그룹을 변경하도록 선택할 수 있습니다. 대기 중인 재부팅 설정을 사용하여 지정한 수정 사항은 대기 중인 유지 관리 목록에 표시되지 않습니다. DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

다음 유지 관리 기간에 보류 중인 수정 사항을 보려면 describe-db-clusters AWS CLI 명령을 사용하고 `PendingModifiedValues` 필드를 확인합니다.

DescribePendingMaintenanceActions API의 최종 일관성

Amazon RDS `DescribePendingMaintenanceActions` API는 최종 일관성 모델을 따릅니다. 즉, `DescribePendingMaintenanceActions` 명령의 결과가 모든 후속 RDS 명령에 즉시 표시되지 않을 수 있습니다. 이전 API 명령을 사용한 즉시 `DescribePendingMaintenanceActions`를 사용할 때는 이 점을 염두에 두세요.

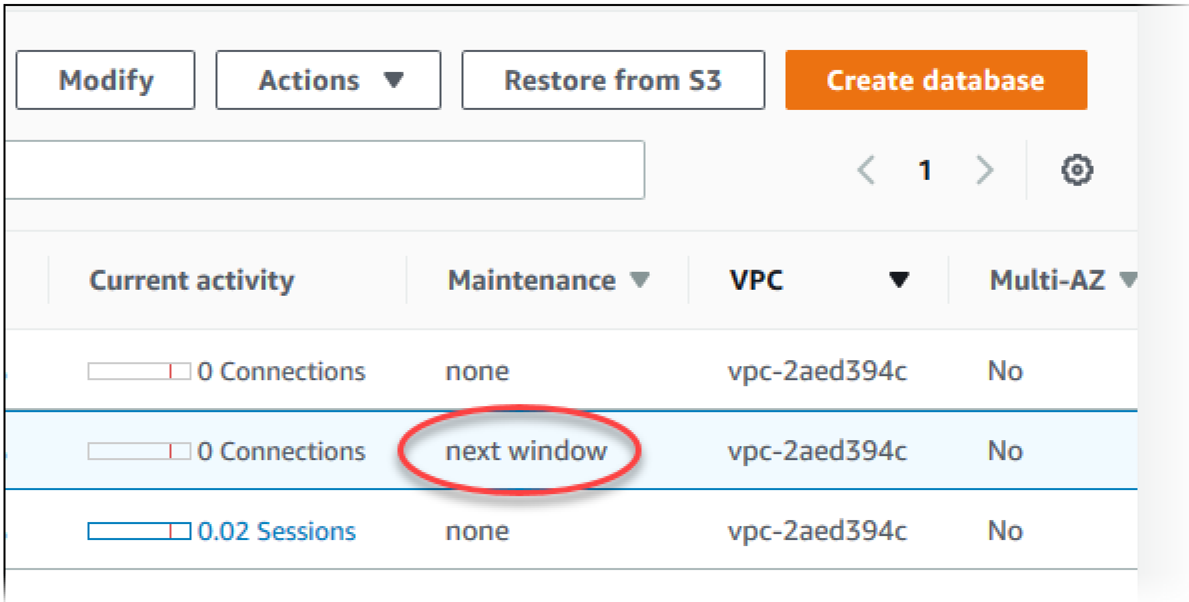
최종 일관성은 유지 관리 업데이트를 관리하는 방식에 영향을 미칠 수 있습니다. 예를 들어 `ApplyPendingMaintenanceActions` 명령을 실행하여 DB 클러스터의 데이터베이스 엔진 버전을 업데이트하면 `DescribePendingMaintenanceActions`가 결국에는 해당 버전을 볼 수 있게 됩니다. 이 시나리오에서 `DescribePendingMaintenanceActions`는 유지 관리 작업이 적용되었음에도 불구하고 적용되지 않았다고 표시할 수 있습니다.

최종 일관성을 관리하려면 다음을 수행할 수 있습니다.

- 수정 명령을 실행하기 전에 DB 클러스터의 상태를 확인합니다. 이전 명령이 시스템에 전파될 시간이 충분하도록 지수 백오프 알고리즘을 사용하여 적절한 `DescribePendingMaintenanceActions` 명령을 실행합니다. 이 작업을 수행하려면 몇 초의 대기 시간부터 시작하여 대기 시간을 5분까지 점진적으로 늘려가며 `DescribePendingMaintenanceActions` 명령을 반복적으로 실행합니다.
- `DescribePendingMaintenanceActions` 명령이 정확한 응답을 반환하더라도 후속 명령 사이에 대기 시간을 추가합니다. 몇 초의 대기 시간으로 시작하는 지수 백오프 알고리즘을 적용하고 대기 시간을 약 5분까지 점진적으로 늘립니다.

보류 중인 유지 관리 업데이트 보기

RDS 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터에 대해 유지 관리 업데이트를 사용할 수 있는지 확인합니다. 업데이트가 있는 경우에는 다음과 같이 Amazon RDS 콘솔에서 DB 클러스터의 유지 관리 열에 사용 가능 여부가 표시됩니다.



DB 클러스터에 대해 유지 관리 업데이트가 제공되지 않는 경우 그에 대한 열 값은 없음입니다.

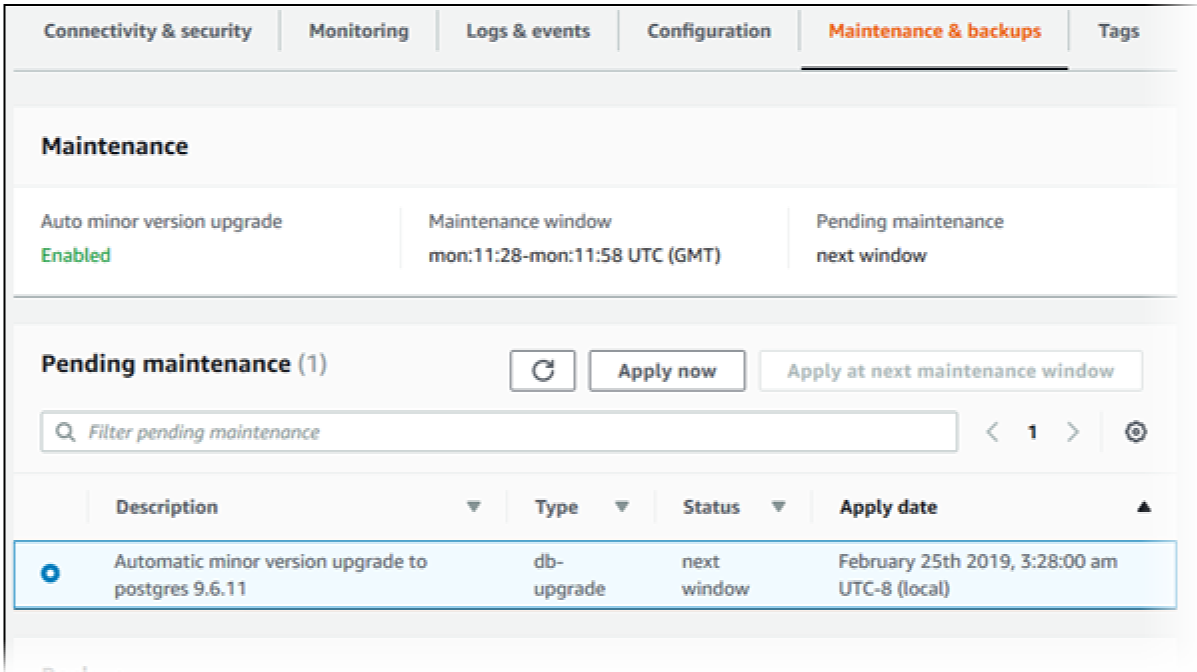
DB 클러스터에 대해 유지 관리 업데이트가 제공되는 경우 다음과 같은 열 값이 가능합니다.

- 필수 – 유지 관리 작업은 리소스에 적용되며 무기한 보류할 수 없습니다.
- 사용 가능 – 유지 관리 작업을 사용할 수 있습니다. 그러나 리소스에 자동으로 적용되지 않고 수동으로 적용할 수 있습니다.
- 다음 기간 – 유지 관리 작업은 다음 유지 관리 기간 중에 리소스에 적용됩니다.
- 진행 중 – 유지 관리 작업이 리소스에 적용되고 있는 중입니다.

업데이트가 있을 경우에는 다음 테이블의 작업 중 하나를 실행할 수 있습니다.

- 유지 관리 값이 다음 기간인 경우 작업에서 업그레이드 보류를 선택하여 유지 관리 항목을 보류하십시오. 유지 관리 작업이 이미 시작된 경우에는 보류할 수 없습니다.
- 유지 관리 항목을 즉시 적용합니다.
- 다음 유지 관리 기간 중 시작할 유지 관리 항목을 예약합니다.
- 작업이 없습니다.

조치를 취하려면 DB 클러스터를 선택하여 세부 정보를 표시한 후 Maintenance & backups(유지 관리 및 백업)을 선택하십시오. 그러면 보류 중인 유지 관리 항목이 표시됩니다.



유지 관리 기간에 따라 대기 중인 작업의 시작 시기가 결정되지만 이러한 작업의 전체 실행 시간이 줄어들지는 않습니다. 유지 관리 기간에 끝나기 전에 반드시 유지 관리 작업이 끝나도록 되어 있는 것은 아니고, 특정 종료 시각을 지나 계속 진행될 수 있습니다. 자세한 내용은 Amazon RDS 유지 관리 기간 섹션을 참조하세요.

Amazon Aurora 엔진으로의 업데이트에 대한 자세한 내용과 엔진을 업그레이드하고 패치를 적용하는 방법을 보려면 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 및 Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하세요.

`describe-pending-maintenance-actions` AWS CLI 명령을 실행하여 DB 클러스터에 유지 관리 업데이트를 사용할 수 있는지 여부를 확인할 수도 있습니다.

유지 관리 업데이트 적용에 대한 내용은 DB 클러스터에 업데이트 적용 섹션을 참조하세요.

DB 클러스터에 업데이트 적용

Amazon RDS를 사용하여 유지 관리 작업을 적용하는 시기를 선택할 수 있습니다. AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Amazon RDS에서 업데이트를 적용하는 시기를 결정할 수 있습니다.

콘솔

DB 클러스터의 업데이트를 관리하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 필수 업데이트가 포함된 DB 클러스터를 선택합니다.
4. 작업에서 다음 중 하나를 선택합니다.
   - 지금 패치
   - 다음 기간에 패치

> **Note**
>
> 다음 기간에 패치를 선택한 후 나중에 업데이트를 연기하려면 업그레이드 연기를 선택합니다. 유지 관리 작업이 이미 시작된 경우에는 보류할 수 없습니다.
> 유지 관리 작업을 취소하려면 DB 인스턴스를 수정하고 마이너 버전 자동 업그레이드를 비활성화합니다.

AWS CLI

대기 중인 업데이트를 DB 클러스터에 적용하려면 apply-pending-maintenance-action AWS CLI 명령을 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
```

```
    --apply-action system-update \
    --opt-in-type immediate
```

Windows의 경우:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> 유지 관리 작업을 연기하려면 `undo-opt-in`에 `--opt-in-type`을 지정합니다. 유지 관리 작업이 이미 시작된 경우 `undo-opt-in`에 `--opt-in-type`을 지정할 수 없습니다.
> 유지 관리 작업을 취소하려면 modify-db-instance AWS CLI 명령을 실행하고 `--no-auto-minor-version-upgrade`을 지정합니다.

하나 이상의 대기 중인 업데이트가 있는 리소스 목록을 반환하려면, describe-pending-maintenance-actions AWS CLI 명령을 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Windows의 경우:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

`describe-pending-maintenance-actions` AWS CLI 명령의 `--filters` 파라미터를 지정하여 DB 클러스터에 대한 리소스 목록을 반환할 수도 있습니다. `--filters` 명령의 형식은 `Name=`*`filter-name`*`,Value=`*`resource-id`*`,...`입니다.

필터의 Name 파라미터에 대해 허용되는 값은 다음과 같습니다.

- `db-instance-id` – DB 인스턴스 식별자 또는 Amazon 리소스 이름(ARN) 목록을 허용합니다. 반환되는 목록에는 이러한 식별자 또는 ARN으로 식별된 DB 인스턴스에 대해 보류 중인 유지 관리 작업만 포함됩니다.
- `db-cluster-id` – Amazon Aurora의 DB 클러스터 식별자 또는 ARN 목록을 허용합니다. 반환되는 목록에는 이러한 식별자 또는 ARN으로 식별된 DB 클러스터에 대해 보류 중인 유지 관리 작업만 포함됩니다.

예를 들어 다음 예에서는 `sample-cluster1` 및 `sample-cluster2` DB 클러스터에 대해 보류 중인 유지 관리 작업을 반환합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Windows의 경우:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

업데이트를 DB 클러스터에 적용하려면 Amazon RDS API `ApplyPendingMaintenanceAction` 작업을 호출합니다.

하나 이상의 대기 중인 업데이트가 있는 리소스 목록을 반환하려면 Amazon RDS API `DescribePendingMaintenanceActions` 작업을 호출합니다.

Amazon RDS 유지 관리 기간

유지 관리 기간은 시스템 변경 내용이 적용되는 주 단위의 기간입니다. 모든 DB 클러스터에는 주간 유지 관리 기간이 있습니다. 유지 관리 기간은 수정 사항 및 소프트웨어 패치 적용 시점을 조절할 수 있는 기회입니다. 유지 관리 기간 조정에 대한 자세한 내용은 기본 DB 클러스터 유지 관리 기간 조정 섹션을 참조하세요.

유지 관리가 적용되는 동안 RDS에서 사용자의 DB 클러스터에 있는 리소스 중 일부를 사용합니다. 이에 따라 성능에 미미한 영향이 있을 수 있습니다. DB 인스턴스의 경우 드물지만, 유지 관리 업데이트를 완료하려면 다중 AZ 장애 조치가 필요한 경우가 있을 수 있습니다.

유지 관리 이벤트가 특정 주에 예정되어 있는 경우 사용자가 지정하는 30분의 유지 관리 기간 중에 해당 이벤트가 시작됩니다. 또한 대부분의 유지 관리 이벤트가 30분의 유지 관리 기간 중에 완료됩니다. 단, 대규모 유지 관리 이벤트는 완료하는 데 30분이 넘게 걸릴 수 있습니다. DB 클러스터가 중지되면 유지 관리 기간이 일시 중지됩니다.

지역별로 8시간 블록 시간 중에서 30분 유지 관리 시간이 임의로 선택됩니다. DB 클러스터 생성 시 유지 관리 기간을 지정하지 않으면 RDS에서 임의로 선택한 요일에 30분 유지 관리 기간을 배정합니다.

다음에서 기본 유지 관리 기간이 할당된 리전별 시간 블록을 확인할 수 있습니다.

리전 이름	리전	시간 블록
미국 동부(오하이오)	us-east-2	03:00~11:00 UTC
미국 동부(버지니아 북부)	us-east-1	03:00~11:00 UTC
미국 서부(캘리포니아 북부 지역)	us-west-1	06:00~14:00 UTC
미국 서부(오리건)	us-west-2	06:00~14:00 UTC
Africa (Cape Town)	af-south-1	03:00~11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00~14:00 UTC
아시아 태평양(하이데라바드)	ap-south-2	06:30~14:30 UTC
아시아 태평양(자카르타)	ap-southeast-3	08:00~16:00 UTC
아시아 태평양(멜버른)	ap-southeast-4	11:00~19:00 UTC
아시아 태평양(뭄바이)	ap-south-1	06:00~14:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	22:00~23:59 UTC

리전 이름	리전	시간 블록
Asia Pacific (Seoul)	ap-northeast-2	13:00~21:00 UTC
아시아 태평양(싱가포르)	ap-southeast-1	14:00~22:00 UTC
아시아 태평양(시드니)	ap-southeast-2	12:00~20:00 UTC
아시아 태평양(도쿄)	ap-northeast-1	13:00~21:00 UTC
Canada (Central)	ca-central-1	03:00~11:00 UTC
캐나다 서부(캘거리)	ca-west-1	18:00~02:00 UTC
중국(베이징)	cn-north-1	06:00~14:00 UTC
China (Ningxia)	cn-northwest-1	06:00~14:00 UTC
Europe (Frankfurt)	eu-central-1	21:00–05:00 UTC
유럽(아일랜드)	eu-west-1	22:00~06:00 UTC
Europe (London)	eu-west-2	22:00~06:00 UTC
유럽(밀라노)	eu-south-1	02:00~10:00 UTC
유럽(파리)	eu-west-3	23:59~07:29 UTC
유럽(스페인)	eu-south-2	02:00~10:00 UTC
Europe (Stockholm)	eu-north-1	23:00~07:00 UTC
유럽(취리히)	eu-central-2	02:00~10:00 UTC
이스라엘(텔아비브)	il-central-1	03:00~11:00 UTC
중동(바레인)	me-south-1	06:00~14:00 UTC
중동(UAE)	me-central-1	05:00~13:00 UTC

리전 이름	리전	시간 블록
남아메리카(상파울루)	sa-east-1	00:00~08:00 UTC
AWS GovCloud(미국 동부)	us-gov-east-1	17:00~01:00 UTC
AWS GovCloud(미국 서부)	us-gov-west-1	06:00~14:00 UTC

기본 DB 클러스터 유지 관리 기간 조정

Aurora DB 클러스터 유지 관리 기간은 사용률이 가장 낮은 시간에 할당되어야 하므로 수시로 수정되어야 할 수 있습니다. 적용 중인 업데이트에 중단이 필요한 경우 이 시간 동안 DB 클러스터를 사용할 수 없습니다. 필수 업데이트를 수행하는 데 필요한 최소 시간 동안 중단됩니다.

콘솔

기본 DB 클러스터 유지 관리 기간을 조정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 유지 관리 기간을 변경하려는 DB 클러스터를 선택합니다.
4. 수정을 선택합니다.
5. 유지 관리 섹션에서 유지 관리 기간을 업데이트합니다.
6. 계속을 선택합니다.

   확인 페이지에서 변경 내용을 검토합니다.
7. 변경 사항을 유지 관리 기간에 즉시 적용하려면 수정 예약(Scheduling of modifications) 섹션에서 즉시(Immediately)를 선택합니다.
8. 클러스터 수정을 선택하여 변경 사항을 저장합니다.

   그렇지 않으면 [Back]을 선택하여 변경 내용을 편집하거나 [Cancel]을 선택하여 변경 내용을 취소합니다.

AWS CLI

기본 DB 클러스터 유지 관리 기간을 조정하려면 AWS CLI `modify-db-cluster` 명령을 다음 파라미터와 함께 사용합니다.

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

다음은 유지 관리 기간을 화요일 4:00–4:30 AM UTC로 설정하는 코드 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Windows의 경우:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

기본 DB 클러스터 유지 관리 기간을 조정하려면 Amazon RDS `ModifyDBCluster` API 작업을 다음 파라미터와 함께 사용합니다.

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Aurora DB 클러스터 마이너 버전 자동 업그레이드

마이너 버전 자동 업그레이드 설정은 Aurora가 DB 클러스터에 업그레이드를 자동으로 적용할지 여부를 지정합니다. 이러한 업그레이드에는 추가 기능과 버그 수정 사항이 들어 있는 패치가 포함된 새 마이너 버전이 포함됩니다.

이 설정은 기본적으로 켜져 있습니다. 각 새 DB 클러스터에 대해 이 설정에 적합한 값을 선택합니다. 중요도, 예상 수명 및 각 업그레이드 후에 수행하는 확인 테스트 양에 따라 값을 선택합니다.

자동 마이너 버전 업그레이드 설정을 켜거나 끄는 방법에 대한 지침은 다음을 참조하세요.

- Aurora DB 클러스터 마이너 버전 자동 업그레이드 사용 설정
- Aurora DB 클러스터의 개별 DB 인스턴스에 마이너 버전 자동 업그레이드 사용 설정

Important

신규 및 기존 DB 클러스터의 경우 이 설정을 클러스터의 DB 인스턴스에 개별적으로 적용하는 것이 아니라 DB 클러스터에 적용하는 것이 좋습니다. 클러스터에서 이 설정이 꺼져 있는 DB 인스턴스가 있으면 DB 클러스터가 자동으로 업그레이드되지 않습니다.

다음 테이블은 클러스터 및 인스턴스 수준에서 적용한 자동 마이너 버전 업그레이드 설정이 작동하는 방식을 보여줍니다.

작업	클러스터 설정	인스턴스 설정	클러스터가 자동으로 업그레이드되었나요?
DB 클러스터에서는 참으로 설정했습니다.	True	모든 신규 및 기존 인스턴스에 대해 참	예
DB 클러스터에서는 거짓으로 설정했습니다.	False	모든 신규 및 기존 인스턴스에 대해 거짓	아니요
이전에 DB 클러스터에서 참으로 설정되었습니다. 하나 이상의 DB 인스턴스에서 거짓으로 설정했습니다.	거짓으로 변경	하나 이상의 인스턴스에 대해 거짓	아니요
이전에 DB 클러스터에서 거짓으로 설정되었습니다. 모든 인스턴스가 아닌 하나 이상의 DB 인스	False	모든 인스턴스가 아닌 하나 이상의 인스턴스에 대해 참	아니요

작업	클러스터 설정	인스턴스 설정	클러스터가 자동으로 업그레이드되었나요?
턴스에서 참으로 설정했습니다.			
이전에 DB 클러스터에서 거짓으로 설정되었습니다. 모든 DB 인스턴스에서 참으로 설정했습니다.	참으로 변경	모든 인스턴스에 대해 참	예

자동 마이너 버전 업그레이드는 카테고리가 `maintenance`이며 ID가 `RDS-EVENT-0156`인 Amazon RDS DB 클러스터 이벤트를 통해 미리 전달됩니다. 자세한 내용은 Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지 단원을 참조하십시오.

자동 업그레이드는 유지 관리 기간 중에 발생합니다. DB 클러스터에서 개별 DB 인스턴스의 유지 관리 기간이 클러스터 유지 관리 기간과 다른 경우 클러스터 유지 관리 기간이 우선합니다.

Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

Aurora MySQL의 마이너 버전 자동 업그레이드(Auto minor version upgrade) 설정에 대한 자세한 내용은 마이너 Aurora MySQL 버전 간 자동 업그레이드 활성화 단원을 참조하세요. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하세요.

Aurora DB 클러스터 마이너 버전 자동 업그레이드 사용 설정

콘솔, CLI, API를 사용하여 DB 클러스터 수정의 일반 절차를 따르세요.

콘솔

DB 클러스터 수정 페이지의 유지 관리 섹션에서 마이너 버전 자동 업그레이드 사용 확인란을 선택합니다.

AWS CLI

modify-db-cluster AWS CLI 명령을 호출합니다. `--db-cluster-identifier` 옵션에 DB 클러스터 이름을 지정하고 `--auto-minor-version-upgrade` 옵션에 `true`를 지정합니다. 필요에 따

라 DB 클러스터에 대해 이 설정을 즉시 사용 설정하는 `--apply-immediately` 옵션을 지정합니다.

RDS API

ModifyDBCluster API 작업을 호출하고, `DBClusterIdentifier` 파라미터에 DB 클러스터 이름을 지정하고 `AutoMinorVersionUpgrade` 파라미터에 `true`를 지정합니다. 필요에 따라 `ApplyImmediately` 파라미터를 `true`로 설정하여 DB 클러스터에 대해 이 설정을 즉시 사용 설정합니다.

Aurora DB 클러스터의 개별 DB 인스턴스에 마이너 버전 자동 업그레이드 사용 설정

DB 클러스터에서 DB 인스턴스 수정의 일반 절차를 따르세요.

콘솔

DB 인스턴스 수정 페이지의 유지 관리 섹션에서 마이너 버전 자동 업그레이드 사용 확인란을 선택합니다.

AWS CLI

modify-db-instance AWS CLI 명령을 호출합니다. `--db-instance-identifier` 옵션에 해당 DB 인스턴스 이름을 지정하고 `true` 옵션에 `--auto-minor-version-upgrade`를 지정합니다. 필요에 따라 DB 인스턴스에 대해 이 설정을 즉시 활성화하는 `--apply-immediately` 옵션을 지정합니다. 클러스터의 각 DB 인스턴스에 대해 별도의 `modify-db-instance` 명령을 실행합니다.

RDS API

ModifyDBInstance API 작업을 호출하고, `DBInstanceIdentifier` 파라미터에 DB 클러스터 이름을 지정하고 `AutoMinorVersionUpgrade` 파라미터에 `true`를 지정합니다. 필요에 따라 `ApplyImmediately` 파라미터를 `true`로 설정하여 DB 인스턴스에 대해 이 설정을 즉시 활성화합니다. 클러스터의 각 DB 인스턴스에 대해 별도의 `ModifyDBInstance` 작업을 호출합니다.

다음과 같은 CLI 명령을 사용하여 Aurora MySQL 클러스터의 모든 DB 인스턴스에 대한 `AutoMinorVersionUpgrade` 설정의 상태를 확인할 수 있습니다.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

이 명령을 실행하면 다음과 비슷한 출력이 생성됩니다.

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

이 예시에서는 클러스터의 DB 인스턴스 중 하나에 대해 마이너 버전 자동 업그레이드 사용이 꺼져 있으므로, DB 클러스터 `my-db-cluster-57`에 대한 마이너 버전 자동 업그레이드 사용도 꺼져 있습니다.

Aurora MySQL 유지 관리 업데이트 빈도 선택

각 DB 클러스터에 대해 Aurora MySQL 업그레이드가 자주 발생하는지 또는 드물게 발생하는지 제어할 수 있습니다. 적합한 방법은 Aurora MySQL 사용량 및 Aurora에서 실행되는 애플리케이션의 우선순위에 따라 다릅니다. 업그레이드를 자주 수행하지 않아도 되는 Aurora MySQL 장기 안정성 (LTS) 릴리스에 대한 자세한 내용은 [Aurora MySQL LTS(장기 지원) 릴리스](#) 섹션을 참조하세요.

다음 조건이 일부 또는 모두 적용되는 경우 Aurora MySQL 클러스터를 드물게 업그레이드하도록 선택할 수 있습니다.

- Aurora MySQL 데이터베이스 엔진을 업데이트할 때마다 애플리케이션의 테스트 주기가 오래 걸리는 경우
- 동일한 Aurora MySQL 버전에서 많은 DB 클러스터 또는 많은 애플리케이션이 모두 실행 중일 때, 모든 DB 클러스터와 관련 애플리케이션을 동시에 업그레이드하고 싶은 경우
- Aurora MySQL과 RDS for MySQL을 모두 사용하는 경우 Aurora MySQL 클러스터 및 RDS MySQL DB 인스턴스를 동일한 수준의 MySQL과 호환되도록 유지하려는 경우
- Aurora MySQL 애플리케이션이 프로덕션 환경이거나 업무상 중요한 애플리케이션일 때, 중요 패치의 경우 드문 경우를 제외하고는 업그레이드에 대한 가동 중지를 감당할 수 없는 경우

- 후속 Aurora MySQL 버전에서 해결되는 성능 문제나 기능 차이로 인해 Aurora MySQL 애플리케이션이 제한되지 않는 경우

위의 요소가 현재 상황에 적용되는 경우 Aurora MySQL DB 클러스터에 강제 적용되는 업그레이드 수를 제한할 수 있습니다. 해당 DB 클러스터를 생성하거나 업그레이드할 때 "LTS(장기 지원)" 버전으로 알려진 특정 Aurora MySQL 버전을 선택하면 됩니다. 이렇게 하면 해당 DB 클러스터의 업그레이드 주기, 테스트 주기 및 업그레이드 관련 중단 수가 최소화됩니다.

다음 조건이 일부 또는 모두 적용되는 경우 Aurora MySQL 클러스터를 자주 업그레이드하도록 선택할 수 있습니다.

- 애플리케이션의 테스트 주기가 간단한 경우
- 애플리케이션이 아직 개발 단계에 있는 경우
- 데이터베이스 환경이 다양한 Aurora MySQL 버전, 또는 Aurora MySQL 및 RDS for MySQL 버전을 사용하는 경우 각 Aurora MySQL 클러스터에는 자체 업그레이드 주기가 있습니다.
- Aurora MySQL 사용량을 늘리기 전에 특정 성능 또는 기능 개선을 대기하고 있는 경우

위의 요소가 현재 상황에 적용되는 경우 Aurora가 중요한 업그레이드를 더 자주 적용하도록 활성화할 수 있습니다. 그러기 위해서는 Aurora MySQL DB 클러스터를 LTS 버전보다 최신인 Aurora MySQL 버전으로 업그레이드해야 합니다. 이렇게 하면 최신 성능 향상, 버그 수정 및 기능을 보다 신속하게 사용할 수 있습니다.

Amazon Aurora DB 클러스터 또는 Amazon Aurora DB 인스턴스 재부팅

일반적으로 유지 관리를 위해 DB 클러스터 또는 클러스터 내의 일부 인스턴스를 재부팅해야 할 수 있습니다. 예를 들어 파라미터 그룹 내의 파라미터를 수정하거나 다른 파라미터 그룹을 클러스터에 연결한다고 가정해 보겠습니다. 이러한 경우 변경 사항을 적용하려면 클러스터를 재부팅해야 합니다. 마찬가지로 클러스터 내에서 하나 이상의 리더 DB 인스턴스를 재부팅할 수 있습니다. 개별 인스턴스에 대한 재부팅 작업을 준비하면 전체 클러스터의 다운타임을 최소화할 수 있습니다.

클러스터의 각 DB 인스턴스를 재부팅하는 데 필요한 시간은 재부팅 시 데이터베이스 활동에 따라 다릅니다. 또한 특정 DB 엔진의 복구 프로세스에 따라서도 다릅니다. 가능하다면 재부팅 프로세스를 시작하기 전에 특정 인스턴스의 데이터베이스 활동을 줄이세요. 이렇게 하면 데이터베이스를 다시 시작하는 데 필요한 시간을 줄일 수 있습니다.

클러스터의 각 DB 인스턴스는 사용 가능한 상태인 경우에만 재부팅할 수 있습니다. DB 인스턴스를 사용할 수 없는 이유로는 여러 가지가 있습니다. 예를 들어 클러스터가 중지 상태에 있는 경우, 인스턴스에 수정을 적용 중인 경우 및 유지 관리 기간 작업(예: 버전 업그레이드)이 여기에 포함됩니다.

DB 인스턴스를 재부팅하면 데이터베이스 엔진 프로세스가 다시 시작됩니다. DB 인스턴스를 재부팅하면 DB 인스턴스 상태가 rebooting으로 설정되면서 잠시 중단됩니다.

> **Note**
>
> DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않은 경우 AWS Management Console에 DB 파라미터 그룹이 재시작 보류중 상태로 표시됩니다. 재시작 보류중 파라미터 그룹 상태로 인해 다음번 유지 관리 기간 중에 자동 재부팅이 되지는 않습니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.

주제

Aurora 클러스터 내의 DB 인스턴스 재부팅

이 절차는 Aurora에서 재부팅을 수행할 때 가장 중요한 작업입니다. 대부분의 유지 관리 절차에는 하나 이상의 Aurora DB 인스턴스를 특정 순서로 재부팅하는 작업이 포함됩니다.

콘솔

DB 인스턴스를 재부팅하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 재부팅하려는 DB 인스턴스를 선택합니다.
3. 작업에서 재부팅을 선택합니다.

   [Reboot DB Instance] 페이지가 나타납니다.
4. DB 인스턴스를 재부팅하려면 [Reboot]를 선택합니다.

   또는 [취소(Cancel)]를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스를 재부팅하려면 `reboot-db-instance` 명령을 호출하십시오.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

Windows의 경우:

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

Amazon RDS API를 사용하여 DB 인스턴스를 재부팅하려면 `RebootDBInstance` 작업을 호출하세요.

읽기 가용성을 포함하여 Aurora 클러스터 재부팅

읽기 가용성 기능을 사용하면 기본 또는 보조 DB 클러스터의 리더 인스턴스를 재부팅하지 않고도 Aurora 클러스터의 라이터 인스턴스를 재부팅할 수 있습니다. 이렇게 하면 라이터 인스턴스를 재부팅하는 동안 읽기 작업에 대한 클러스터의 고가용성을 유지하는 데 도움이 됩니다. 나중에 편리한 일정에 따라 리더 인스턴스를 재부팅할 수 있습니다. 예를 들어 프로덕션 클러스터에서는 프라이머리 인스턴스의 재부팅이 완료된 후에만 리더 인스턴스를 한 번에 하나씩 재부팅할 수 있습니다. 재부팅하는 각 DB 인스턴스에 대해 Aurora 클러스터 내의 DB 인스턴스 재부팅의 절차를 따릅니다.

기본 DB 클러스터의 읽기 가용성 기능은 Aurora MySQL 버전 2.10 이상에서 사용할 수 있습니다. 보조 DB 클러스터의 읽기 가용성은 Aurora MySQL 버전 3.06 이상에서 사용할 수 있습니다.

이 함수는 다음 Aurora PostgreSQL 버전에서 기본적으로 사용할 수 있습니다.

- 15.2 이상의 15 버전
- 14.7 이상의 14 버전
- 13.10 이상의 13 버전
- 12.14 이상의 12 버전

Aurora PostgreSQL의 읽기 가용성 기능에 대한 자세한 내용은 Aurora 복제본의 읽기 가용성 향상 섹션을 참조하세요.

이 기능이 출시되기 전에는 기본 인스턴스를 재부팅하면 각 리더 인스턴스가 동시에 재부팅되었습니다. Aurora 클러스터에서 이전 버전을 실행 중인 경우 읽기 가용성을 포함하지 않고 Aurora 클러스터 재부팅의 재부팅 절차를 대신 사용합니다.

> **Note**
>
> Aurora MySQL 버전 3.06 미만의 Aurora 글로벌 데이터베이스의 경우 읽기 가용성이 포함된 Aurora DB 클러스터의 재부팅 동작에 대한 변경 사항이 다릅니다. Aurora 글로벌 데이터베이스의 프라이머리 클러스터에 대한 라이터 인스턴스를 재부팅하는 경우 프라이머리 클러스터의 리더 프로그램 인스턴스는 사용할 수 있는 상태로 유지됩니다. 그러나 세컨더리 클러스터의 DB 인스턴스는 동시에 재부팅됩니다.
> 향상된 읽기 가용성 기능의 제한된 버전은 Aurora PostgreSQL 버전 12.16, 13.12, 14.9, 15.4 이상용 Aurora 글로벌 데이터베이스에서 지원됩니다.

클러스터 파라미터 그룹을 변경한 후에는 클러스터를 자주 재부팅하게 됩니다. 파라미터를 변경하려면 Amazon Aurora의 파라미터 그룹의 절차를 따릅니다. 클러스터 파라미터에 변경 사항을 적용하기 위해 Aurora 클러스터에서 라이터 DB 인스턴스를 재부팅한다고 가정해 보겠습니다. 리더 DB 인스턴스의 일부 또는 전부에는 이전 파라미터 설정이 계속해서 사용될 수 있습니다. 그러나 다른 파라미터 설정이 클러스터의 데이터 무결성에 영향을 미치지는 않습니다. 데이터 파일 구성에 영향을 주는 클러스터 파라미터는 라이터 DB 인스턴스에만 사용됩니다.

예를 들어, Aurora MySQL 인스턴스에서 리더 인스턴스 전에 라이터 인스턴스의 `binlog_format` 및 `innodb_purge_threads`와 같은 클러스터 파라미터를 업데이트할 수 있습니다. 라이터 인스턴스만 바이너리 로그를 작성하고 실행 취소 레코드를 제거합니다. 쿼리에서 SQL 문 또는 쿼리 출력을 해석하는 방식을 변경하는 파라미터의 경우 리더 인스턴스를 즉시 재부팅해야 할 수 있습니다. 이렇게 해야 쿼리 중에 예기치 않은 애플리케이션 동작을 방지할 수 있습니다. 예를 들어 `lower_case_table_names` 파라미터를 변경하고 라이터 인스턴스를 재부팅한다고 가정합시다. 이 경우 리더 인스턴스가 모두 재부팅될 때까지 새로 생성한 테이블에 액세스하지 못할 수 있습니다.

모든 Aurora MySQL 클러스터 파라미터 목록은 클러스터 수준 파라미터 섹션을 참조하세요.

모든 Aurora PostgreSQL 클러스터 파라미터 목록은 Aurora PostgreSQL 클러스터 수준 파라미터 섹션을 참조하세요.

> **Tip**
>
> 클러스터에서 처리량이 높은 워크로드를 처리 중인 경우 Aurora MySQL은 라이터 인스턴스와 함께 일부 리더 인스턴스를 재부팅할 수 있습니다.
> 재부팅 횟수의 감소는 장애 조치 작업 중에도 적용됩니다. Aurora MySQL은 장애 조치 중에 라이터 DB 인스턴스와 장애 조치 대상만 재시작합니다. 클러스터의 다른 리더 DB 인스턴스는 리더 엔드포인트에 대한 연결을 통해 쿼리를 처리하는 데 계속해서 사용할 수 있습니다. 따라서 클러스터에 리더 DB 인스턴스를 2개 이상 보유하여 장애 조치 중에 가용성을 개선할 수 있습니다.

읽기 가용성을 포함하지 않고 Aurora 클러스터 재부팅

읽기 가용성 기능을 사용하지 않으면 클러스터의 라이터 DB 인스턴스를 재부팅하여 전체 Aurora DB 클러스터를 재부팅할 수 있습니다. 이 작업을 수행하려면 의 절차를 수행합니다Aurora 클러스터 내의 DB 인스턴스 재부팅

라이터 DB 인스턴스를 재부팅하면 클러스터의 각 리더 DB 인스턴스에 대해서도 재부팅이 시작됩니다. 이 방식으로 클러스터 전체 파라미터 변경 사항이 모든 DB 인스턴스에 동시에 적용됩니다. 그러나

모든 DB 인스턴스를 재부팅하면 클러스터가 잠시 중단됩니다. 라이터 DB 인스턴스의 재부팅이 완료되고 사용할 수 있게 될 때까지 리더 DB 인스턴스를 사용할 수 없게 됩니다.

이 재부팅 동작은 Aurora MySQL 버전 2.09 이하에서 생성된 모든 DB 클러스터에 적용됩니다.

Aurora PostgreSQL의 경우 이 동작은 다음 버전에 적용됩니다.

- 14.6 이하 14 버전
- 13.9 이하 13 버전
- 12.13 이하 12 버전
- 모든 PostgreSQL 11 버전

RDS 콘솔에서 라이터 DB 인스턴스의 값인 [라이터(Writer)]는 [데이터베이스(Databases)] 페이지의 [역할(Role)] 열에 표시됩니다. RDS CLI에서 `describe-db-clusters` 명령 출력에는 `DBClusterMembers` 섹션이 포함됩니다. 라이터 DB 인스턴스를 나타내는 `DBClusterMembers` 요소는 `true` 필드에 `IsClusterWriter` 값이 표시됩니다.

> **Important**
>
> 읽기 가용성 기능을 사용하면 Aurora MySQL 및 Aurora PostgreSQL에서는 재부팅 동작이 다릅니다. 일반적으로 라이터 인스턴스를 재부팅하는 동안 리더 DB 인스턴스를 사용할 수 있습니다. 그런 다음 편리한 시간에 리더 인스턴스를 재부팅할 수 있습니다. 일부 리더 인스턴스를 항상 사용할 수 있도록 하려면 시간 간격을 두고 리더 인스턴스를 재부팅하면 됩니다. 자세한 내용은 읽기 가용성을 포함하여 Aurora 클러스터 재부팅 단원을 참조하십시오.

Aurora 클러스터 및 인스턴스의 가동 시간 확인

Aurora 클러스터의 각 DB 인스턴스에 대한 마지막 재부팅 이후 경과 시간을 확인하고 모니터링할 수 있습니다. Amazon CloudWatch 지표 `EngineUptime`은 DB 인스턴스가 마지막으로 시작된 이후의 시간(초)을 보고합니다. 특정 시점에 이 지표를 검토하여 DB 인스턴스의 가동 시간을 확인할 수 있습니다. 이 지표를 시간대별로 모니터링하여 인스턴스가 재부팅되는 시기를 감지할 수도 있습니다.

클러스터 수준에서 `EngineUptime` 지표를 검사할 수도 있습니다. `Minimum` 및 `Maximum` 차원은 클러스터의 모든 DB 인스턴스에 대한 최소 및 최대 가동 시간 값을 보고합니다. 클러스터의 리더 인스턴스가 재부팅되거나 다른 이유로 다시 시작된 최근 시간을 확인하려면 `Minimum` 차원을 사용하여 클러스터 수준 지표를 모니터링합니다. 클러스터에서 재부팅 없이 가장 오래 지속된 인스턴스를 확인하려면

Maximum 차원을 사용하여 클러스터 수준 지표를 모니터링합니다. 예를 들어 구성 변경 후 클러스터의 모든 DB 인스턴스가 재부팅되었는지 확인해야 할 수 있습니다.

> **Tip**
>
> 장기 모니터링의 경우 클러스터 수준 대신 개별 인스턴스에 대한 EngineUptime 지표를 모니터링하는 것이 좋습니다. 클러스터에 새 DB 인스턴스가 추가되면 클러스터 수준 EngineUptime 지표가 0으로 설정됩니다. 이러한 클러스터 변경은 Auto Scaling에서 수행되는 것과 같은 유지 관리 및 확장 작업의 일부로 발생할 수 있습니다.

다음 CLI 예제는 클러스터의 라이터 및 리더 인스턴스에 대한 EngineUptime 지표를 검사하는 방법을 보여줍니다. 이 예제에는 이름이 tpch100g인 클러스터가 사용됩니다. 이 클러스터에는 라이터 DB 인스턴스 instance-1234가 있습니다. 또한 2개의 리더 DB 인스턴스 instance-7448과 instance-6305가 있습니다.

먼저 reboot-db-instance 명령은 리더 인스턴스 중 하나를 재부팅합니다. wait 명령은 인스턴스 재부팅이 완료될 때까지 대기합니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch get-metric-statistics 명령은 지난 5분간의 EngineUptime 지표를 1분 간격으로 검사합니다. instance-6305 인스턴스의 가동 시간이 0으로 재설정되고 다시 계산되기 시작합니다. 이 Linux용 AWS CLI 예제에서는 $() 변수 대체를 사용하여 CLI 명령에 해당하는 타임스탬프를 삽입합니다. 또한 Linux sort 명령을 사용하여 지표가 수집된 시간별로 출력 순서를 지정합니다. 이 타임스탬프 값은 출력의 각 줄에서 세 번째 필드입니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
```

```
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

클러스터의 인스턴스 중 하나가 재부팅되었으므로 클러스터의 최소 가동 시간이 0으로 재설정됩니다. 클러스터의 DB 인스턴스 중 하나 이상이 사용 가능한 상태로 남아 있기 때문에 클러스터의 최대 가동 시간은 재설정되지 않습니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

그런 다음 다른 `reboot-db-instance` 명령에 의해 클러스터의 라이터 인스턴스가 재부팅됩니다. 다른 `wait` 명령은 라이터 인스턴스의 재부팅이 완료될 때까지 일시 중지됩니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

이제 라이터 인스턴스의 `EngineUptime` 지표에서 인스턴스 `instance-1234`가 최근에 재부팅되었음을 알 수 있습니다. 리더 인스턴스 `instance-6305`도 라이터 인스턴스와 함께 자동으로 재부팅되었습니다. 이 클러스터는 Aurora MySQL 2.09를 실행 중이므로 라이터 인스턴스가 재부팅될 때 리더 인스턴스의 실행이 유지되지 않습니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Aurora 재부팅 작업의 예

다음 Aurora MySQL 예제에서는 Aurora DB 클러스터의 리더 및 라이터 DB 인스턴스에 대한 재부팅 작업의 여러 조합을 보여줍니다. 재부팅할 때마다 SQL 쿼리는 클러스터의 인스턴스에 대한 가동 시간을 보여줍니다.

주제

Aurora 클러스터의 라이터 및 리더 인스턴스 찾기

DB 인스턴스가 여러 개인 Aurora MySQL 클러스터에서는 어느 인스턴스가 라이터이고, 어느 인스턴스가 리더인지 파악하는 것이 중요합니다. 또한 라이터 인스턴스와 리더 인스턴스는 장애 조치 작업이 수행될 때 역할을 전환할 수 있습니다. 따라서 라이터 또는 리더 인스턴스가 필요한 작업을 수행하기 전에 다음과 같은 검사를 수행하는 것이 가장 좋습니다. 이 경우 `False`의 `IsClusterWriter` 값은 리더 인스턴스 `instance-6305` 및 `instance-7448`을 식별합니다. `True` 값은 라이터 인스턴스 `instance-1234`를 식별합니다.

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

재부팅 예제를 시작하기 전에 라이터 인스턴스의 가동 시간은 약 1주일입니다. 이 예제의 SQL 쿼리는 가동 시간을 확인하는 MySQL 관련 방법을 보여줍니다. 데이터베이스 애플리케이션에서 이 기술을 사용할 수 있습니다. AWS CLI를 사용하고 두 Aurora 엔진에서 모두 작동하는 다른 기술은 [Aurora 클러스터 및 인스턴스의 가동 시간 확인](#) 섹션을 참조하세요.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

단일 리더 인스턴스 재부팅

이 예제에서는 리더 DB 인스턴스 중 하나를 재부팅합니다. 아마도 이 인스턴스는 거대한 쿼리 또는 많은 동시 연결로 인해 오버로드되었을 수 있습니다. 또는 네트워크 문제로 인해 라이터 인스턴스보다 뒤

쳐졌을 수 있습니다. 재부팅 작업이 시작된 후 이 예제에서는 wait 명령을 사용하여 인스턴스를 사용할 수 있게 될 때까지 일시 중지합니다. 이 시점까지 인스턴스의 가동 시간은 몇 분입니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

리더 인스턴스를 재부팅해도 라이터 인스턴스의 가동 시간은 영향을 받지 않았습니다. 여전히 가동 시간은 약 1주일입니다.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

라이터 인스턴스 재부팅

이 예제에서는 라이터 인스턴스를 재부팅합니다. 이 클러스터는 Aurora MySQL 버전 2.09를 실행 중입니다. Aurora MySQL 버전이 2.10보다 낮기 때문에 라이터 인스턴스를 재부팅하면 클러스터의 모든 리더 인스턴스도 재부팅됩니다.

`wait` 명령에 의해 재부팅이 완료될 때까지 일시 중지됩니다. 이제 해당 인스턴스의 가동 시간이 0으로 재설정됩니다. 라이터 및 리더 DB 인스턴스에 대한 재부팅 작업에 소요되는 시간은 크게 다를 수 있습니다. 라이터 및 리더 DB 인스턴스는 역할에 따라 서로 다른 종류의 정리 작업을 수행합니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

라이터 DB 인스턴스를 재부팅한 후 두 리더 DB 인스턴스의 가동 시간이 모두 재설정됩니다. 라이터 인스턴스가 재부팅되어 리더 인스턴스도 재부팅되었습니다. 이 동작은 Aurora PostgreSQL 클러스터 및 버전 2.10 이전의 Aurora MySQL 클러스터에 적용됩니다.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
```

```
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

라이터와 리더를 독립적으로 재부팅

다음 예제에서는 Aurora MySQL 버전 2.10을 실행하는 클러스터를 보여줍니다. 이 Aurora MySQL 버전 이상에서는 모든 리더 인스턴스에 대한 재부팅을 야기하지 않고 라이터 인스턴스를 재부팅할 수 있습니다. 이렇게 하면 라이터 인스턴스를 재부팅할 때 쿼리 집약적인 애플리케이션이 중단되지 않습니다. 나중에 리더 인스턴스를 재부팅할 수 있습니다. 쿼리 트래픽이 낮은 시간에 이러한 재부팅을 수행할 수 있습니다. 리더 인스턴스를 한 번에 하나씩 재부팅할 수도 있습니다. 이렇게 하면 애플리케이션의 쿼리 트래픽에 항상 하나 이상의 리더 인스턴스를 사용할 수 있습니다.

다음 예제에서는 Aurora MySQL 버전 `cluster-2393`을 실행하는 `5.7.mysql_aurora.2.10.0`이라는 이름의 클러스터를 사용합니다. 이 클러스터에는 이름이 `instance-9404`인 라이터 인스턴스 1개와 이름이 `instance-6772`, `instance-2470` 및 `instance-5138`인 리더 인스턴스 3개가 있습니다.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138       False
Instance:       instance-2470       False
Instance:       instance-6772       False
Instance:       instance-9404       True
```

`uptime` 명령을 통해 각 데이터베이스 인스턴스의 `mysql` 값을 확인하면 각 인스턴스의 가동 시간이 거의 동일하다는 것을 알 수 있습니다. 예를 들어 `instance-5138`의 가동 시간은 다음과 같습니다.

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

CloudWatch를 사용하면 실제로 인스턴스에 로그인하지 않고도 해당 가동 시간 정보를 얻을 수 있습니다. 이 방식으로 관리자는 데이터베이스를 모니터링할 수 있지만 테이블 데이터를 보거나 변경할 수는 없습니다. 이 예에서는 5분에 걸친 기간을 지정하고 1분 간격으로 가동 시간 값을 확인합니다. 가동 시간 값이 증가하면 해당 기간 동안 인스턴스가 다시 시작되지 않았음을 나타냅니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

이제 리더 인스턴스 중 하나인 `instance-5138`을 재부팅합니다. 재부팅 후 인스턴스를 다시 사용할 수 있을 때까지 기다립니다. 이제 5분 동안 가동 시간을 모니터링하면 해당 시간 동안 가동 시간이 0으로 재설정된 것을 알 수 있습니다. 가장 최근의 가동 시간 값은 재부팅이 완료되고 5초 후에 측정되었습니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

다음으로 라이터 인스턴스 `instance-9404`에 대해 재부팅을 수행합니다. 라이터 인스턴스와 리더 인스턴스 중 하나에 대한 가동 시간 값을 비교합니다. 이렇게 하면 라이터를 재부팅해도 리더가 재부팅되지 않았다는 것을 알 수 있습니다. Aurora MySQL 2.10 이전 버전에서는 모든 리더의 가동 시간 값이 라이터와 동시에 재설정됩니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds
```

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

모든 리더 인스턴스에서 라이터 인스턴스와 동일한 구성 파라미터가 변경되도록 하려면 라이터 다음에 모든 리더 인스턴스를 재부팅합니다. 이 예제에서는 모든 리더를 재부팅한 다음 사용할 수 있을 때까지 기다린 후 계속합니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

이제 라이터 DB 인스턴스의 가동 시간이 가장 높다는 것을 알 수 있습니다. 이 인스턴스의 가동 시간 값은 모니터링 기간 동안 꾸준히 증가했습니다. 리더 DB 인스턴스는 모두 리더 이후에 재부팅되었습니

다. 모니터링 기간 내에서 각 리더가 재부팅되고 가동 시간이 0으로 재설정된 시점을 확인할 수 있습니다.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

Aurora MySQL 버전 2.10 클러스터에 클러스터 파라미터 변경 사항 적용

다음 예제는 Aurora MySQL 2.10 클러스터의 모든 DB 인스턴스에 파라미터 변경 사항을 적용하는 방법을 보여줍니다. 이 Aurora MySQL 버전에서는 라이터 인스턴스와 모든 리더 인스턴스를 독립적으로 재부팅합니다.

이 예제에서는 MySQL 구성 파라미터 `lower_case_table_names`를 사용하여 설명합니다. 이 파라미터 설정이 라이터와 리더 DB 인스턴스 간에 다른 경우 쿼리에서 대문자 또는 대/소문자 혼합 이름으로 선언된 테이블에 액세스하지 못할 수 있습니다. 또는 2개의 테이블 이름이 대문자와 소문자의 측면에서만 다른 경우 쿼리가 잘못된 테이블에 액세스할 수 있습니다.

이 예제에서는 각 인스턴스의 `IsClusterWriter` 속성을 검사하여 클러스터의 라이터 및 리더 인스턴스를 확인하는 방법을 보여줍니다. 클러스트 이름은 `cluster-2393`입니다. 이 클러스터에는 `instance-9404`라는 이름의 라이터 인스턴스가 있습니다. 클러스터의 리더 인스턴스 이름은 `instance-5138` 및 `instance-2470`입니다.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

`lower_case_table_names` 파라미터 변경의 효과를 보여주기 위해 2개의 DB 클러스터 파라미터 그룹을 설정했습니다. `lower-case-table-names-0` 파라미터 그룹에서는 이 파라미터가 0으로 설정되어 있습니다. `lower-case-table-names-1` 파라미터 그룹에서는 이 파라미터 그룹이 1로 설정되어 있습니다.

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}
```

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

`lower_case_table_names`의 기본값은 0입니다. 이 파라미터 설정에서 테이블 `foo`는 테이블 `FOO`와 구별됩니다. 이 예제에서는 파라미터가 여전히 기본 설정에 있는지 확인합니다. 그런 다음 이름에서 대문자와 소문자만 다른 테이블 3개를 생성합니다.

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+
```

```
mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

다음으로 DB 파라미터 그룹을 클러스터에 연결하여 `lower_case_table_names` 파라미터를 1로 설정합니다. 이 변경 사항은 각 DB 인스턴스를 재부팅한 후에만 적용됩니다.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

첫 번째 재부팅은 라이터 DB 인스턴스에 대해 수행합니다. 그런 다음 인스턴스를 다시 사용할 수 있을 때까지 기다립니다. 이 시점에서 라이터 엔드포인트에 연결하고 라이터 인스턴스에 변경된 파라미터 값이 있는지 확인합니다. `SHOW TABLES` 명령을 사용하여 데이터베이스에 3개의 서로 다른 테이블이 포함되어 있음을 확인합니다. 그러나 `foo`, `Foo` 또는 `FOO` 테이블을 참조하는 쿼리는 이름이 모두 소문자인 `foo` 테이블에 액세스합니다.

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

이제 클러스터 엔드포인트를 사용하는 쿼리를 통해 파라미터 변경의 영향을 볼 수 있습니다. 쿼리의 테이블 이름이 대문자인지, 소문자인지, 대/소문자 혼합인지 여부에 관계없이 SQL 문은 이름이 모두 소문자인 테이블에 액세스합니다.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
```

```
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

다음 예제는 이전 쿼리와 동일한 쿼리를 보여줍니다. 이 예에서 쿼리는 리더 엔드포인트를 사용하고 리더 DB 인스턴스 중 하나에서 실행됩니다. 이러한 인스턴스는 아직 재부팅되지 않았습니다. 따라서 `lower_case_table_names` 파라미터에 대한 원래 설정이 유지됩니다. 즉, 쿼리는 `foo`, `Foo` 및`FOO` 테이블 각각에 액세스할 수 있습니다.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
```

```
+--------------------------+
```

다음으로 리더 인스턴스 중 하나를 재부팅하고 다시 사용할 수 있을 때까지 기다립니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

`instance-2470`에 대한 인스턴스 엔드포인트에 연결되어 있는 동안 쿼리는 새 파라미터가 적용되었음을 보여줍니다.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+
```

이 시점에서 클러스터의 두 리더 인스턴스는 서로 다른 `lower_case_table_names` 설정으로 실행됩니다. 따라서 클러스터의 리더 엔드포인트에 대한 연결에서 이 설정에는 예측할 수 없는 값이 사용됩니다. 둘 다 일관된 설정을 갖도록 다른 리더 인스턴스를 즉시 재부팅하는 것이 중요합니다.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

다음 예제에서는 모든 리더 인스턴스의 `lower_case_table_names` 파라미터 설정이 동일한지 확인합니다. 명령은 각 리더 인스턴스의 `lower_case_table_names` 설정 값을 확인합니다. 그런 다음 리더 엔드포인트를 사용하는 동일한 명령으로 리더 엔드포인트에 대한 각 연결에 리더 인스턴스 중 하나(어느 것인지는 예측할 수 없음)가 사용되고 있음을 보여줍니다.

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
```

```
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

파라미터 변경이 모든 위치에 적용되면 `lower_case_table_names=1` 설정의 효과를 확인할 수 있습니다. 테이블이 `foo`든, `Foo`든, `FOO`든 쿼리는 이 이름을 `foo`로 변환하고 모든 경우 동일한 테이블에 액세스합니다.

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
```

```
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Amazon Aurora DB 클러스터 장애 조치

예를 들어, 프로비저닝된 라이터 DB 인스턴스를 Aurora Serverless v2 라이터 인스턴스로 교체하려는 경우 Aurora DB 클러스터 장애 조치를 수동으로 수행할 수 있습니다.

Aurora는 다음 두 방법 중 하나를 사용하여 새 기본 DB 인스턴스로 장애 조치합니다.

- 기존 리더 DB 인스턴스를 새 기본 인스턴스로 승격
- 새로운 기본 인스턴스 만들기

DB 클러스터에 하나 이상의 리더 인스턴스가 있는 경우, 리더는 장애 이벤트 중에 기본 인스턴스로 승격됩니다. DB 클러스터의 가용성을 높이려면 최소 하나 이상의 리더 인스턴스를 둘 이상의 서로 다른 가용 영역에서 생성하는 것이 좋습니다. 장애 조치 메커니즘에 대한 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 수동 장애 조치를 수행할 수 있습니다.

콘솔

DB 클러스터를 장애 조치하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 DB 클러스터에서 장애 조치할 DB 인스턴스를 선택합니다.
3. 작업(Actions)으로 장애 조치(Failover)를 선택합니다.

   확인 페이지가 표시됩니다.
4. Failover(장애 조치)를 선택합니다.

   데이터베이스 페이지에는 DB 클러스터 상태가 장애 조치 중으로 표시됩니다. 장애 조치가 완료되면 상태가 사용 가능으로 복귀하고 새 기본 DB 인스턴스와 이전 기본 DB 인스턴스의 역할이 표시됩니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터를 장애 조치하려면 failover-db-cluster 명령을 직접 호출합니다. 다음 파라미터를 지정합니다.

- `--db-cluster-identifier` – 장애 조치하려는 DB 클러스터
- `--target-db-instance-identifier` – 기본 DB 인스턴스로 승격시킬 DB 인스턴스의 이름

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds failover-db-cluster \
    --db-cluster-identifier mydbcluster \
    --target-db-instance-identifier mydbcluster-instance-2
```

Windows의 경우:

```
aws rds failover-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --target-db-instance-identifier mydbcluster-instance-2
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터를 수정하려면 FailoverDBCluster 작업을 직접 호출합니다. 다음 파라미터를 지정합니다.

- DBClusterIdentifier
- TargetDBInstanceIdentifier

Aurora DB 클러스터 및 DB 인스턴스 삭제

Aurora DB 클러스터가 더 이상 필요하지 않은 경우 삭제할 수 있습니다. 클러스터를 삭제하면 데이터가 포함된 클러스터 볼륨이 제거됩니다. 클러스터를 삭제하기 전에 데이터의 스냅샷을 저장할 수 있습니다. 나중에 스냅샷을 복원하여 동일한 데이터가 포함된 새 클러스터를 만들 수 있습니다.

클러스터 자체와 클러스터에 포함된 데이터를 보존하면서 클러스터에서 DB 인스턴스를 삭제할 수도 있습니다. DB 인스턴스를 삭제하면 클러스터가 사용량이 많지 않거나 여러 DB 인스턴스의 계산 용량이 필요하지 않은 경우 요금을 줄일 수 있습니다.

주제

Aurora DB 클러스터 삭제

Aurora 는 DB 클러스터를 삭제하는 단일 단계 방법을 제공하지 않습니다. 이 설계는 실수로 데이터가 손실되거나 애플리케이션을 오프라인으로 전환하지 않도록 하기 위한 것입니다. Aurora 애플리케이션은 일반적으로 미션 크리티컬하며 고가용성이 필요합니다. 따라서 Aurora 에서는 DB 인스턴스를 추가 및 제거하여 클러스터 용량을 쉽게 확장하거나 축소할 수 있습니다. 그러나 클러스터 자체를 제거하려면 별도로 삭제해야 할 것이 있습니다.

클러스터에서 모든 DB 인스턴스를 제거한 다음 클러스터 자체를 삭제하려면 다음 일반 절차를 따르십시오.

1. 클러스터의 모든 판독기 인스턴스를 삭제합니다. Aurora DB 클러스터에서 DB 인스턴스 삭제의 절차를 따릅니다.

   클러스터에 리더 인스턴스가 있는 경우 인스턴스 중 하나를 삭제하면 해당 클러스터의 컴퓨팅 용량만 줄어듭니다. 읽기 장치 인스턴스를 먼저 삭제하면 절차 전체에서 클러스터가 사용 가능한 상태로 유지되고 불필요한 장애 조치 (failover) 작업이 수행되지 않습니다.
2. 클러스터에서 작성기 인스턴스를 삭제합니다. 다시 한 번, Aurora DB 클러스터에서 DB 인스턴스 삭제의 절차를 따릅니다.

DB 인스턴스를 삭제하면 모든 DB 인스턴스를 삭제한 후에도 클러스터 및 연결된 클러스터 볼륨이 유지됩니다.

3. DB 클러스터를 삭제합니다.

   - AWS Management Console – 클러스터를 선택한 다음 작업 메뉴에서 삭제를 선택합니다. 나중에 필요할 경우에 대비하여 다음 옵션을 선택하여 클러스터의 데이터를 보존할 수 있습니다.
     - 클러스터 볼륨의 최종 스냅샷을 생성합니다. 기본 설정은 최종 스냅샷을 생성하는 것입니다.
     - 자동 백업을 보존합니다. 기본 설정은 자동 백업을 보존하지 않는 것입니다.

     > **Note**
     > Aurora Serverless v1 DB 클러스터의 자동 백업은 유지되지 않습니다.

     또한 Aurora에서는 클러스터를 삭제할지 확인해야 합니다.
   - CLI와 API – `delete-db-cluster` CLI 명령 또는 `DeleteDBCluster` API 작업을 호출합니다. 나중에 필요할 경우에 대비하여 다음 옵션을 선택하여 클러스터의 데이터를 보존할 수 있습니다.
     - 클러스터 볼륨의 최종 스냅샷을 생성합니다.
     - 자동 백업을 보존합니다.

     > **Note**
     > Aurora Serverless v1 DB 클러스터의 자동 백업은 유지되지 않습니다.

주제

- 빈 Aurora 클러스터 생성
- 단일 DB 인스턴스로 Aurora 클러스터 삭제
- 여러 DB 인스턴스가 있는 Aurora 클러스터 삭제

빈 Aurora 클러스터 생성

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용해 빈 DB 클러스터를 삭제할 수 있습니다.

Tip

DB 인스턴스가 없는 클러스터를 유지하여 클러스터에 CPU 요금이 부과되지 않고 데이터를 보존할 수 있습니다. 클러스터에 대해 하나 이상의 새 DB 인스턴스를 만들어 클러스터를 다시 빠르게 사용할 수 있습니다. 연결된 DB 인스턴스가 없는 상태에서 클러스터에서 Aurora특정 관리 작업을 수행할 수 있습니다. 데이터에 액세스하거나 DB 인스턴스에 연결해야 하는 작업은 수행할 수 없습니다.

콘솔

DB 클러스터를 삭제하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 삭제하려는 DB 클러스터를 선택합니다.
3. 작업에 대해 삭제를 선택합니다.
4. DB 클러스터의 최종 DB 스냅샷을 생성하려면 최종 스냅샷 생성을 선택합니다. 이것이 기본 설정입니다.
5. 최종 스냅샷을 생성하도록 선택한 경우 최종 스냅샷 이름을 입력합니다.
6. 자동 백업을 보관하려면 자동 백업 보관을 선택합니다. 이것은 기본 설정이 아닙니다.
7. 상자에 **delete me**를 입력합니다.
8. 삭제를 선택합니다.

CLI

AWS CLI를 사용하여 빈 Aurora DB 클러스터를 삭제하려면 delete-db-cluster 명령을 호출합니다.

빈 클러스터는 개발 및 테스트에만 `deleteme-zero-instances` 사용되었으며 중요한 데이터가 포함되어 있지 않다고 가정합니다. 이 경우 클러스터를 삭제할 때 클러스터 볼륨의 스냅샷을 보존할 필요가 없습니다. 다음 예시에서는 클러스터에 DB 인스턴스가 포함되어 있지 않은 것을 보여준 다음 최종 스냅샷을 만들거나 자동 백업을 보존하지 않고 빈 클러스터를 삭제합니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
```

```
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

Amazon RDS API를 사용하여 빈 Aurora DB 클러스터를 삭제하려면 DeleteDBCluster 작업을 호출합니다.

단일 DB 인스턴스로 Aurora 클러스터 삭제

DB 클러스터에 삭제 방지 기능이 활성화된 경우에도 최근 DB 인스턴스를 삭제할 수 있습니다. 이 경우 DB 클러스터 자체는 여전히 존재하고 데이터는 보존됩니다. 새로운 DB 인스턴스를 클러스터에 연결하여 데이터에 다시 액세스할 수 있습니다.

다음 예제에서는 클러스터에 연결된 DB 인스턴스가 있는 경우 `delete-db-cluster` 명령이 작동하지 않는 방법을 보여 줍니다. 이 클러스터에는 단일 작성기 DB 인스턴스가 있습니다. 클러스터의 DB 인스턴스를 검사할 때 각 인스턴스의 `IsClusterWriter` 속성을 확인합니다. 클러스터에는 0개 또는 1개의 작성기 DB 인스턴스가 있을 수 있습니다. 값은 작성기 DB 인스턴스를 `true` 나타냅니다. 값은 읽기 DB 인스턴스를 `false` 나타냅니다. 클러스터에는 0, 1 또는 여러 개의 읽기 DB 인스턴스가 있을 수 있습니다. 이 경우 `delete-db-instance` 명령을 사용하여 작성기 DB 인스턴스를 삭제합니다. DB 인스턴스가 `deleting` 상태로 전환되는 즉시 클러스터도 삭제할 수 있습니다. 또한 이 예시에서는 클러스터에 보존할 가치가 있는 데이터가 없다고 가정합니다. 따라서 클러스터 볼륨의 스냅샷을 만들거나 자동 백업을 보존하지 않습니다.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
```

```
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

여러 DB 인스턴스가 있는 Aurora 클러스터 삭제

클러스터에 DB 인스턴스가 여러 개 포함되어 있는 경우 일반적으로 하나의 작성기 인스턴스와 하나 이상의 읽기 전용 인스턴스가 있습니다. 리더 인스턴스는 작성기 인스턴스에 문제가 발생할 경우 인수하기 위해 대기 상태에 있어 고가용성을 지원합니다. 읽기 전용 인스턴스를 사용하여 쓰기 인스턴스에 오버헤드를 추가하지 않고도 읽기 집약적인 워크로드를 처리할 수 있도록 클러스터를 확장할 수도 있습니다.

읽기 장치 DB 인스턴스가 여러 개인 클러스터를 삭제하려면 먼저 읽기 장치 인스턴스를 삭제한 다음 작성기 인스턴스를 삭제합니다. 라이터 인스턴스를 삭제하면 클러스터와 해당 데이터가 그대로 유지됩니다. 클러스터는 별도의 작업을 통해 삭제합니다.

- Aurora DB 인스턴스를 삭제하는 절차는 단원을 참조하십시오 [Aurora DB 클러스터에서 DB 인스턴스 삭제](#).

- Aurora 클러스터에서 작성기 DB 인스턴스를 삭제하는 절차는 단원을 참조하십시오 단일 DB 인스턴스로 Aurora 클러스터 삭제.
- 빈 Aurora 클러스터를 삭제하는 절차는 단원을 참조하십시오 빈 Aurora 클러스터 생성.

이 CLI 예제에서는 라이터 DB 인스턴스와 단일 리더 DB 인스턴스가 모두 포함된 클러스터를 삭제하는 방법을 보여 줍니다. `describe-db-clusters` 출력은 `instance-7384` 이 작성기 `instance-1039` 인스턴스이며 읽기 인스턴스임을 보여 줍니다. 읽기 장치 인스턴스가 있는 동안 작성기 인스턴스를 삭제하면 장애 조치 (failover) 작업이 발생하기 때문에 이 예제에서는 읽기 전용 인스턴스를 먼저 삭제합니다. 해당 인스턴스도 삭제하려는 경우 읽기 인스턴스를 작성자로 승격하는 것은 의미가 없습니다. 이 경우에도 이러한 `db.t2.small` 인스턴스가 개발 및 테스트에만 사용되는 것으로 간주하고 삭제 작업이 최종 스냅샷을 건너뛰고 자동 백업을 보존하지 않습니다.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039   False
Instance:       instance-7384    True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}
```

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

다음 예에서는 작성기 DB 인스턴스와 여러 읽기 장치 DB 인스턴스가 포함된 DB 클러스터를 삭제하는 방법을 보여 줍니다. 이 작가와 리더 인스턴스의 보고서를 얻기 위해 `describe-db-clusters` 명령에서 간결한 출력을 사용합니다. 다시 말하지만, 작성기 DB 인스턴스를 삭제하기 전에 모든 판독기 DB 인스턴스를 삭제합니다. 독자 DB 인스턴스를 삭제하는 순서는 중요하지 않습니다.

DB 인스턴스가 여러 개 있는 이 클러스터에 보존할 가치가 있는 데이터가 포함되어 있다고 가정합니다. 따라서 이 예제의 `delete-db-cluster` 명령에는 생성할 스냅샷의 세부 정보를 지정하는 `--no-skip-final-snapshot` 및 `--final-db-snapshot-identifier` 매개 변수가 포함됩니다. 여기에는 자동 백업을 보존하기 위한 `--no-delete-automated-backups` 파라미터도 포함됩니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True

$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}
```

```
$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

다음 예에서는 요청된 스냅샷이 Aurora 생성되었는지 확인하는 방법을 보여 줍니다. 식별자를 지정하여 특정 스냅샷에 대한 세부 정보를 요청할 수 deleteme-multiple-readers-final-snapshot있습니다. 클러스터 식별자를 지정하여 삭제된 클러스터의 모든 스냅샷에 대한 보고서를 가져올 수도 deleteme-multiple-readers있습니다. 두 명령 모두 동일한 스냅 샷에 대한 정보를 반환합니다.

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

```
$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

Aurora 클러스터의 삭제 방지

삭제 보호가 설정된 클러스터는 삭제할 수 없습니다. 클러스터 내에서 DB 인스턴스를 삭제할 수 있지만 클러스터 자체는 삭제할 수 없습니다. 이렇게 하면 모든 데이터가 포함된 클러스터 볼륨이 실수로 삭제되지 않습니다. Aurora는 콘솔, AWS CLI 또는 RDS API를 사용하여 클러스터를 삭제하려고 할 때 DB 클러스터에 대해 삭제 방지를 적용합니다.

AWS Management Console을 사용하여 프로덕션 DB 클러스터를 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 하지만 AWS CLI 또는 API를 사용하여 클러스터를 생성하는 경우 삭제 방지가 기본적으로 비활성화됩니다. 삭제 보호를 활성화 또는 비활성화해도 중단이 발생하지 않습니다. 클러스터를 삭제하려면 클러스터를 수정하고 삭제 방지 기능을 비활성화합니다. 삭제 방지 활성화 및 비활성화에 대한 자세한 내용은 콘솔, CLI, API를 사용하여 DB 클러스터 수정 단원을 참조하십시오.

> **Tip**
>
> 모든 DB 인스턴스가 삭제되더라도 클러스터에 새 DB 인스턴스를 만들어 데이터에 액세스할 수 있습니다.

중지된 Aurora 클러스터 삭제

클러스터가 `stopped` 상태인 경우 삭제할 수 없습니다. 이 경우 클러스터를 삭제하기 전에 클러스터를 시작하십시오. 자세한 내용은 Aurora DB 클러스터 시작 섹션을 참조하세요.

읽기 전용 복제본인 Aurora MySQL 클러스터 삭제

Aurora MySQL에서는 다음 조건에 해당할 경우 DB 클러스터에서 DB 인스턴스를 삭제할 수 없습니다.

- DB 클러스터는 다른 Aurora DB 클러스터의 읽기 전용 복제본입니다.
- DB 인스턴스는 DB 클러스터의 유일한 인스턴스입니다.

이 경우 DB 인스턴스를 삭제하려면 더 이상 읽기 전용 복제본이 되지 않도록 먼저 DB 클러스터를 승격합니다. 승격이 완료된 후 DB 클러스터에서 최종 DB 인스턴스를 삭제할 수 있습니다. 자세한 내용은 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 섹션을 참조하세요.

클러스터를 삭제할 때의 최종 스냅샷

이 섹션에서는 Aurora 클러스터를 삭제할 때 최종 스냅샷을 생성할지 여부를 선택하는 방법을 보여 줍니다. 최종 스냅샷을 생성하도록 선택했지만 지정한 이름이 기존 스냅샷과 일치하는 경우 작업이 중지되고 오류가 발생합니다. 이 경우 스냅샷 세부 정보를 검토하여 현재 세부 정보를 나타내는지 또는 이전 스냅샷인지 확인합니다. 기존 스냅샷에 보존할 최신 데이터가 없는 경우 스냅샷 이름을 변경하고 다시 시도하거나 최종 스냅샷 파라미터에 다른 이름을 지정하세요.

Aurora DB 클러스터에서 DB 인스턴스 삭제

전체 클러스터를 삭제하는 프로세스의 일부로 Aurora DB 클러스터에서 DB 인스턴스를 삭제할 수 있습니다. 클러스터에 특정 수의 DB 인스턴스가 포함되어 있는 경우 클러스터를 삭제하려면 각 DB 인스턴스를 삭제해야 합니다. 클러스터를 실행 중인 상태로 둔 상태에서 클러스터에서 하나 이상의 판독기 인스턴스를 삭제할 수도 있습니다. 클러스터가 사용량이 아닌 경우 계산 용량 및 관련 요금을 줄이기 위해 이렇게 할 수 있습니다.

DB 인스턴스를 삭제하려면 인스턴스의 이름을 지정합니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 인스턴스를 삭제할 수 있습니다.

> **Note**
>
> Aurora 복제본이 삭제되면 인스턴스 엔드포인트가 즉시 제거되고 Aurora 복제본이 리더 엔드포인트에서 제거됩니다. 삭제되는 Aurora 복제본을 실행하는 문이 있는 경우 3분의 유예 기간이 있습니다. 기존 문은 유예 기간 동안 완료할 수 있습니다. 유예 기간이 종료되면 Aurora 복제본이 종료 및 삭제됩니다.

Aurora DB 클러스터의 경우 DB 인스턴스를 삭제해도 전체 클러스터가 삭제되지는 않습니다. 클러스터가 사용 중이 아닐 때 Aurora 클러스터에서 DB 인스턴스를 삭제하여 컴퓨팅 파워와 관련 요금을 줄일 수 있습니다. DB 인스턴스가 하나 또는 DB 인스턴스가 0인 Aurora 클러스터의 특수한 상황에 대한

자세한 내용은 단일 DB 인스턴스로 Aurora 클러스터 삭제 및 단원을 참조하십시오 빈 Aurora 클러스터 생성.

> **Note**
>
> 삭제 방지가 활성화된 DB 클러스터는 삭제할 수 없습니다. 자세한 내용은 Aurora 클러스터의 삭제 방지 섹션을 참조하세요.
>
> DB 클러스터를 수정하여 삭제 방지를 비활성화할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 단원을 참조하십시오.

콘솔

DB 클러스터에서 DB 인스턴스를 삭제하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 후 삭제하려는 DB 인스턴스를 선택합니다.
3. [Actions]에 대해 [Delete]를 선택합니다.
4. 상자에 **`delete me`**를 입력합니다.
5. 삭제를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스를 삭제하려면 delete-db-instance 명령을 호출하고 `--db-instance-identifier` 값을 지정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

Windows의 경우:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

Amazon RDS API를 사용하여 DB 인스턴스를 삭제하려면 DeleteDBInstance 작업을 호출하고 `DBInstanceIdentifier` 파라미터를 지정합니다.

> **Note**
>
> DB 인스턴스의 상태가 `deleting`인 경우 CA 인증서 값이 RDS 콘솔이나 AWS CLI 명령 또는 RDS API 작업의 출력에 나타나지 않습니다. CA 인증서에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오.

Amazon Aurora 및 Amazon RDS 리소스에 태그 지정

Amazon RDS 태그는 사용자가 정의하고 DB 인스턴스 또는 DB 스냅샷과 같은 Amazon RDS 리소스와 연결하는 이름-값 페어입니다. 이 이름을 키라고 합니다. 원하는 경우 키에 값을 입력할 수 있습니다.

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 Amazon RDS 리소스에서 태그를 추가, 나열 및 삭제할 수 있습니다. CLI 또는 API를 사용할 때 사용할 RDS 리소스에 대한 Amazon 리소스 이름(ARN)을 제공해야 합니다. ARN 생성에 대한 자세한 내용은 Amazon RDS의 ARN 구성 주제단원을 참조하십시오.

태그를 사용하여 Aurora 및 Amazon RDS 리소스에 메타데이터를 추가할 수 있습니다. 태그를 사용하여 데이터베이스 인스턴스, 스냅샷, Aurora 클러스터 등에 대한 고유한 표기법을 추가할 수 있습니다. 이렇게 하면 Aurora 및 Amazon RDS 리소스를 문서화하는 데 도움이 될 수 있습니다. 자동화된 유지 관리 절차와 함께 태그를 사용할 수도 있습니다.

특히 이러한 태그를 IAM 정책에 사용할 수 있습니다. 태그를 사용하여 Aurora 및 Amazon RDS 리소스에 대한 액세스를 관리하고 해당 리소스에 적용 가능한 작업을 제어할 수 있습니다. 또한 비슷하게 태그가 지정된 리소스에 대한 비용을 그룹화하여 이러한 태그로 비용을 추적할 수 있습니다.

다음 Aurora 및 Amazon RDS 리소스에 태그를 지정할 수 있습니다.

- DB 인스턴스
- DB 클러스터
- Aurora 글로벌 클러스터
- DB 클러스터 엔드포인트
- 읽기 전용 복제본
- DB 스냅샷
- DB 클러스터 스냅샷
- 예약 DB 인스턴스
- 이벤트 구독
- DB 옵션 그룹
- DB 파라미터 그룹
- DB 클러스터 파라미터 그룹
- DB 서브넷 그룹
- RDS 프록시

- RDS Proxy 엔드포인트
- 블루/그린 배포
- 제로 ETL 통합

> **Note**
>
> 현재는 AWS Management Console을 사용하여 RDS 프록시 및 RDS 프록시 엔드포인트에 태그를 지정할 수 없습니다.

주제

Amazon RDS 리소스 태그를 사용하는 이유는 무엇인가요?

태그를 사용하여 다음을 수행할 수 있습니다.

- 애플리케이션, 프로젝트, 부서, 환경 등을 기준으로 RDS 리소스를 분류하세요. 예를 들어 태그 키를 사용하여 범주를 정의하고 태그 값은 이 범주의 항목으로 정의할 수 있습니다. `environment=prod` 태그를 생성할 수도 있습니다. 또는 `project`의 태그 키와 `Salix`의 태그 값을 정의하여 Amazon RDS 리소스가 Salix 프로젝트에 할당되었음을 나타낼 수 있습니다.
- 리소스 관리 작업을 자동화하세요. 예를 들어, `environment=prod` 태그가 지정된 인스턴스에 대한 유지 보수 창을 `environment=test` 태그가 지정된 인스턴스에 대한 창과 다르게 만들 수 있습니다. `environment=prod` 태그가 지정된 인스턴스에 대한 자동 DB 스냅샷을 구성할 수도 있습니다.
- IAM 정책 내에서 RDS 리소스에 대한 액세스를 제어하세요. 전역 `aws:ResourceTag/`*`tag-key`* 조건 키를 사용하면 됩니다. 예를 들어, 정책에서 DBAdmin 그룹의 사용자만 `environment=prod` 태그가 지정된 DB 인스턴스를 수정하도록 허용할 수 있습니다. IAM 정책으로 태그가 지정된 리소스에 대한 액세스를 관리하는 방법에 대한 자세한 내용은 AWS ID 및 액세스 관리 사용 설명서에서 Amazon Aurora의 자격 증명 및 액세스 관리 및 AWS에 대한 액세스 제어를 참조하세요.

- 태그를 기반으로 리소스를 모니터링하세요. 예를 들어, `environment=prod` 태그가 지정된 DB 인스턴스에 대한 Amazon CloudWatch 대시보드를 만들 수 있습니다.
- 유사하게 태그가 지정된 리소스에 대한 비용을 그룹화하여 비용을 추적하세요. 예를 들어, Salix 프로젝트와 관련된 RDS 리소스에 `project=Salix` 태그를 지정하면 이 프로젝트에 대한 비용 보고서를 생성하고 비용을 할당할 수 있습니다. 자세한 내용은 Amazon RDS에서 태그와 함께 AWS 결제가 작동하는 방식 단원을 참조하십시오.

Amazon RDS 리소스 태그 작동 방식

AWS은(는) 태그에 의미론적 의미를 적용하지 않습니다. 태그는 엄격히 문자열로 해석됩니다.

주제

Amazon RDS의 태그 세트

모든 Amazon RDS 리소스에는 태그 세트라는 컨테이너가 있습니다. 컨테이너에는 리소스에 할당된 모든 태그가 포함됩니다. 리소스에는 정확히 하나의 태그 세트가 있습니다.

태그 세트 하나에는 0~50개의 태그가 포함됩니다. 기존 리소스 태그와 키가 동일한 태그를 RDS 리소스에 추가하면 새 값이 이전 값을 덮어씁니다.

Amazon RDS의 태그 구조

RDS 태그의 구조는 다음과 같습니다

태그 키

키는 태그의 필수 이름입니다. 문자열 값은 1~128자의 유니코드 문자 길이여야 하며 앞에 `aws:` 또는 `rds:` 접두사를 붙일 수 없습니다. 문자열에는 유니코드 문자, 숫자, 공백, `_`, `.`, `:`, `/`, `=`, `+`, `-` 및 `@`의 세트만 포함할 수 있습니다. Java 정규식은 `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`입니다. 태그 키는 대/소문자를 구분합니다. 따라서 `project`와 `Project`는 구별됩니다.

키는 태그 세트에 고유합니다. 예를 들어, 태그 세트에 `project=Trinity`와 `project=Xanadu`처럼 키는 같지만 값은 다른 키-페어가 있을 수 없습니다.

태그 값

값은 태그의 선택적 문자열 값입니다. 문자열 값은 1~256자의 유니코드 문자여야 합니다. 문자열에는 유니코드 문자, 숫자, 공백, _, ., :, /, =, +, - 및 @의 세트만 포함할 수 있습니다. Java 정규식은 `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`입니다. 태그 값은 대소문자를 구분합니다. 따라서 `prod` 값과 `Prod`은 구별됩니다.

태그 세트에서 값이 고유할 필요는 없으며 null일 수도 있습니다. 예를 들어 `project=Trinity` 및 `cost-center=Trinity`의 태그 세트에 키-값 페어가 있을 수 있습니다.

태그 지정이 가능한 Amazon RDS 리소스

다음 Amazon RDS 리소스에 태그를 지정할 수 있습니다.

- DB 인스턴스
- DB 클러스터
- DB 클러스터 엔드포인트
- 읽기 전용 복제본
- DB 스냅샷
- DB 클러스터 스냅샷
- 예약 DB 인스턴스
- 이벤트 구독
- DB 옵션 그룹
- DB 파라미터 그룹
- DB 클러스터 파라미터 그룹
- DB 서브넷 그룹
- RDS 프록시
- RDS Proxy 엔드포인트

> **Note**
>
> 현재는 AWS Management Console을 사용하여 RDS 프록시 및 RDS 프록시 엔드포인트에 태그를 지정할 수 없습니다.

- 블루/그린 배포
- 제로 ETL 통합(미리 보기)

Amazon RDS에서 태그와 함께 AWS 결제가 작동하는 방식

태그를 사용하여 비용 구조를 반영하도록 AWS 청구서를 구성합니다. 이렇게 하려면 가입하여 태그 키 값이 포함된 AWS 계정 청구서를 가져옵니다. 그런 다음 같은 태그 키 값을 가진 리소스에 따라 결제 정보를 구성하여 리소스 비용의 합을 볼 수 있습니다. 예를 들어, 특정 애플리케이션 이름으로 여러 리소스에 태그를 지정한 다음 결제 정보를 구성하여 여러 서비스에 걸친 해당 애플리케이션의 총 비용을 볼 수 있습니다. 자세한 내용은 AWS Billing 사용 설명서의 비용 할당 태그 사용을 참조하십시오.

비용 할당 태그가 DB 클러스터 스냅샷에서 작동하는 방식

DB 클러스터 스냅샷에 태그를 추가할 수 있습니다. 그러나 청구서에는 이러한 그룹화가 반영되지 않습니다. 비용 할당 태그를 DB 클러스터 스냅샷에 적용하려면 다음 조건을 충족해야 합니다.

- 태그가 상위 DB 인스턴스에 연결되어 있어야 합니다.
- 상위 DB 인스턴스가 DB 클러스터 스냅샷과 동일한 AWS 계정 계정에 존재해야 합니다.
- 상위 DB 인스턴스가 DB 클러스터 스냅샷과 동일한 AWS 리전 계정에 존재해야 합니다.

DB 클러스터 스냅샷이 상위 DB 인스턴스와 동일한 리전에 존재하지 않거나 상위 DB 인스턴스가 삭제된 경우 분리된 것으로 간주됩니다. 분리된 DB 스냅샷은 비용 할당 태그를 지원하지 않습니다. 분리된 스냅샷에 대한 비용은 태그가 지정되지 않은 품목에 집계됩니다. 다음 조건을 충족하는 경우 크로스 계정 DB 클러스터 스냅샷은 분리된 것으로 간주되지 않습니다.

- 상위 DB 인스턴스와 동일한 리전에 있어야 합니다.
- 상위 DB 인스턴스는 소스 계정에서 소유합니다.

> **Note**
> 상위 DB 인스턴스가 다른 계정에서 소유하고 있는 경우, 대상 계정의 크로스 계정 스냅샷에는 비용 할당 태그가 적용되지 않습니다.

Amazon RDS 리소스에 태그를 지정하는 모범 사례

태그를 사용할 때는 다음 모범 사례를 준수하는 것이 좋습니다.

- 조직의 모든 팀이 따르는 태그 사용 규칙을 문서화하세요. 특히 이름을 설명적이면서도 일관성 있게 지어야 합니다. 예를 들어 일부 리소스에 `env:production`으로 태그를 지정하는 대신 `environment:prod` 형식으로 표준화하세요.

> **Important**
> 개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 태그에 저장하지 마세요.

- 태그 지정을 자동화하여 일관성을 보장하세요. 예를 들어 다음과 같은 기술을 사용할 수 있습니다.
  - AWS CloudFormation 템플릿에 태그를 포함하세요. 템플릿으로 리소스를 만들면 리소스에 자동으로 태그가 지정됩니다.
  - AWS Lambda 함수를 사용하여 태그를 정의하고 적용하세요.
  - RDS 리소스에 태그를 추가하는 단계가 포함된 SSM 문서를 생성하세요.
- 필요한 경우에만 태그를 사용하세요. 단일 RDS 리소스에 대해 최대 50개의 태그를 추가할 수 있지만, 불필요한 태그의 증가와 복잡성을 피하는 것이 가장 좋습니다.
- 태그의 관련성과 정확성을 정기적으로 검토하세요. 필요에 따라 오래된 태그를 제거하거나 수정하세요.
- AWS Management Console의 AWS Tag Editor를 사용하여 태그를 생성해 보세요. Tag Editor를 사용하여 RDS 리소스를 포함하여 지원되는 여러 AWS 리소스에 동시에 태그를 추가할 수 있습니다. 자세한 내용은 AWS 리소스 그룹 사용 설명서의 [Tag Editor](https://docs.aws.amazon.com/tag-editor/latest/userguide/tag-editor.html)를 참조하세요.

DB 클러스터 스냅샷에 태그 복사

DB 클러스터를 생성하거나 복원할 때 DB 클러스터의 태그가 클러스터의 스냅샷에 복사되도록 지정할 수 있습니다. 태그를 복사하면 DB 스냅샷의 메타데이터가 원본 DB 클러스터의 메타데이터와 일치하게 됩니다. DB 스냅샷의 액세스 정책 또한 원본 DB 클러스터의 액세스 정책과 같아집니다. 태그는 기본적으로 복사되지 않습니다.

다음 작업 시 DB 스냅샷으로 태그를 복사하도록 지정할 수 있습니다.

- DB 클러스터 생성
- DB 클러스터 복원
- 읽기 전용 복제본 생성
- DB 클러스터 스냅샷 복사

> **Note**
>
> 경우에 따라 create-db-snapshot AWS CLI 명령의 `--tags` 파라미터 값을 포함할 수 있습니다. 또는 CreateDBSnapshot API 작업에 하나 이상의 태그를 제공할 수도 있습니다. 이러한 경우 RDS는 원본 DB 인스턴스에서 새 DB 스냅샷으로 태그를 복사하지 않습니다. 이 기능은 원본 DB 인스턴스에 `--copy-tags-to-snapshot`(`CopyTagsToSnapshot`) 옵션이 활성화되어 있어도 적용됩니다.
> 이 방법을 사용할 경우 DB 스냅샷에서 DB 인스턴스의 복사본을 생성할 수 있습니다. 이 접근 방식을 사용하면 새 DB 인스턴스에 적용되지 않는 태그를 추가하는 것을 피할 수 있습니다. AWS CLI `create-db-snapshot` 명령(또는 `CreateDBSnapshot` RDS API 작업)을 사용하여 DB 스냅샷을 생성합니다. DB 스냅샷을 생성한 후 이 주제의 뒷부분에서 설명된 대로 태그를 추가할 수 있습니다.

Amazon RDS에서 태그 추가 및 삭제

다음을 수행할 수 있습니다.

- 리소스를 생성할 때 태그를 생성합니다(예: AWS CLI 명령 `create-db-instance`를 실행할 때).
- `add-tags-to-resource` 명령을 사용하여 기존 리소스에 태그를 추가합니다.
- `list-tags-for-resource` 명령을 사용하여 특정 리소스와 연결된 태그를 나열합니다.
- `add-tags-to-resource` 명령을 사용하여 태그를 업데이트합니다.
- `remove-tags-from-resource` 명령을 사용하여 리소스에서 태그를 제거합니다.

다음 절차에서는 DB 인스턴스 및 Aurora DB 클러스터와 관련된 리소스에 대해 일반적인 태그 지정 작업을 수행하는 방법을 설명합니다. 권한 부여 목적으로 태그가 캐시됩니다. 이러한 이유로 Amazon RDS 리소스에서 태그를 추가하거나 업데이트할 때 수정 사항을 사용할 수 있기까지 몇 분이 걸릴 수 있습니다.

콘솔

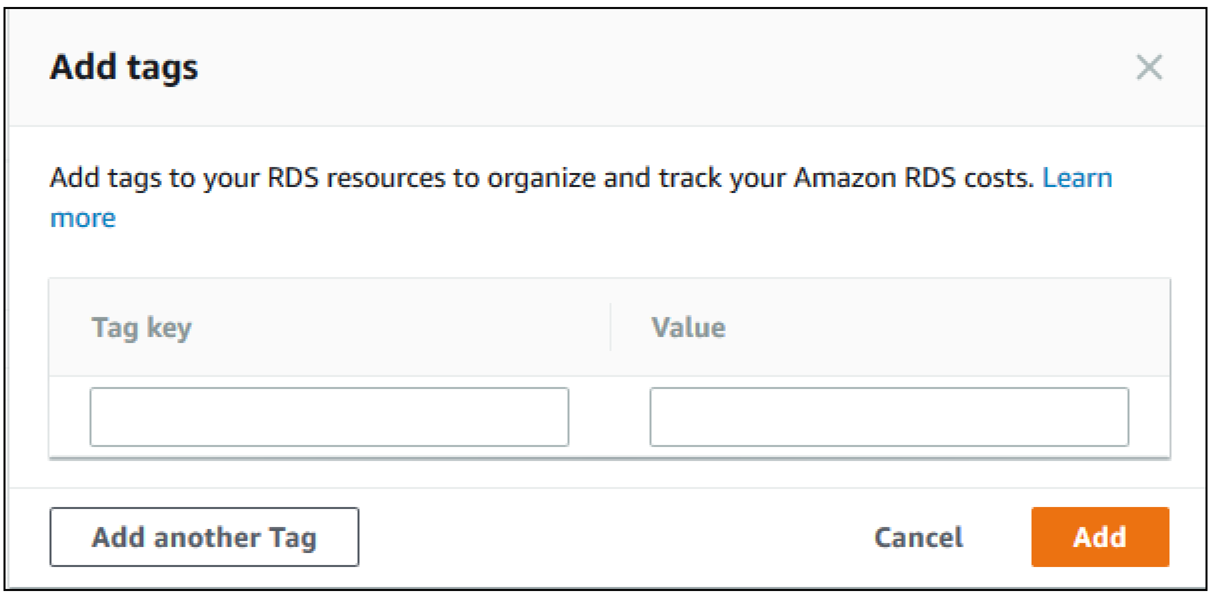
Amazon RDS 리소스에 태그를 지정하는 프로세스는 모든 리소스에서 비슷합니다. 다음 절차에서는 Amazon RDS DB 인스턴스에 태그를 지정하는 방법을 보여줍니다.

DB 인스턴스에 태그를 추가하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

> **Note**
>
> DB 인스턴스 목록을 필터링하려면 데이터베이스 창의 Filter databases(데이터베이스 필터링)에 텍스트 문자열을 입력합니다. 해당 문자열을 포함하는 DB 인스턴스만 표시됩니다.

3. 세부 정보를 보기 위해 태그 지정하려는 DB 인스턴스의 이름을 선택합니다.
4. 세부 정보 섹션에서 아래에 있는 태그 섹션으로 스크롤합니다.
5. [추가]를 선택합니다. 태그 추가 창이 나타납니다.



6. 태그 키와 값에 값을 입력합니다.
7. 다른 태그를 추가하려면 다른 태그 추가를 선택하고 태그 키와 값에 값을 입력합니다.

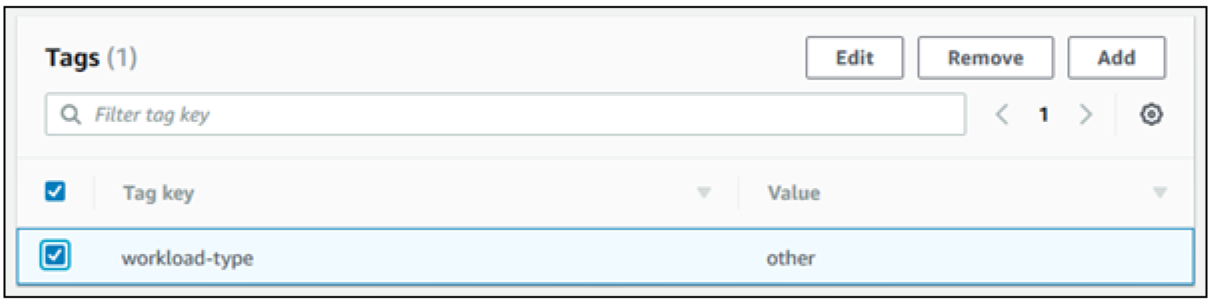
   이 단계를 필요한 만큼 반복합니다.
8. [추가]를 선택합니다.

DB 인스턴스에서 태그를 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

> **Note**
>
> DB 인스턴스 목록을 필터링하려면 데이터베이스 창의 Filter databases(데이터베이스 필터링) 상자에 텍스트 문자열을 입력합니다. 해당 문자열을 포함하는 DB 인스턴스만 표시됩니다.

3. 세부 정보를 표시할 DB 인스턴스의 이름을 선택합니다.
4. 세부 정보 섹션에서 아래에 있는 태그 섹션으로 스크롤합니다.
5. 삭제하려는 태그를 선택합니다.



6. Delete(삭제)를 선택한 다음 Delete tags(삭제 태그)창에서 Delete(삭제)를 선택합니다.

AWS CLI

AWS CLI 사용을 통해 DB 인스턴스에 대한 태그를 추가, 나열 또는 제거할 수 있습니다.

- Amazon RDS 리소스에 하나 이상의 태그를 추가하려면 AWS CLI 명령 `add-tags-to-resource`를 사용합니다.
- Amazon RDS 리소스의 태그를 나열하려면 AWS CLI 명령 `list-tags-for-resource`를 사용합니다.
- Amazon RDS 리소스에서 하나 이상의 태그를 삭제하려면 AWS CLI 명령 `remove-tags-from-resource`를 사용합니다.

필수 ARN을 생성하는 방법에 대해 자세히 알아보려면 Amazon RDS의 ARN 구성 단원을 참조하십시오.

RDS API

Amazon RDS API를 사용하여 DB 인스턴스에 대한 태그를 추가, 나열 또는 제거할 수 있습니다.

- Amazon RDS 리소스에 태그를 추가하려면 AddTagsToResource 작업을 사용합니다.
- Amazon RDS 리소스에 배정된 태그를 나열하려면 ListTagsForResource를 사용합니다.
- Amazon RDS 리소스에서 태그를 제거하려면 RemoveTagsFromResource 작업을 사용합니다.

필수 ARN을 생성하는 방법에 대해 자세히 알아보려면 Amazon RDS의 ARN 구성 단원을 참조하십시오.

Amazon RDS API를 사용한 XML 작업 시 다음 스키마를 사용하십시오.

```
<Tagging>
      <TagSet>
          <Tag>
              <Key>Project</Key>
              <Value>Trinity</Value>
          </Tag>
          <Tag>
              <Key>User</Key>
              <Value>Jones</Value>
          </Tag>
      </TagSet>
  </Tagging>
```

다음 표에는 허용되는 XML 태그와 해당 특성의 목록이 나와 있습니다. Key와 Value은 대/소문자를 구분합니다. 예를 들어 project=Trinity와 PROJECT=Trinity는 서로 다른 태그입니다.

태그 지정 요소	설명
TagSet	태그 세트에는 Amazon RDS 리소스에 배정된 모든 태그가 포함됩니다. 리소스당 하나의 태그 세트만 있을 수 있습니다. Amazon RDS API를 통해서만 TagSet로 작업합니다.
Tag	태그는 사용자가 정의하는 키-값 페어입니다. 태그 세트에 1~50개의 태그가 있을 수 있습니다.

태그 지정 요소	설명
키	키는 태그의 필수 이름입니다. 제한 사항에 대한 자세한 내용은 Amazon RDS의 태그 구조 단원을 참조하세요. 문자열 값은 길이가 1~128자인 유니코드 문자이며 `aws:` 또는 `rds:`를 접두사로 사용할 수 없습니다. 문자열에는 유니코드 문자, 숫자, 공백, '_', '.', '/', '=', '+', '-'(Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$")만 포함될 수 있습니다. 키는 태그 집합에 대해 고유해야 합니다. 예를 들어, 태그 세트에 project/Trinity와 project/Xanadu처럼 키는 같지만 값은 다른 키-페어가 있을 수 없습니다.
값	값은 태그의 선택적 값입니다. 제한 사항은 을 참조하십시오Amazon RDS의 태그 구조 단원을 참조하세요. 문자열 값은 길이가 1~256자인 유니코드 문자이며 `aws:` 또는 `rds:`를 접두사로 사용할 수 없습니다. 문자열에는 유니코드 문자, 숫자, 공백, '_', '.', '/', '=', '+', '-'(Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$")만 포함될 수 있습니다. 값은 태그 세트에서 고유할 필요는 없으며 null일 수 있습니다. 예를 들어, project/Trinity 및 cost-center/Trinity의 태그 세트에 키-값 페어가 있을 수 있습니다.

자습서: 태그를 사용하여 중지할 Aurora DB 클러스터 지정

개발 환경이나 테스트 환경에서 여러 Aurora DB 클러스터를 생성한다고 가정합니다. 며칠 동안 이러한 클러스터를 모두 유지해야 합니다. 어떤 클러스터는 밤새 테스트를 실행합니다. 또 다른 클러스터는 밤새 중단하고 다음 날 다시 시작할 수 있습니다. 다음 예제에서는 밤새 중단하기에 적합한 클러스터에 태그를 할당하는 방법을 보여줍니다. 그런 다음 이 예제에서는 스크립트가 해당 태그가 있는 클러스터를 감지한 후 해당 클러스터를 중지하는 방법을 보여줍니다. 이 예제에서 키-값 쌍의 값 부분은 중요하지 않습니다. `stoppable` 태그가 있다는 것은 클러스터에 이 사용자 정의 속성이 있음을 의미합니다.

중지할 Aurora DB 클러스터를 지정하려면

1. 중지 가능으로 지정하려는 클러스터의 ARN을 결정합니다.

   태그 지정을 위한 명령 및 API는 ARN과 함께 작동합니다. 이렇게 하면 AWS 리전, AWS 계정 및 동일한 짧은 이름을 가질 수 있는 다양한 유형의 리소스 간에 원활하게 작동할 수 있습니다. 클러

스터에서 작동하는 CLI 명령에서 클러스터 ID 대신 ARN을 지정할 수 있습니다. 사용자 클러스터의 이름을 *dev-test-cluster*로 대체합니다. ARN 파라미터를 사용하는 후속 명령에서, 사용자 클러스터의 ARN을 대체합니다. ARN에는 사용자의 AWS 계정 ID와 클러스터가 위치한 AWS 리전의 이름이 포함되어 있습니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
  --query "*[].{DBClusterArn:DBClusterArn}" --output text
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
```

2. `stoppable` 태그를 이 클러스터에 추가합니다.

   이 태그의 이름은 사용자가 선택합니다. 이 방법을 사용하면 모든 관련 정보를 이름에 인코딩하는 명명 규칙을 만드는 것을 피할 수 있습니다. 이러한 규칙 하에서는 DB 인스턴스 이름 또는 다른 리소스의 이름에 정보를 인코딩할 수 있습니다. 이 예제에서는 태그를 있거나 없는 속성으로 취급하기 때문에 `Value=` 파라미터의 `--tags` 일부를 생략합니다.

```
$ aws rds add-tags-to-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
  --tags Key=stoppable
```

3. 태그가 클러스터에 있는지 확인합니다.

   이러한 명령은 클러스터에 대한 태그 정보를 JSON 형식 및 탭으로 구분된 일반 텍스트로 검색합니다.

```
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. stoppable(으)로 지정된 모든 클러스터를 중지하려면 모든 클러스터 목록을 준비합니다. 목록을 반복하고 각 클러스터에 관련 속성으로 태그가 지정되어 있는지 확인합니다.

   이 Linux 예제에서는 셸 스크립팅을 사용하여 클러스터 ARN 목록을 임시 파일에 저장한 후 각 클러스터에 대해 CLI 명령을 수행합니다.

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

매일 일과가 끝날 때 이와 같은 스크립트를 실행하여 불필요한 클러스터가 중지되도록 할 수 있습니다. 매일 밤 이러한 검사를 수행하기 위해 cron 등의 유틸리티를 사용하여 작업을 예약할 수도 있습니다. 예를 들어, 일부 클러스터가 실수로 실행 중으로 남아 있지 않도록 하려면 검사를 예약하면 됩니다. 이때 검사할 클러스터 목록을 준비하는 명령을 세부 조정할 수 있습니다.

다음 명령은 클러스터 목록을 생성하지만 `available` 상태에 있는 클러스터만 생성합니다. 스크립트는 이미 중지된 클러스터를 무시할 수 있습니다. 이러한 클러스터는 `stopped` 또는 `stopping` 등 상태 값이 다르기 때문입니다.

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> 태그를 할당하고 해당 태그가 있는 클러스터를 찾는 기능을 사용하여 다른 방법으로 비용을 줄일 수 있습니다. 개발 및 테스트에 사용되는 Aurora DB 클러스터를 예로 들어 보겠습니다. 여기서 일과가 끝날 때 일부 클러스터를 삭제하도록 지정하거나 해당 클러스터의 리더 DB 인스턴스만 삭제하도록 지정할 수도 있습니다. 또는 사용량이 낮을 것으로 예상되는 시기에 DB 인스턴스를 소규모 DB 인스턴스 클래스로 변경하도록 지정할 수 있습니다.

Amazon RDS의 Amazon 리소스 이름(ARN)

Amazon Web Services에 생성되는 리소스는 각기 고유한 Amazon 리소스 이름(ARN)으로 식별됩니다. 특정 Amazon RDS 작업에서는 ARN을 지정하여 Amazon RDS 리소스를 고유한 이름으로 식별해야 합니다. 예를 들어, RDS DB 인스턴스 읽기 전용 복제본을 생성할 때 원본 DB 인스턴스에 대한 ARN을 제공해야 합니다.

ARN 구성 및 기존 ARN 가져오기에 대한 자세한 내용은 아래의 주제를 참조하세요.

주제

- Amazon RDS의 ARN 구성
- Amazon RDS의 기존 ARN 가져오기

Amazon RDS의 ARN 구성

Amazon Web Services에 생성되는 리소스는 각기 고유한 Amazon 리소스 이름(ARN)으로 식별됩니다. 다음 구문을 사용하여 Amazon RDS 리소스에 대한 ARN을 생성할 수 있습니다.

```
arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

리전 이름	지역	엔드포인트	프로토콜	
미국 동부 (오하이오)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	
미국 동부 (버지니아 북부)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
		rds-fips.us-east-1.api.aws	HTTPS	
		rds-fips.us-east-1.amazonaws.com	HTTPS	
		rds.us-east-1.api.aws	HTTPS	

리전 이름	지역	엔드포인트	프로토콜
미국 서부 (캘리포니아 북부)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
미국 서부 (오레곤)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
아프리카 (케이프타운)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
아시아 태평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
아시아 태평양(하이데라바드)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS
아시아 태평양(자카르타)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
		rds.ap-southeast-3.api.aws	HTTPS
아시아 태평양(말레이시아)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS

리전 이름	지역	엔드포인트	프로토콜
아시아 태평양(멜버른)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com rds.ap-southeast-4.api.aws	HTTPS HTTPS
아시아 태평양(뭄바이)	ap-south-1	rds.ap-south-1.amazonaws.com rds.ap-south-1.api.aws	HTTPS HTTPS
아시아 태평양(오사카)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com rds.ap-northeast-3.api.aws	HTTPS HTTPS
아시아 태평양(서울)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com rds.ap-northeast-2.api.aws	HTTPS HTTPS
아시아 태평양(싱가포르)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com rds.ap-southeast-1.api.aws	HTTPS HTTPS
아시아 태평양(시드니)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com rds.ap-southeast-2.api.aws	HTTPS HTTPS
아시아 태평양(타이베이)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
아시아 태평양(태국)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
아시아 태평양(도쿄)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com rds.ap-northeast-1.api.aws	HTTPS HTTPS

리전 이름	지역	엔드포인트	프로토콜
캐나다(중부)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
캐나다 서부(캘거리)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS
유럽(프랑크푸르트)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
		rds.eu-central-1.api.aws	HTTPS
유럽(아일랜드)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
		rds.eu-west-1.api.aws	HTTPS
유럽(런던)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
		rds.eu-west-2.api.aws	HTTPS
유럽(밀라노)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
		rds.eu-south-1.api.aws	HTTPS
유럽(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
		rds.eu-west-3.api.aws	HTTPS
유럽(스페인)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
		rds.eu-south-2.api.aws	HTTPS
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
		rds.eu-north-1.api.aws	HTTPS

리전 이름	지역	엔드포인트	프로토콜
유럽(취리히)	eu-central-2	rds.eu-central-2.amazonaws.com rds.eu-central-2.api.aws	HTTPS HTTPS
이스라엘(텔아비브)	il-central-1	rds.il-central-1.amazonaws.com rds.il-central-1.api.aws	HTTPS HTTPS
멕시코(중부)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
중동(바레인)	me-south-1	rds.me-south-1.amazonaws.com rds.me-south-1.api.aws	HTTPS HTTPS
중동(UAE)	me-central-1	rds.me-central-1.amazonaws.com rds.me-central-1.api.aws	HTTPS HTTPS
남아메리카(상파울루)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud(미국 동부)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud(미국 서부)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

다음 표에는 특정 Amazon RDS 리소스 유형에 대한 ARN 생성 시 사용해야 하는 형식이 나와 있습니다.

리소스 유형	ARN 형식
DB 인스턴스	arn:aws:rds:*<region>*:*<account>* :db:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
DB 클러스터	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1`
이벤트 구독	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
DB 파라미터 그룹	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB 클러스터 파라미터 그룹	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`
예약 DB 인스턴스	arn:aws:rds:*<region>*:*<account>* :ri:*<name>*

리소스 유형	ARN 형식
	예: `arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql`
DB 보안 그룹	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public`
자동화된 DB 스냅샷	arn:aws:rds:*<region>*:*<account>* :snapshot:rds:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
자동화된 DB 클러스터 스냅샷	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:rds:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
수동 DB 스냅샷	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`

리소스 유형	ARN 형식
수동 DB 클러스터 스냅샷	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* 예: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:  my-aurora-cluster-snap`
DB 서브넷 그룹	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 예시: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

Amazon RDS의 기존 ARN 가져오기

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 RDS API를 사용해 RDS 리소스에 대한 ARN을 가져올 수 있습니다.

콘솔

AWS Management Console에서 ARN을 확인하려면 ARN을 보려는 리소스를 탐색하거나, 해당 리소스의 세부 정보를 확인합니다.

예를 들어, DB 클러스터 세부 정보의 구성 탭에서 DB 클러스터의 ARN을 가져올 수 있습니다.

AWS CLI

AWS CLI에서 특정 RDS 리소스에 대한 ARN을 가져오려면 해당 리소스에 `describe` 명령을 사용합니다. 다음 표에서는 각 AWS CLI ARN 명령, 그리고 ARN을 가져오기 위해 명령과 함께 사용하는 ARN 속성을 보여 줍니다.

AWS CLI 명령	ARN 속성
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn

AWS CLI 명령	ARN 속성
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

예를 들어, 다음 AWS CLI 명령은 DB 인스턴스에 대한 ARN을 가져옵니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Windows의 경우:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

이 명령의 출력은 다음과 같습니다.

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

특정 RDS 리소스에 대한 ARN을 확인하기 위해 다음과 같이 RDS API 작업을 호출하고 ARN 속성을 사용할 수 있습니다.

RDS API 작업	ARN 속성
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora 업데이트

Amazon Aurora는 정기적으로 업데이트를 릴리스합니다. 업데이트는 시스템 유지 관리 기간 중에 Amazon Aurora DB 클러스터에 적용됩니다. 업데이트가 적용되는 시기는 업데이트의 유형뿐 아니라 DB 클러스터에 대한 유지 관리 기간 설정 및 리전에 따라 다릅니다. 업데이트 후에는 데이터베이스를 다시 시작해야 하므로 다운타임이 20~30초간 발생할 수 있습니다. 다운타임 후에 DB 클러스터(들)를 다시 사용할 수 있습니다. 에서 유지 관리 기간 설정을 확인하거나 변경할 수 있습니다..AWS Management Console

> **Note**
>
> DB 인스턴스를 재부팅하는 시간은 충돌 복구 프로세스, 재부팅 시의 데이터베이스 활동 및 특정 DB 엔진의 동작에 따라 다릅니다. 따라서 재부팅 시간을 단축하려면 재부팅 프로세스에서 데이터베이스 작업을 최소화하는 것이 좋습니다. 데이터베이스 작업을 줄이면 중간 트랜잭션의 롤백 작업이 줄어듭니다.

Amazon Aurora 운영 체제 업데이트에 대한 자세한 내용은 ??? 단원을 참조하십시오

일부 업데이트는 Aurora에서 지원하는 데이터베이스 엔진으로 국한됩니다. 데이터베이스 엔진 업데이트에 대한 자세한 내용은 다음 표를 참조하십시오.

데이터베이스 엔진	업데이트
Amazon Aurora MySQL	Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트섹션을 참조하십시오.
Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트섹션을 참조하십시오.

Amazon Aurora 버전 식별

Amazon Aurora는 Aurora에 일반적이며 모든 Aurora DB 클러스터에서 사용할 수 있는 특정 기능을 포함합니다. Aurora는 Aurora가 지원하는 특정 데이터베이스 엔진에 특정한 다른 기능을 포함합니다. 이러한 기능들은 Aurora PostgreSQL 같이 해당 데이터베이스 엔진을 사용하는 Aurora DB 클러스터에만 제공됩니다.

Aurora DB 인스턴스는 Aurora 버전 번호와 Aurora 데이터베이스 엔진 버전 번호 등 두 가지 버전 번호를 제공합니다. Aurora 버전 번호에 사용되는 형식은 다음과 같습니다.

```
<major version>.<minor version>.<patch version>
```

특정 데이터베이스 엔진을 사용하여 Aurora DB 인스턴스에서 Aurora 버전 번호를 가져오려면 다음 쿼리 중 하나를 사용하십시오.

데이터베이스 엔진	쿼리
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

Amazon Aurora를 사용한 Amazon RDS 추가 지원

RDS 추가 지원을 사용하면 Aurora 표준 지원 종료일이 지난 후 메이저 엔진 버전에서 추가 비용을 지불하고 데이터베이스를 계속 실행할 수 있습니다.

DB 인스턴스를 처음 생성하거나 복원할 때 RDS 추가 지원을 활성화해야만 RDS 추가 지원에 데이터베이스를 등록할 수 있습니다. 복원하지 않으면 기존 DB 인스턴스에서 RDS 추가 지원 등록 상태를 업데이트할 수 없습니다.

DB 인스턴스를 생성 또는 복원하는 동안 RDS 추가 지원을 활성화했다면 Aurora의 표준 지원 종료일 이후, Amazon Aurora가 자동으로 해당 DB 인스턴스를 RDS 추가 지원에 등록합니다. RDS 추가 지원에 자동 등록해도 데이터베이스 엔진은 변경되지 않으며, DB 인스턴스의 가동 시간이나 성능에도 영향을 미치지 않습니다.

RDS 추가 지원은 다음과 같은 업데이트 및 기술 지원을 제공합니다.

- 데이터베이스 엔진을 포함하여 DB 인스턴스 또는 DB 클러스터의 중요 및 높은 CVE에 대한 보안 업데이트
- 중요한 문제에 대한 버그 수정 및 패치
- 표준 Amazon RDS 서비스 수준 계약 내에서 지원 사례를 열고 문제 해결 지원을 받을 수 있는 기능

이 유료 기능을 사용하면 지원되는 메이저 엔진 버전으로 업그레이드하는 데 더 많은 시간을 할애할 수 있습니다. 예를 들어 Aurora MySQL 버전 2에 대한 Aurora 표준 지원 종료일은 2024년 10월 31일입니다. 하지만 종료일 이전에 Aurora MySQL 버전 3으로 수동 업그레이드할 준비가 되지 않을 수 있습니다. 이 경우, 2024년 10월 31일에 Amazon Aurora는 RDS 추가 지원에 클러스터를 자동으로 등록합니다. 그러면 사용자는 Aurora MySQL 버전 2를 계속 실행할 수 있습니다. 2024년 12월 1일부터 Amazon Aurora에서 RDS 추가 지원 요금을 자동으로 청구합니다.

RDS 추가 지원은 메이저 엔진 버전의 커뮤니티 수명 종료일(Aurora MySQL 버전 2의 경우 3년 4개월) 이후 최대 3년 동안 사용할 수 있습니다. 3년이 지난 후에도 지원되는 버전으로 메이저 엔진 버전을 업그레이드하지 않으면 Amazon Aurora는 메이저 엔진 버전을 자동 업그레이드합니다. 따라서 지원되는 메이저 엔진 버전으로 최대한 빨리 업그레이드하는 것이 좋습니다.

Aurora 표준 지원 종료일 및 RDS 추가 지원 종료일에 대한 자세한 내용은 Aurora MySQL 메이저 버전 릴리스 일정 및 Aurora PostgreSQL 메이저 버전 릴리스 일정을 참조하세요.

주제

- Amazon RDS 추가 지원 개요
- Amazon RDS 추가 지원 요금
- Amazon RDS 추가 지원이 포함된 버전
- Amazon Aurora 및 Amazon RDS 추가 지원 사용 시 고객 책임
- Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 생성
- Amazon RDS 추가 지원의 Aurora DB 클러스터 또는 글로벌 클러스터 등록 보기
- Amazon RDS 추가 지원에서 엔진 버전에 대한 지원 날짜 보기
- Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 복원

Amazon RDS 추가 지원 개요

Aurora의 표준 지원 종료일 이후, DB 인스턴스를 생성 또는 복원하는 동안 RDS 추가 지원을 비활성화하지 않은 경우 Amazon Aurora가 자동으로 해당 인스턴스를 RDS 추가 지원에 등록합니다. Aurora는 Aurora 표준 지원 종료일이 되기 전 마지막 마이너 버전 릴리스(아직 이 버전을 실행하지 않는 경우)로 DB 인스턴스를 자동 업그레이드합니다. Amazon Aurora는 메이저 엔진 버전에 적용되는 Aurora 표준 지원 종료일이 지날 때까지 마이너 버전을 업그레이드하지 않습니다.

또한 Aurora 표준 지원 종료일에 도달한 메이저 엔진 버전으로 새 데이터베이스를 생성할 수 있습니다. Aurora는 이러한 새 데이터베이스를 RDS 확장 지원에 자동으로 등록하고 이 서비스에 대한 비용을 청구합니다.

Aurora의 표준 지원 종료일 이전에 아직 Aurora 표준 지원이 적용되는 엔진으로 업그레이드하는 경우 Amazon Aurora는 엔진을 RDS 추가 지원에 등록하지 않습니다.

Aurora 표준 지원 종료일이 지났지만 RDS 추가 지원에 등록되지 않은 엔진과 호환되는 데이터베이스의 스냅샷을 복원하려고 시도하는 경우 Amazon Aurora는 아직 Aurora 표준 지원이 적용되는 최신 엔진 버전과 호환되도록 스냅샷을 업그레이드하려고 시도합니다. 복원이 실패할 경우 Amazon Aurora는 스냅샷과 호환되는 버전으로 엔진을 RDS 추가 지원에 자동으로 등록합니다.

언제든지 RDS 추가 지원 등록을 종료할 수 있습니다. 등록을 종료하려면 등록된 각 엔진을 아직 Aurora 표준 지원이 적용되는 최신 엔진 버전으로 업그레이드하세요. RDS 추가 지원 등록 종료는 아직 Aurora 표준 지원이 적용되는 최신 엔진 버전으로 업그레이드를 완료하는 날부터 유효합니다.

Aurora 표준 지원 종료일 및 RDS 추가 지원 종료일에 대한 자세한 내용은 Aurora MySQL 메이저 버전 릴리스 일정 및 Aurora PostgreSQL 메이저 버전 릴리스 일정을 참조하세요.

Amazon RDS 추가 지원 요금

Aurora 표준 지원 종료일 이후부터 RDS 추가 지원에 등록된 모든 엔진에 대해 요금이 부과됩니다. Aurora 표준 지원 종료일에 대해 알아보려면 Amazon Aurora 메이저 버전 섹션을 참조하세요.

다음 작업 중 하나를 수행하는 경우 RDS 추가 지원에 대한 추가 요금이 자동으로 중지됩니다.

- 표준 지원이 적용되는 엔진 버전으로 업그레이드하세요.
- Aurora 표준 지원 종료일이 지난 메이저 버전을 실행 중인 데이터베이스를 삭제합니다.

향후 대상 엔진 버전이 RDS 추가 지원으로 전환되면 요금이 다시 부과됩니다.

예를 들어, Aurora PostgreSQL 11은 2024년 3월 1일에 확장 지원이 시작되지만, 2024년 4월 1일까지는 요금이 청구되지 않습니다. 2024년 4월 30일에 Aurora PostgreSQL 11 데이터베이스를 Aurora PostgreSQL 12로 업그레이드합니다. Aurora PostgreSQL 11에 대한 30일간의 추가 지원 비용만 청구됩니다. RDS 표준 지원 종료일인 2025년 2월 28일 이후에도 이 DB 인스턴스에서 Aurora PostgreSQL 12를 계속 실행할 수 있습니다. 2025년 3월 1일부터 데이터베이스에 RDS 추가 지원 요금이 다시 부과됩니다.

자세한 내용을 알아보려면 Amazon Aurora 요금을 참조하세요.

Amazon RDS 추가 지원에 대한 비용 방지

Aurora 표준 지원 종료일이 지난 후 Aurora가 Aurora DB 클러스터 또는 글로벌 클러스터를 생성 또는 복원하지 못하게 하여 RDS 추가 지원 요금이 부과되지 않도록 할 수 있습니다. 이렇게 하려면 AWS CLI 또는 RDS API를 사용하세요.

AWS CLI에서 `--engine-lifecycle-support` 옵션에 대해 `open-source-rds-extended-support-disabled`를 지정합니다. RDS API에서 `LifeCycleSupport` 파라미터에 `open-source-rds-extended-support-disabled`를 지정합니다. 자세한 내용은 Aurora DB 클러스터 또는 글로벌 클러스터 생성 또는 Aurora DB 클러스터 또는 글로벌 클러스터 복원 단원을 참조하세요.

Amazon RDS 추가 지원이 포함된 버전

RDS 추가 지원은 Aurora MySQL 및 Aurora PostgreSQL에서 사용할 수 있습니다. 자세한 내용은 Amazon Aurora 메이저 버전 섹션을 참조하세요.

RDS 추가 지원은 특정 마이너 버전에서만 사용할 수 있습니다. 마이너 버전은 메이저 버전이 커뮤니티 수명 종료 날짜에 도달한 후에만 RDS 추가 지원을 지원하는 것으로 표시됩니다. 자세한 내용

은 Aurora MySQL 릴리스 정보의 Aurora MySQL 릴리스 일정 및 Aurora PostgreSQL 릴리스 정보의 Aurora PostgreSQL 릴리스 일정을 참조하시기 바랍니다.

RDS 추가 지원은 Aurora Serverless v2에서만 사용할 수 있습니다. Aurora Serverless v1에서는 사용할 수 없습니다.

AWS CLI 또는 RDS API를 사용하여 엔진 버전의 지원 날짜에 대한 정보를 볼 수도 있습니다. 자세한 내용은 Amazon RDS 추가 지원에서 엔진 버전에 대한 지원 날짜 보기 섹션을 참조하세요.

Amazon Aurora 및 Amazon RDS 추가 지원 사용 시 고객 책임

다음 콘텐츠에서는 Amazon Aurora의 책임과 RDS 추가 지원 사용 시 사용자 책임에 대해 설명합니다.

주제

- Amazon Aurora 책임
- 귀하의 책임

Amazon Aurora 책임

Aurora의 표준 지원 종료일 이후, Amazon Aurora는 RDS 추가 지원에 등록된 엔진에 대한 패치, 버그 수정 및 업그레이드를 제공합니다. 이는 최대 3년 동안 또는 엔진 사용을 중지할 때까지(둘 중 먼저 발생하는 날짜) 지원됩니다.

패치를 통해 국가 취약성 데이터베이스(NVD) CVSS 심각도 등급에 정의된 대로 중요 및 상위 CVE에 대해 지원합니다. 자세한 내용을 알아보려면 Vulnerability Metrics 페이지를 참조하세요.

귀하의 책임

RDS 추가 지원에 등록된 Aurora DB 클러스터 또는 글로벌 클러스터에 제공된 패치, 버그 수정 및 업그레이드를 적용하는 것은 사용자의 책임입니다. Amazon Aurora는 언제든지 이러한 패치, 버그 수정 및 업그레이드를 변경, 교체 또는 철회할 권한을 보유합니다. 보안 또는 중요한 안정성 문제를 해결하기 위해 패치가 필요한 경우 Amazon Aurora는 패치를 사용하여 Aurora DB 클러스터 또는 글로벌 클러스터를 업데이트하거나 패치 설치를 요구할 권한을 보유합니다.

또한 RDS의 추가 지원 종료일 이전에 엔진을 최신 엔진 버전으로 업그레이드하는 것은 사용자의 책임입니다. RDS 추가 지원 종료일은 일반적으로 커뮤니티 종료일로부터 3년입니다. . 데이터베이스 메이저 엔진 버전의 RDS 추가 지원 종료일은 Amazon Aurora 메이저 버전 섹션을 참조하세요.

엔진을 업그레이드하지 않으면 RDS 추가 지원 종료일 이후에 Amazon Aurora가 Aurora 표준 지원에서 지원되는 최신 엔진 버전으로 엔진 업그레이드를 시도합니다. 업그레이드가 실패할 경우 Amazon Aurora는 Aurora 표준 지원 종료일이 지난 후 엔진을 실행 중인 Aurora DB 클러스터 또는 글로벌 클러스터를 삭제할 권한을 보유합니다. 하지만 이렇게 하기 전에 Amazon Aurora는 해당 엔진의 데이터를 보존합니다.

Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 생성

Aurora DB 클러스터 또는 글로벌 클러스터를 생성할 때는 콘솔에서 RDS 추가 지원 활성화를 선택하거나 AWS CLI의 추가 지원 옵션이나 RDS API의 파라미터를 사용하세요. Amazon RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터를 등록하면 Aurora DB 클러스터 또는 글로벌 클러스터의 수명 주기 동안 RDS 추가 지원에 영구적으로 등록됩니다.

콘솔을 사용하는 경우 RDS 추가 지원 활성화를 선택해야 합니다. 이 설정은 기본적으로 선택되어 있지 않습니다.

AWS CLI 또는 RDS API를 사용하며 RDS 추가 지원 설정을 지정하지 않는 경우 Amazon RDS는 기본적으로 RDS 추가 지원을 활성화합니다. AWS CloudFormation 또는 기타 서비스를 사용하여 자동화하면 이 기본 동작은 Aurora 표준 지원 종료일 이후에도 데이터베이스의 가용성을 유지합니다.

AWS CLI 또는 RDS API를 사용하여 Aurora DB 클러스터 또는 글로벌 클러스터를 생성하면 RDS 추가 지원에 등록되지 않도록 할 수 있습니다.

주제

RDS 추가 지원 동작

다음 표에는 메이저 엔진 버전이 Aurora 표준 지원 종료 시점에 도달했을 때 발생하는 상황이 요약되어 있습니다.

RDS 추가 지원 상태*	동작
활성화됨	Amazon RDS가 RDS 추가 지원에 대한 요금 청구

RDS 추가 지원 상태*	동작
비활성	Amazon RDS는 Aurora DB 클러스터 또는 글로벌 클러스터를 지원되는 엔진 버전으로 업그레이드합니다. 이 업그레이드는 Aurora 표준 지원 종료일 당일 또는 그 직후에 진행됩니다.

* RDS 콘솔에서 RDS 추가 지원 상태는 '예' 또는 '아니요'로 표시됩니다. AWS CLI 또는 RDS API에서 RDS 추가 지원 상태는 `open-source-rds-extended-support` 또는 `open-source-rds-extended-support-disabled`로 표시됩니다.

RDS 추가 지원 고려 사항

Aurora DB 클러스터 또는 글로벌 클러스터를 만들기 전에 다음 항목을 고려하세요.

- Aurora 표준 지원 종료일이 지난 후에 새 Aurora DB 클러스터 또는 새 글로벌 클러스터가 생성되지 않게 하고 RDS 확장 지원 요금을 피할 수 있습니다. 이렇게 하려면 AWS CLI 또는 RDS API를 사용하세요. AWS CLI에서 `--engine-lifecycle-support` 옵션에 대해 `open-source-rds-extended-support-disabled`를 지정합니다. RDS API에서 `LifeCycleSupport` 파라미터에 `open-source-rds-extended-support-disabled`를 지정합니다. `open-source-rds-extended-support-disabled`를 지정하고 Aurora 표준 지원 종료일이 지난 경우, Aurora DB 클러스터 또는 글로벌 클러스터 생성이 항상 실패합니다.
- RDS 추가 지원은 클러스터 수준에서 설정됩니다. 클러스터 멤버는 RDS 콘솔, AWS CLI의 `--engine-lifecycle-support`, RDS API의 `EngineLifecycleSupport`에서 항상 동일한 RDS 추가 지원 설정을 가집니다.

자세한 내용은 Amazon Aurora 단원을 참조하십시오.

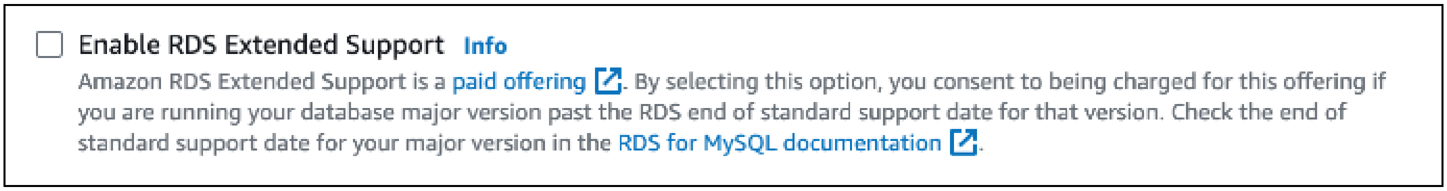
RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 RDS 추가 지원 버전이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터를 만들 수 있습니다.

콘솔

Aurora DB 클러스터 또는 글로벌 클러스터를 생성할 때는 엔진 옵션 섹션에서 RDS 확장 지원 활성화를 선택합니다. 이 설정은 기본적으로 선택되어 있지 않습니다.

다음 이미지는 RDS 추가 지원 활성화 설정을 보여줍니다.



AWS CLI

create-db-cluster 또는 create-global-cluster AWS CLI 명령을 실행하는 경우 `--engine-lifecycle-support` 옵션에 대해 `open-source-rds-extended-support`를 지정하여 RDS 추가 지원을 선택합니다. 기본적으로 이 옵션은 `open-source-rds-extended-support`(으)로 설정되어 있습니다.

Aurora 표준 지원 종료일 이후에 새 Aurora DB 클러스터 또는 글로벌 클러스터가 생성되지 않도록 하려면 `--engine-lifecycle-support` 옵션에 `open-source-rds-extended-support-disabled`를 지정하세요. 이렇게 하면 RDS 확장 지원 관련 요금을 피할 수 있습니다.

RDS API

CreateDBCluster 또는 CreateGlobalCluster Amazon RDS API 작업을 사용할 때는 `EngineLifecycleSupport` 파라미터를 `open-source-rds-extended-support`로 설정하여 RDS 추가 지원을 선택합니다. 이 파라미터는 기본적으로 `open-source-rds-extended-support`로 설정되어 있습니다.

Aurora 표준 지원 종료일 이후에 새 Aurora DB 클러스터 또는 글로벌 클러스터가 생성되지 않도록 하려면 `EngineLifecycleSupport` 파라미터에 `open-source-rds-extended-support-disabled`를 지정하세요. 이렇게 하면 RDS 확장 지원 관련 요금을 피할 수 있습니다.

자세한 정보는 다음의 주제를 참조하세요.

- Aurora DB 클러스터를 만들려면 Amazon Aurora DB 클러스터 생성 섹션의 DB 엔진에 대한 지침을 따르세요.
- 글로벌 클러스터를 만들려면 Amazon Aurora 글로벌 데이터베이스 생성 섹션의 DB 엔진에 대한 지침을 따르세요.

Amazon RDS 추가 지원의 Aurora DB 클러스터 또는 글로벌 클러스터 등록 보기

AWS Management Console(AWS CLI) 또는 RDS API를 사용하여 RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터 등록을 확인할 수 있습니다.

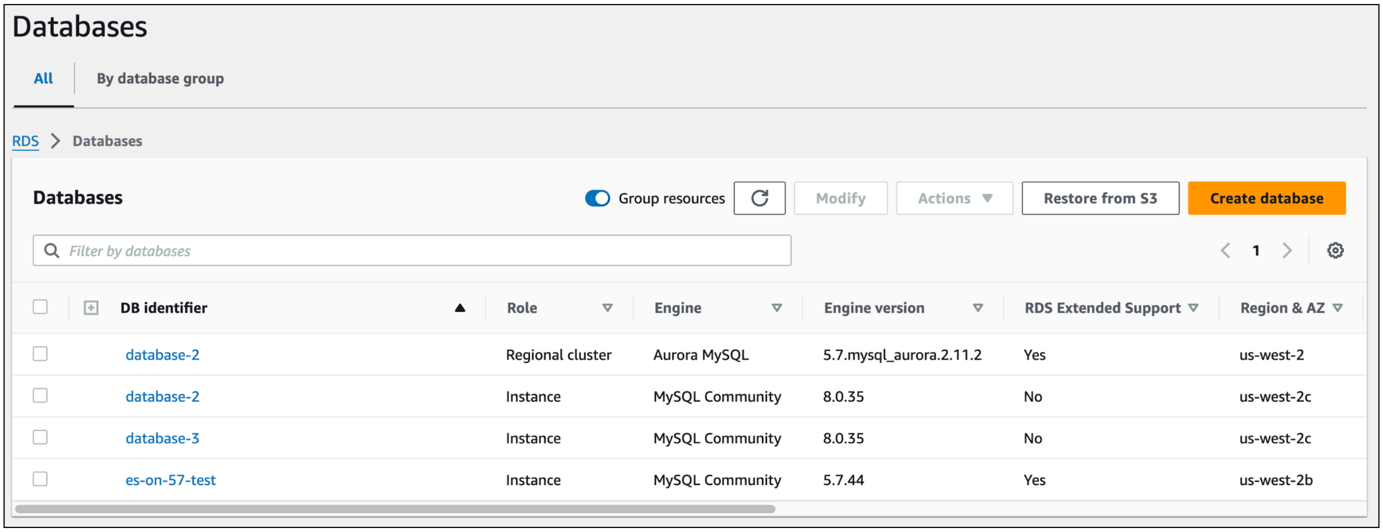
> **Note**
>
> 콘솔의 RDS 추가 지원 열, AWS CLI의 `--engine-lifecycle-support` 옵션 및 RDS API의 `EngineLifecycleSupport` 파라미터는 RDS 추가 지원 등록만 나타냅니다. RDS 추가 지원 요금은 DB 엔진 버전이 Aurora 표준 지원 종료 시점에 도달한 경우에만 시작됩니다. 자세한 내용은 [Amazon Aurora 메이저 버전](https://docs.aws.amazon.com/) 단원을 참조하십시오.
>
> 예를 들어, RDS 추가 지원에 등록된 Aurora PostgreSQL 11 데이터베이스가 있다고 가정해 보겠습니다. 2024년 4월 1일부터 Amazon RDS는 이 데이터베이스에 대해 RDS 추가 지원 요금을 청구하기 시작했습니다. 2024년 7월 31일에 이 데이터베이스를 Aurora PostgreSQL 12로 업그레이드했습니다. 이 데이터베이스의 RDS 추가 지원 상태는 활성화된 상태로 유지됩니다. 하지만 Aurora PostgreSQL 12이 아직 Aurora 표준 지원 종료 시점에 도달하지 않았기 때문에 이 데이터베이스에 대한 RDS 추가 지원 요금 청구는 중단되었습니다. Amazon RDS는 2025년 3월 1일까지 이 데이터베이스에 대한 RDS 추가 지원 요금을 청구하지 않습니다. 이 날짜가 되면 Aurora PostgreSQL 12에 대한 Aurora 표준 지원이 종료됩니다.

콘솔

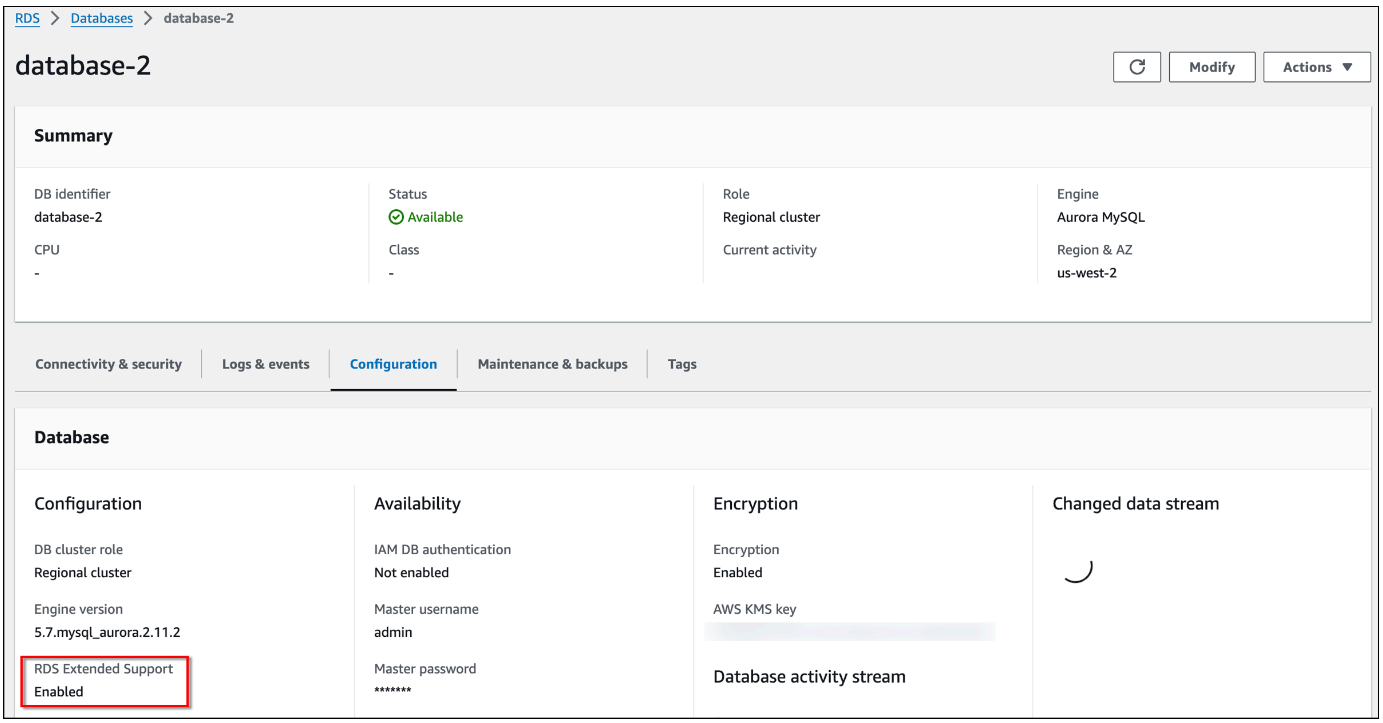
RDS 추가 지원의 Aurora DB 클러스터 또는 글로벌 클러스터 등록을 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다. RDS 추가 지원의 값은 Aurora DB 클러스터 또는 글로벌 클러스터가 RDS 추가 지원에 등록되었는지 여부를 나타냅니다. 값이 표시되지 않으면 데이터베이스에 대해 RDS 확장 지원을 사용할 수 없는 것입니다.

   > **Tip**
   >
   > RDS 추가 지원 열이 표시되지 않는 경우 기본 설정 아이콘을 선택한 다음 RDS 추가 지원을 활성화합니다.



3. 각 데이터베이스의 구성 탭에서 등록을 볼 수도 있습니다. DB 식별자에서 데이터베이스를 선택합니다. 구성 탭의 추가 지원에서 데이터베이스가 등록되었는지 여부를 확인합니다.



AWS CLI

AWS CLI를 사용하여 RDS 추가 지원에서 데이터베이스의 등록을 보려면 describe-db-clusters 또는 describe-global-clusters 명령을 실행하세요.

데이터베이스에 대해 RDS 추가 지원을 사용할 수 있는 경우, 응답에 `EngineLifecycleSupport` 파라미터가 포함됩니다. `open-source-rds-extended-support` 값은 Aurora DB 클러스터 또는 글로벌 클러스터가 RDS 추가 지원에 등록되어 있음을 나타냅니다. `open-source-rds-extended-support-disabled` 값은 RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터의 등록이 비활성화되었음을 나타냅니다.

예

다음 명령은 모든 Aurora DB 클러스터에 대한 정보를 반환합니다.

```
aws rds describe-db-clusters
```

다음 응답은 Aurora DB 클러스터 `database-1`에서 실행 중인 Aurora PostgreSQL 엔진이 RDS 추가 지원에 등록되어 있음을 나타냅니다.

```
{
    "DBClusterIdentifier": "database-1",
    ...
    "Engine": "aurora-postgresql",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

RDS API

Amazon RDS API를 사용하여 RDS 추가 지원에서 데이터베이스의 등록을 보려면 DescribeDBClusters 또는 DescribeGlobalClusters 작업을 사용하세요.

데이터베이스에 대해 RDS 추가 지원을 사용할 수 있는 경우, 응답에 `EngineLifecycleSupport` 파라미터가 포함됩니다. `open-source-rds-extended-support` 값은 Aurora DB 클러스터 또는 글로벌 클러스터가 RDS 추가 지원에 등록되어 있음을 나타냅니다. `open-source-rds-extended-support-disabled` 값은 RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터의 등록이 비활성화되었음을 나타냅니다.

Amazon RDS 추가 지원에서 엔진 버전에 대한 지원 날짜 보기

AWS CLI 또는 RDS API를 사용하여 Amazon RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터의 엔진 버전에 대한 지원 날짜 정보를 볼 수 있습니다. 이 정보는 업그레이드를 계획하는 데 도움이 될 수 있습니다.

AWS CLI 명령 및 RDS API 작업은 Aurora 표준 지원 및 RDS 추가 지원에 대한 시작 및 종료 날짜를 반환합니다. 이러한 날짜는 메이저 엔진 버전 테이블에서도 확인할 수 있습니다. 자세한 내용은 [Amazon Aurora 메이저 버전](#) 섹션을 참조하세요.

AWS CLI

주요 엔진 버전에 대한 Aurora 표준 지원 및 RDS 추가 지원의 시작 날짜와 종료 날짜를 확인하려면 AWS CLI를 사용하여 [describe-db-major-engine-versions](#) 명령을 실행합니다.

이 명령은 다음과 같은 관련 파라미터를 반환합니다.

- `SupportedEngineLifecycles` - 이 파라미터는 `LifecycleSupportName`, `LifecycleSupportStartDate` 및 `LifecycleSupportEndDate`를 포함하는 객체의 배열입니다.
- `LifecycleSupportName` - 이 파라미터는 엔진 버전이 속한 지원 유형(Aurora 표준 지원(`open-source-rds-standard-support`) 또는 RDS 추가 지원(`open-source-rds-extended-support`))을 나타냅니다.
- `LifecycleSupportStartDate` - 이 파라미터는 `LifecycleSupportName`의 값에 따라 메이저 엔진 버전의 Aurora 표준 지원 또는 RDS 추가 지원에 대한 시작 날짜를 나열합니다.
- `LifecycleSupportEndDate` - 이 파라미터는 `LifecycleSupportName`의 값에 따라 메이저 엔진 버전의 Aurora 표준 지원 또는 RDS 추가 지원에 대한 종료 날짜를 나열합니다.

예제

응답 예제는 Aurora MySQL 버전 2(MySQL 5.7)의 지원되는 엔진 수명 주기 `open-source-rds-standard-support` 및 `open-source-rds-extended-support`에 대한 시작 및 종료 날짜를 보여 줍니다. Aurora MySQL 버전 2(MySQL 5.7)에서는 RDS 추가 지원을 사용할 수 있습니다.

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "aurora-mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2018-02-06T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-10-31T23:59:59.999000+00:00"
                },
                {
```

```
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-11-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
                }
            ]
        },
        ...
    ]
}
```

RDS API

RDS API를 사용하여 메이저 엔진 버전의 Aurora 표준 지원 및 RDS 추가 지원에 대한 시작 및 종료 날짜를 보려면 DescribeDBMajorEngineVersions 작업을 사용합니다.

엔진 버전에 대해 RDS 추가 지원을 사용할 수 있는 경우 응답에는 두 개의 객체가 있는 배열로 `SupportedEngineLifeCycles` 파라미터가 포함됩니다. 하나의 객체에는 Aurora 표준 지원을 위한 시작 및 종료 날짜가 포함됩니다. 두 번째 객체에는 RDS 추가 지원의 시작 및 종료 날짜가 포함됩니다.

엔진 버전에서 RDS 추가 지원을 사용할 수 없는 경우 응답에는 단일 객체가 있는 배열로 `SupportedEngineLifeCycles` 파라미터만 포함됩니다. 이 객체에는 Aurora 표준 지원의 시작 및 종료 날짜가 포함됩니다.

Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 복원

Aurora DB 클러스터 또는 글로벌 클러스터를 복원할 때는 콘솔에서 RDS 추가 지원 활성화를 선택하거나 AWS CLI의 추가 지원 옵션이나 RDS API의 파라미터를 사용하세요. RDS 추가 지원에서 Aurora DB 클러스터 또는 글로벌 클러스터를 등록하면 Aurora DB 클러스터 또는 글로벌 클러스터의 수명 주기 동안 RDS 추가 지원에 영구적으로 등록됩니다.

RDS 추가 지원 설정의 기본값은 데이터베이스를 복원할 때 콘솔, AWS CLI, RDS API 중 무엇을 사용하는지에 따라 달라집니다. 콘솔을 사용할 때 RDS 추가 지원 활성화를 선택하지 않고 복원 중인 메이저 엔진 버전이 Aurora 표준 지원 종료 시점이 지난 경우 Amazon Aurora는 자동으로 DB 인스턴스를 최신 엔진 버전으로 업그레이드합니다. AWS CLI 또는 RDS API를 사용하며 RDS 추가 지원 설정을 지정하지 않는 경우 Amazon RDS는 기본적으로 RDS 추가 지원을 활성화합니다. AWS CloudFormation 또는 기타 서비스를 사용하여 자동화하면 이 기본 동작은 Aurora 표준 지원 종료일 이후에도 데이터베이스의 가용성을 유지합니다. AWS CLI 또는 RDS API를 사용하여 RDS 추가 지원을 비활성화할 수 있습니다.

주제

- RDS 추가 지원 동작
- RDS 추가 지원 고려 사항
- RDS 확장 지원이 적용되는 Aurora DB 클러스터 또는 글로벌 클러스터 복원

RDS 추가 지원 동작

다음 표에는 복원 중인 Aurora DB 클러스터 또는 글로벌 클러스터의 메이저 엔진 버전이 Aurora 표준 지원 종료 시점에 도달했을 때 발생하는 상황이 요약되어 있습니다.

RDS 추가 지원 상태*	동작
활성화됨	Amazon RDS가 RDS 추가 지원에 대한 요금 청구
비활성화됨	복원이 완료되면 Amazon RDS는 향후 유지 관리 기간에 Aurora DB 클러스터 또는 글로벌 클러스터를 최신 엔진 버전으로 자동 업그레이드합니다.

* RDS 콘솔에서 RDS 추가 지원 상태는 '예' 또는 '아니요'로 표시됩니다. AWS CLI 또는 RDS API에서 RDS 추가 지원 상태는 `open-source-rds-extended-support` 또는 `open-source-rds-extended-support-disabled`로 표시됩니다.

RDS 추가 지원 고려 사항

Aurora DB 클러스터 또는 글로벌 클러스터를 복원하기 전에 다음 항목을 고려하세요.

- Aurora 표준 지원 종료일이 지난 후 Amazon S3에서 Aurora DB 클러스터 또는 글로벌 클러스터를 복원하려는 경우 AWS CLI 또는 RDS API를 사용해서만 복원할 수 있습니다. restore-db-cluster-from-s3 AWS CLI 명령의 `--engine-lifecycle-support` 옵션을 사용하거나 RestoreDBClusterFromS3 RDS API 작업의 `EngineLifecycleSupport` 파라미터를 사용하세요.
- Aurora가 데이터베이스를 RDS 확장 지원 버전으로 복원하지 못하게 하려면 AWS CLI 또는 RDS API에서 `open-source-rds-extended-support-disabled`를 지정하세요. 이렇게 하면 RDS 확장 지원 관련 요금을 피할 수 있습니다.

  이 설정을 지정하면 Amazon Aurora가 복원된 데이터베이스를 지원되는 최신 메이저 버전으로 자동 업그레이드합니다. 업그레이드가 업그레이드 검사에 실패하는 경우 Amazon Aurora는 RDS 확장 지

원 엔진 버전으로 안전하게 롤백합니다. 이 데이터베이스는 RDS 확장 지원 모드로 유지되며, 데이터베이스를 수동으로 업그레이드하기 전까지 Amazon Aurora에서 RDS 확장 지원 요금을 청구합니다.

- RDS 추가 지원은 클러스터 수준에서 설정됩니다. 클러스터 멤버는 RDS 콘솔, AWS CLI의 `--engine-lifecycle-support`, RDS API의 `EngineLifecycleSupport`에서 항상 동일한 RDS 추가 지원 설정을 가집니다.

자세한 내용은 Amazon Aurora 단원을 참조하십시오.

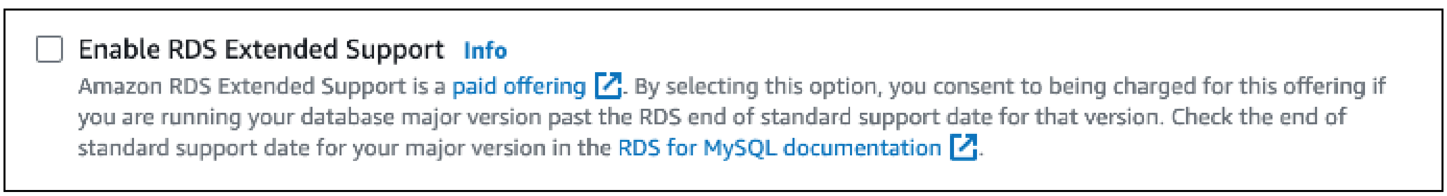
RDS 확장 지원이 적용되는 Aurora DB 클러스터 또는 글로벌 클러스터 복원

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 RDS 추가 지원 버전이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터를 복원할 수 있습니다.

콘솔

Aurora DB 클러스터 또는 글로벌 클러스터를 복원할 때는 엔진 옵션 섹션에서 RDS 추가 지원 활성화를 선택합니다. 이 설정을 선택하지 않았으며 복원 중인 메이저 엔진 버전이 Aurora 표준 지원 종료 시점이 지난 경우 Amazon Aurora는 Aurora DB 클러스터 또는 글로벌 클러스터를 Aurora 표준 지원 버전으로 자동 업그레이드합니다.

다음 이미지는 RDS 추가 지원 활성화 설정을 보여줍니다.



AWS CLI

restore-db-cluster-from-snapshot AWS CLI 명령을 실행하는 경우 `--engine-lifecycle-support` 옵션에 대해 `open-source-rds-extended-support`를 지정하여 RDS 추가 지원을 선택합니다.

RDS 확장 지원과 관련된 비용을 피하려면 `--engine-lifecycle-support` 옵션을 `open-source-rds-extended-support-disabled`로 설정하세요. 기본적으로 이 옵션은 `open-source-rds-extended-support`(으)로 설정되어 있습니다.

다음 AWS CLI 명령을 사용하여 이 값을 지정할 수도 있습니다.

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

RDS API

RestoreDBClusterFromSnapshot Amazon RDS API 작업을 사용하는 경우 `EngineLifecycleSupport` 파라미터를 `open-source-rds-extended-support`로 설정하여 RDS 추가 지원을 선택합니다.

RDS 확장 지원과 관련된 비용을 피하려면 `EngineLifecycleSupport` 파라미터를 `open-source-rds-extended-support-disabled`로 설정하세요. 이 파라미터는 기본적으로 `open-source-rds-extended-support`로 설정되어 있습니다.

다음 RDS API 작업으로 이 값을 지정할 수도 있습니다.

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime

Aurora DB 클러스터 복원에 대한 자세한 내용은 Amazon Aurora DB 클러스터 백업 및 복구 섹션의 DB 엔진에 대한 지침을 따르세요.

데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용

블루/그린 배포는 프로덕션 데이터베이스 환경을 별도의 동기화된 스테이징 환경에 복사합니다. Amazon Aurora 블루/그린 배포를 사용하면 프로덕션 환경에 영향을 주지 않고 스테이징 환경에서 데이터베이스를 변경할 수 있습니다. 예를 들어 메이저 또는 마이너 DB 엔진 버전을 업그레이드하거나, 스테이징 환경 내 스키마를 변경할 수 있습니다. 준비가 되면 대부분의 경우 1분 미만의 가동 중지 시간으로 스테이징 환경을 새로운 프로덕션 데이터베이스 환경으로 승격할 수 있습니다.

Amazon Aurora는 프로덕션 환경에서 기본 Aurora 스토리지 볼륨을 복제하여 스테이징 환경을 만듭니다. 스테이징 환경의 클러스터 볼륨은 해당 환경에 대한 증분 변경만 저장합니다.

> **Note**
>
> 현재 블루/그린 배포는 Aurora MySQL 및 Aurora PostgreSQL Amazon RDS 엔진 가용성에 대해서는 Amazon RDS 사용 설명서의 데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용을 참조하세요.

주제

Aurora용 Amazon RDS 블루/그린 배포 개요

Amazon RDS 블루/그린 배포를 사용하면 프로덕션 환경에서 구현하기 전에 데이터베이스를 변경하고 테스트할 수 있습니다. 블루/그린 배포는 프로덕션 환경을 복사하는 스테이징 환경을 만듭니다. 블루/그린 배포에서는 블루 환경이 현재 프로덕션 환경입니다. 그린 환경은 스테이징 환경입니다. 스테이징 환경은 논리적 복제를 사용하여 현재 프로덕션 환경과 계속 동기화됩니다.

프로덕션 워크로드에 영향을 주지 않고 그린 환경에서 Aurora DB 클러스터를 변경할 수 있습니다. 예를 들어 메이저 또는 마이너 DB 엔진 버전을 업그레이드하거나 스테이징 환경에서 데이터베이스 파라미터를 변경할 수 있습니다. 그린 환경의 변경 사항을 철저하게 테스트할 수 있습니다. 준비가 되면 환경을 전환하여 그린 환경을 새로운 프로덕션 환경으로 승격할 수 있습니다. 전환은 일반적으로 1분도 걸리지 않으며 데이터 손실이 발생하지 않고 애플리케이션을 변경할 필요도 없습니다.

그린 환경은 프로덕션 환경 토폴로지의 복사본이므로, DB 클러스터와 DB 클러스터의 모든 DB 인스턴스가 배포 시 복사됩니다. 그린 환경에는 DB 클러스터 스냅샷, 성능 개선 도우미와 확장 모니터링 같은 DB 클러스터에서 사용하는 기능도 포함됩니다Aurora Serverless v2.

> **Note**
>
> - 블루/그린 배포는 Aurora MySQL 및 Aurora PostgreSQL에서 지원됩니다. Amazon RDS 가용성에 대해서는 Amazon RDS 사용 설명서의 [데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용]을 참조하세요.
> - 블루/그린 배포는 다음 버전의 Babelfish for Aurora PostgreSQL에서만 지원됩니다.
>   - 15.7 이상 버전
>   - 16.3 이상 버전

주제

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전 리전에 따라 다릅니다. 자세한 내용은 the section called "블루/그린 배포" 단원을 참조하십시오.

Amazon RDS 블루/그린 배포 사용의 장점

Amazon RDS 블루/그린 배포를 사용하면 보안 패치를 최신 상태로 유지하고, 데이터베이스 성능을 개선할 수 있으며 짧고 예측 가능한 다운타임으로 최신 데이터베이스 기능을 채택할 수 있습니다. 블루/그린 배포를 통해 메이저 또는 마이너 엔진 버전 업그레이드와 같이 데이터베이스 업데이트로 인해 발생가능한 리스크 및 다운타임을 줄일 수 있습니다.

블루/그린 배포는 다음과 같은 장점을 제공합니다.

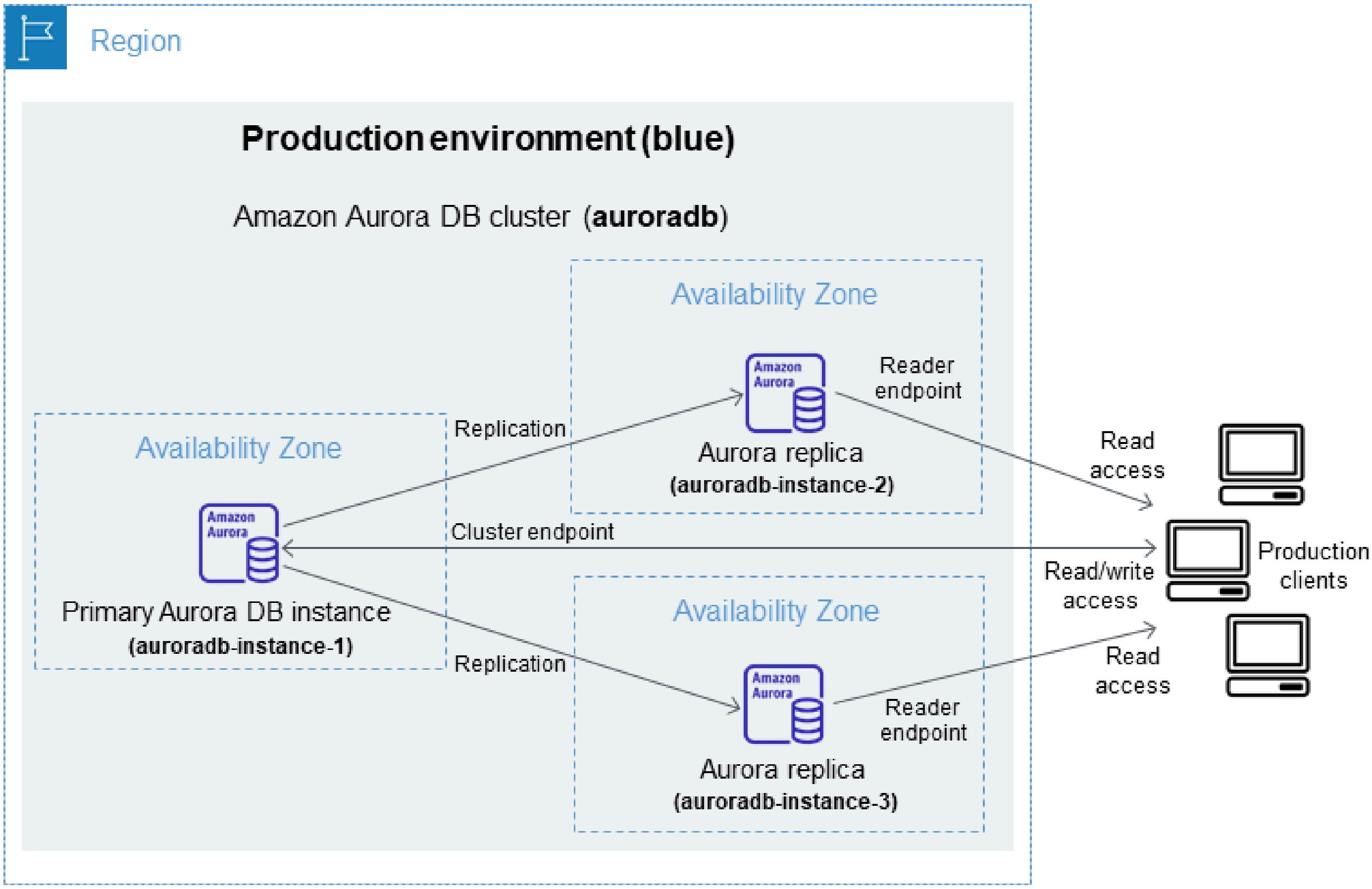
- 프로덕션 준비가 된 스테이징 환경을 쉽게 만들 수 있습니다.
- 데이터베이스 변경 사항을 프로덕션 환경에서 스테이징 환경으로 자동으로 복제합니다.
- 프로덕션 환경에 영향을 주지 않고 안전한 스테이징 환경에서 데이터베이스 변경 사항을 테스트합니다.
- 데이터베이스 패치 및 시스템 업데이트로 최신 상태를 유지하세요.
- 새로운 데이터베이스 기능을 구현하고 테스트합니다.
- 애플리케이션을 변경하지 않고도 스테이징 환경을 새 프로덕션 환경으로 전환합니다.
- 내장된 전환 가드레일을 사용하여 안전하게 전환합니다.
- 전환 중 데이터 손실이 발생하지 않습니다.
- 워크로드에 따라 대부분 1분 이내에 빠르게 전환합니다.

블루/그린 배포 워크플로우

Aurora DB 클러스터 업데이트에 블루/그린 배포를 사용한다면 다음 주요 단계를 완료하십시오.

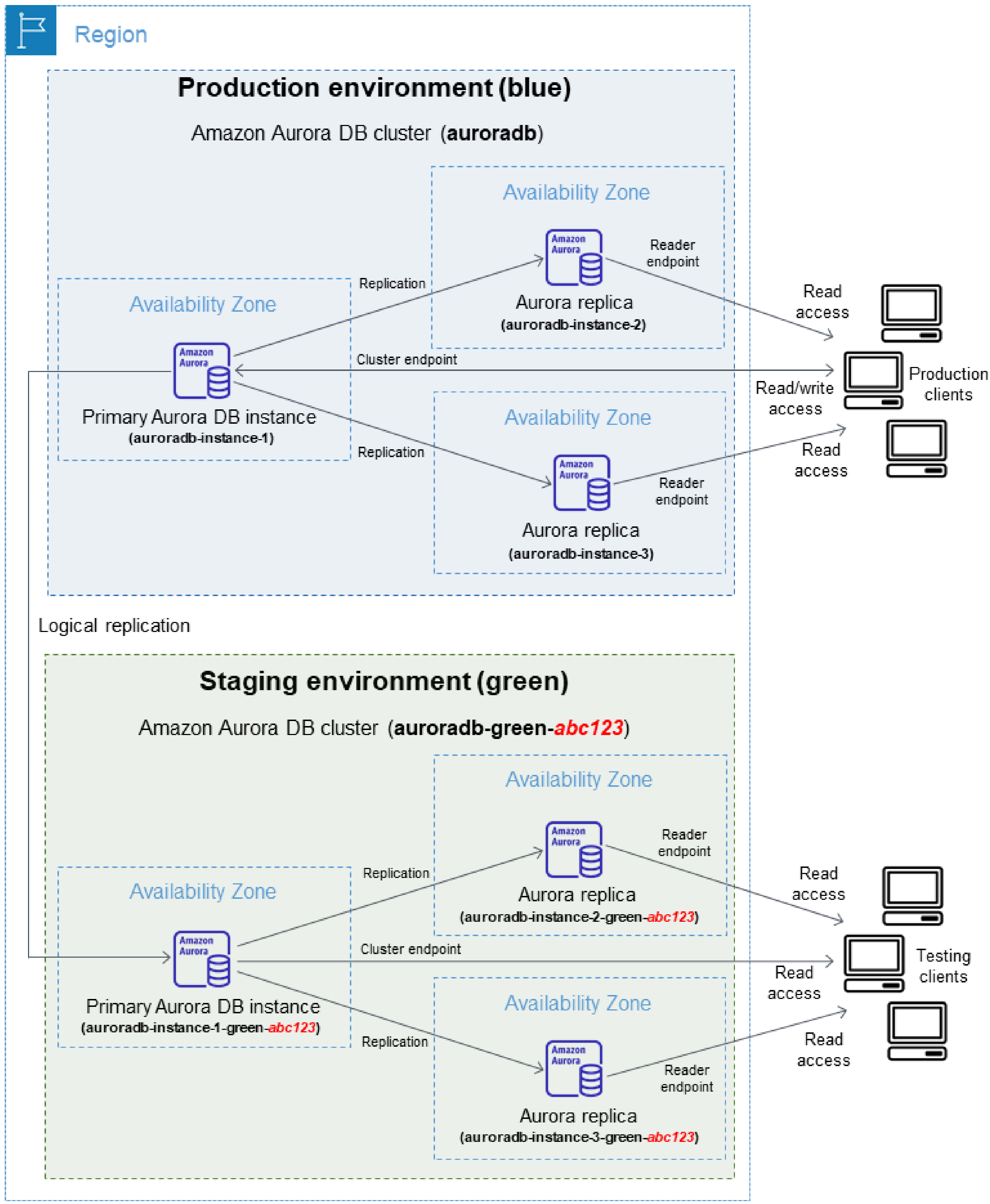
1. 업데이트가 필요한 프로덕션 DB 클러스터를 식별합니다.

   다음 이미지에서는 프로덕션 DB 클러스터 예시를 확인할 수 있습니다.



2. 블루/그린 배포 생성 지침은 [Amazon Aurora에서 블루/그린 배포 생성](#) 단원을 참조하십시오.

   다음 이미지는 1단계에서 설명하는 프로덕션 환경의 블루/그린 배포 예시입니다. 블루/그린 배포를 생성하는 동안, RDS는 Aurora DB 클러스터의 전체 토폴로지 및 구성을 복사하여 그린 환경을 만듭니다. 복사된 DB 클러스터 및 DB 인스턴스 이름 뒤에는 `-green-`*`random-characters`*가 추가됩니다. 이미지의 스테이징 환경에는 DB 클러스터(auroradb-green-*abc123*)가 있습니다. 또한 DB 클러스터에서는 DB 인스턴스 3개가 있습니다(auroradb-instance1-green-*abc123*, auroradb-instance2-green-*abc123* 및 auroradb-instance3-green-*abc123*).


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Reader endpoint
Aurora replica
(auroradb-instance-2)
Replication
Availability Zone
Read access
Cluster endpoint
Primary Aurora DB instance
(auroradb-instance-1)
Read/write access
Production clients
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Reader endpoint
Replication
Aurora replica
(auroradb-instance-2-green-abc123)
Availability Zone
Read access
Cluster endpoint
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Read access
Testing clients
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3-green-abc123)

블루/그린 배포를 생성하는 도중 더 높은 DB 엔진 버전을 지정하고, 그린 환경에 있는 DB 클러스터에 다른 DB 클러스터 파라미터 그룹을 지정할 수 있습니다. DB 클러스터의 DB 인스턴스에 다른 DB 파라미터 그룹을 지정할 수 있습니다.

또한 RDS는 블루 환경의 기본 DB 인스턴스에서 그린 환경의 기본 DB 인스턴스로의 복제를 구성합니다.

> **Important**
>
> Aurora MySQL 버전 3의 경우 블루/그린 배포를 생성하면 그린 환경의 DB 클러스터는 기본적으로 쓰기 작업을 허용하지 않습니다. 그러나 이는 Aurora 마스터 사용자를 비롯하여 `CONNECTION_ADMIN` 권한이 있는 사용자에게 적용되지 않습니다. 이 권한이 있는 사용자는 `read_only` 동작을 재정의할 수 있습니다. 자세한 내용은 [역할 기반 권한 모델](#) 단원을 참조하십시오.

3. 스테이징 환경에 변경 사항을 적용합니다.

   예를 들어 데이터베이스의 스키마를 변경하거나 그린 환경에 있는 하나 이상의 DB 인스턴스에서 사용하는 DB 인스턴스 클래스를 변경할 수 있습니다.

   DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정](#) 섹션을 참조하세요.

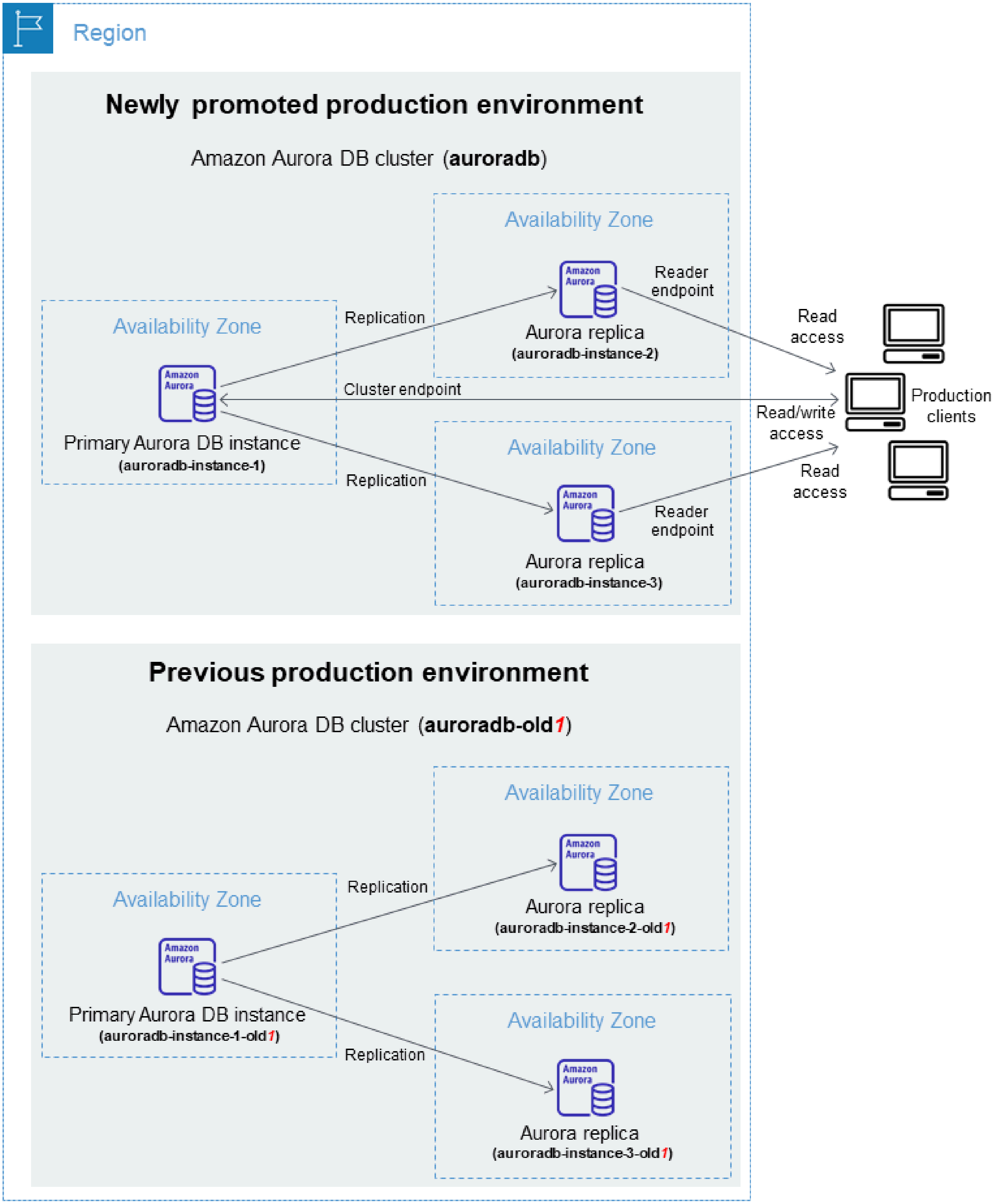
4. 스테이징 환경을 테스트합니다.

   테스트하는 동안에는 그린 환경의 데이터베이스를 읽기 전용으로 유지하는 것이 좋습니다. 복제 충돌이 발생할 수 있으므로 그린 환경에서 쓰기 작업을 사용 설정하세요. 전환 후 프로덕션 데이터베이스에 의도하지 않은 데이터가 저장될 수도 있습니다. Aurora MySQL에서 쓰기 작업을 활성화하려면 `read_only` 파라미터를 `0`으로 설정한 다음 DB 인스턴스를 재부팅합니다. Aurora PostgreSQL의 경우 `default_transaction_read_only` 파라미터를 세션 수준에서 `off`로 설정합니다.

5. 준비가 되면 전환하여 그린 환경을 새로운 프로덕션 환경으로 승격합니다. 지침은 [Amazon Aurora에서 블루/그린 배포 전환](#) 단원을 참조하십시오.

   전환하면 다운타임이 발생하게 됩니다. 다운타임은 보통 1분 미만이지만 워크로드에 따라 더 길어질 수 있습니다.

   다음 이미지는 전환 이후의 DB 클러스터를 보여줍니다.


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Reader endpoint
Aurora replica
(auroradb-instance-2)
Availability Zone
Replication
Primary Aurora DB instance
(auroradb-instance-1)
Cluster endpoint
Replication
Availability Zone
Reader endpoint
Aurora replica
(auroradb-instance-3)
Read access
Read/write access
Read access
Production clients
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Aurora replica
(auroradb-instance-2-old1)
Availability Zone
Replication
Primary Aurora DB instance
(auroradb-instance-1-old1)
Replication
Availability Zone
Aurora replica
(auroradb-instance-3-old1)

전환 후에는 그린 환경의 Aurora DB 클러스터가 새로운 프로덕션 DB 클러스터가 됩니다. 현재 프로덕션 환경의 이름과 엔드포인트가 새로 승격된 프로덕션 환경에 할당되므로, 애플리케이션을 변경할 필요가 없습니다. 따라서 이제 프로덕션 트래픽이 새 프로덕션 환경으로 이동합니다. 이전 블루 환경의 DB 클러스터와 DB 인스턴스는 현재 이름에 `-old`*n*이 추가됩니다(여기서 *n*은 숫자입니다). 예를 들어 블루 환경의 DB 인스턴스 이름이 `auroradb-instance-1`이라고 가정하겠습니다. 전환이 끝나면 DB 인스턴스 이름은 `auroradb-instance-1-old1`이 됩니다.

이미지의 예시에서는 전환 중에 다음과 같은 변경 사항이 발생합니다.

- 그린 환경 DB 클러스터 `auroradb-green-abc123`가 `auroradb`라는 프로덕션 DB 클러스터가 됩니다.
- 이름이 `auroradb-instance1-green-abc123`인 그린 환경 DB 인스턴스가 이름이 `auroradb-instance1`인 프로덕션 DB 클러스터가 됩니다.
- 이름이 `auroradb-instance2-green-abc123`인 그린 환경 DB 인스턴스가 이름이 `auroradb-instance2`인 프로덕션 DB 클러스터가 됩니다.
- 이름이 `auroradb-instance3-green-abc123`인 그린 환경 DB 인스턴스가 이름이 `auroradb-instance3`인 프로덕션 DB 클러스터가 됩니다.
- 이름이 `auroradb`인 블루 환경 DB 클러스터가 `auroradb-old1`이 됩니다.
- 이름이 `auroradb-instance1`인 블루 환경 DB 인스턴스가 `auroradb-instance1-old1`이 됩니다.
- 이름이 `auroradb-instance2`인 블루 환경 DB 인스턴스가 `auroradb-instance2-old1`이 됩니다.
- 이름이 `auroradb-instance3`인 블루 환경 DB 인스턴스가 `auroradb-instance3-old1`이 됩니다.

6. 블루/그린 배포가 더 이상 필요하지 않다면 삭제해도 됩니다. 지침은 Amazon Aurora에서 블루/그린 배포 삭제 단원을 참조하십시오.

   전환 후에도 이전 프로덕션 환경이 삭제되지 않으므로, 필요하다면 회귀 테스트에 사용할 수 있습니다.

Amazon Aurora 블루/그린 배포 작업에 대한 액세스 권한 부여

사용자는 블루/그린 배포와 관련된 작업을 수행하는 데 필요한 권한이 있어야 합니다. 지정된 리소스에서 특정 API 태스크를 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성할 수 있습

니다. 그런 다음 해당 권한이 필요한 IAM 권한 세트 또는 역할에 이러한 정책을 연결할 수 있습니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 단원을 참조하십시오.

블루/그린 배포를 만드는 사용자는 다음 RDS 작업을 수행할 권한이 있어야 합니다.

- `rds:CreateBlueGreenDeployment`
- `rds:AddTagsToResource`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:CreateDBClusterEndpoint`

블루/그린 배포로 전환하는 사용자는 다음 RDS 작업을 수행할 권한이 있어야 합니다.

- `rds:SwitchoverBlueGreenDeployment`
- `rds:ModifyDBCluster`
- `rds:PromoteReadReplicaDBCluster`

블루/그린 배포를 삭제하는 사용자는 다음 RDS 작업을 수행할 권한이 있어야 합니다.

- `rds:DeleteBlueGreenDeployment`
- `rds:DeleteDBCluster`
- `rds:DeleteDBInstance`
- `rds:DeleteDBClusterEndpoint`

Aurora는 사용자를 대신하여 스테이징 환경에서 리소스를 프로비저닝하고 수정합니다. 이러한 리소스에는 내부적으로 정의된 명명 규칙을 사용하는 DB 인스턴스가 포함됩니다. 따라서 연결된 IAM 정책에는 `my-db-prefix-*`와 같은 부분적인 리소스 이름 패턴이 포함될 수 없습니다. 와일드카드(*)만 지원됩니다. 일반적으로 와일드카드 대신 리소스 태그와 기타 지원되는 속성을 사용하여 이러한 리소스에 대한 액세스를 제어하는 것이 좋습니다. 자세한 내용은 Amazon RDS에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

Amazon Aurora 블루/그린 배포 관련 제한 사항 및 고려 사항

Amazon RDS에서 블루/그린 배포를 하려면 복제 슬롯, 리소스 관리, 인스턴스 크기 조정, 데이터베이스 성능에 미치는 잠재적 영향과 같은 요소를 신중하게 고려해야 합니다. 다음 섹션은 데이터베이스 환

경의 가동 중지 시간을 최소화하고, 원활한 전환을 보장하고, 효과적으로 관리할 수 있도록 배포 전략을 최적화하는 데 도움이 되는 지침을 제공합니다.

주제

블루/그린 배포 관련 제한 사항

블루/그린 배포에는 다음과 같은 제한 사항이 적용됩니다.

주제

블루/그린 배포 관련 일반 제한 사항

블루/그린 배포에는 다음과 같은 일반 제한 사항이 적용됩니다.

- 블루/그린 배포의 일부인 클러스터는 중지 및 시작이 불가능합니다.
- 블루/그린 배포의 경우 AWS Secrets Manager에서 마스터 사용자 암호 관리를 지원하지 않습니다.
- 블루 DB 클러스터를 강제로 역추적하려고 하면 블루/그린 배포가 중단되고 전환이 차단됩니다.
- 전환 중에는 블루 및 그린 환경을 Amazon Redshift와 제로 ETL로 통합할 수 없습니다. 먼저 통합을 삭제하고 전환한 다음 통합을 다시 생성해야 합니다.
- 블루/그린 배포를 생성할 때는 그린 환경에서 Event Scheduler(`event_scheduler` 파라미터)를 비활성화해야 합니다. 이렇게 하면 그린 환경에서 이벤트가 생성되어 불일치가 발생하는 것을 방지할 수 있습니다.
- 블루 DB 클러스터에 구성된 오토 스케일링 정책은 그린 환경으로 복사되지 않습니다. 처음에 블루 또는 그린 환경에 설정되었는지와 관계없이 전환 후 다시 구성해야 합니다.
- 암호화되지 않은 DB 클러스터는 암호화된 DB 클러스터로 변경할 수 없습니다. 더욱이 암호화된 DB 클러스터는 암호화되지 않은 DB 클러스터로 변경할 수 없습니다.
- 블루 DB 클러스터를 해당하는 그린 DB 클러스터보다 높은 엔진 버전으로 변경할 수 없습니다.
- 블루 환경과 그린 환경의 리소스는 동일한 AWS 계정에 있어야 합니다.
- 블루/그린 배포는 다음 기능에는 지원되지 않습니다.

- Amazon RDS Proxy
- 리전 간 읽기 전용 복제본
- Aurora Serverless v1 DB 클러스터
- Aurora Global Database의 일부인 DB 클러스터
- AWS CloudFormation

블루/그린 배포에 대한 Aurora MySQL 제한 사항

Aurora MySQL 블루/그린 배포에는 다음과 같은 제한 사항이 적용됩니다.

- 소스 DB 클러스터에는 이름이 `tmp`인 데이터베이스가 포함될 수 없습니다. 이 이름을 가진 데이터베이스는 그린 환경으로 복사되지 않습니다.
- 블루 DB 클러스터는 외부 binlog 복제본이 될 수 없습니다.
- 역추적이 설정된 소스 DB 클러스터가 활성화된 경우 역추적 지원 없이 그린 DB 클러스터가 생성됩니다. 블루/그린 배포에 필요한 이진 로그(binlog) 복제본에서는 역추적이 작동하지 않기 때문입니다. 자세한 내용은 [the section called “DB 클러스터 역추적”]() 섹션을 참조하세요.
- 블루/그린 배포는 MySQL용 AWS JDBC 드라이버를 지원하지 않습니다. 자세한 내용은 GitHub의 [Known Limitations]()를 참조하세요.

블루/그린 배포에 대한 Aurora PostgreSQL 제한 사항

다음 제한 사항은 Aurora PostgreSQL 블루/그린 배포에 적용됩니다.

- 블루 DB 클러스터에서 `rds.logically_replicate_unlogged_tables` 파라미터가 `1`로 설정되지 않은 경우 [로깅되지 않은]() 표는 그린 환경으로 복제되지 않습니다. 블루/그린 배포를 생성한 후에는 로깅되지 않은 테이블에서 발생할 수 있는 복제 오류를 방지하기 위해 이 파라미터 값을 수정하지 마세요.
- 블루 DB 클러스터는 논리적 소스(게시자) 또는 복제본(구독자)일 수 없습니다.
- 블루 DB 클러스터가 외부 데이터 래퍼(FDW) 확장의 외부 서버로 구성된 경우 IP 주소 대신 클러스터 엔드포인트 이름을 사용해야 합니다. 이렇게 하면 전환 후에도 구성이 계속 작동합니다.
- 블루/그린 배포에서는 각 데이터베이스에 논리적 복제 슬롯이 필요합니다. 데이터베이스 수가 늘어나면 리소스 오버헤드가 증가하고, 특히 DB 클러스터 규모가 충분히 조정되지 않은 경우 복제 지연이 발생할 수 있습니다. 영향은 데이터베이스 워크로드 및 연결 수와 같은 요인에 따라 달라집니다. 이를 완화하려면 DB 인스턴스 클래스 규모를 조정하거나 소스 클러스터의 데이터베이스 수를 줄이는 것이 좋습니다.

- 블루/그린 배포는 Babelfish for Aurora PostgreSQL 버전 15.7 이상 15 버전 및 16.3 이상 16 버전에서만 지원됩니다.
- Aurora 복제본에서 실행 계획을 캡처하려면 `apg_plan_mgmt.create_replica_plan_capture` 함수를 호출할 때 블루 DB 클러스터 엔드포인트를 제공해야 합니다. 이렇게 하면 전환 후에도 계획 캡처가 계속 작동합니다. 자세한 내용은 the section called "복제본에서 Aurora PostgreSQL 실행 계획 캡처" 섹션을 참조하세요.
- 그린 환경의 논리적 복제 적용 프로세스는 단일 스레드입니다. 블루 환경이 대량의 쓰기 트래픽을 생성하는 경우 그린 환경이 따라가지 못할 수 있습니다. 이로 인해 복제 지연 또는 실패가 발생할 수 있으며, 특히 쓰기 처리량이 지속적으로 높은 워크로드의 경우 더욱 그렇습니다. 워크로드를 철저히 테스트해야 합니다. 메이저 버전 업그레이드와 대용량 쓰기 워크로드 처리가 필요한 시나리오의 경우 AWS Database Migration Service(AWS DMS) 또는 자체 관리형 논리적 복제를 사용하는 등의 대체 접근 방식을 고려합니다.
- Aurora에 대한 블루/그린 배포 중에는 파티셔닝된 테이블에 새 파티션을 생성할 수 없습니다. 새 파티션을 생성하려면 블루 환경에서 그린 환경으로 복제되지 않는 `CREATE TABLE` 등의 데이터 정의 언어(DDL) 작업이 필요합니다. 그러나 기존에 파티셔닝된 테이블과 해당 데이터는 그린 환경에 복제됩니다.
- PostgreSQL 확장에는 다음과 같은 제한 사항이 적용됩니다.
  - 블루/그린 배포를 만들 때는 블루 환경에서 `pg_partman` 확장을 비활성화해야 합니다. 확장은 블루 환경에서 그린 환경으로의 논리적 복제를 중단하는 `CREATE TABLE`과 같은 DDL 작업을 수행합니다.
  - 블루/그린 배포를 생성한 후에는 모든 그린 데이터베이스에서 `pg_cron` 확장을 비활성화한 상태로 유지해야 합니다. 확장에는 슈퍼유저로 실행되고 그린 환경의 읽기 전용 설정을 우회하는 백그라운드 작업자가 있어 복제 충돌이 발생할 수 있습니다.
  - 블루 환경에서 동일한 계획을 캡처하는 경우 프라이머리프라이머리 키 충돌을 방지하려면 모든 그린 데이터베이스에서 `apg_plan_mgmt` 확장의 `apg_plan_mgmt.capture_plan_baselines` 파라미터를 `off`로 설정해야 합니다. 자세한 내용은 the section called "Aurora PostgreSQL 쿼리 계획 관리 개요" 섹션을 참조하세요.
  - 블루/그린 배포를 만들 때는 블루 환경에서 `pglogical` 및 `pgactive` 확장을 비활성화해야 합니다. 그린 환경을 새로운 프로덕션 환경으로 전환한 후 확장 기능을 다시 활성화할 수 있습니다. 또한 블루 데이터베이스는 외부 인스턴스의 논리적 구독자가 될 수 없습니다.
  - `pgAudit` 확장을 사용하는 경우, 해당 확장은 블루 및 그린 DB 인스턴스 모두에 대한 사용자 지정 DB 파라미터 그룹의 공유 라이브러리(`shared_preload_libraries`)에 남아 있어야 합니다. 자세한 내용은 the section called "pgAudit 확장 설정" 섹션을 참조하세요.

블루/그린 배포의 논리적 복제 관련 제한 사항

PostgreSQL에는 논리적 복제와 관련된 몇 가지 제한 사항이 있으며, 이로 인해 Aurora PostgreSQL DB 클러스터에 대한 블루/그린 배포를 생성할 때 제한이 적용됩니다.

다음 표에는 Aurora PostgreSQL의 블루/그린 배포에 적용되는 논리적 복제 제한 사항이 설명되어 있습니다. 자세한 내용은 PostgreSQL 논리적 복제 설명서의 제한 사항을 참조하세요.

제한 사항	설명
`CREATE TABLE` 및 `CREATE SCHEMA`와 같은 데이터 정의 언어(DDL) 문은 블루 환경에서 그린 환경으로 복제되지 않습니다.	Aurora가 블루 환경에서 DDL 변경을 감지하면 그린 데이터베이스는 복제 성능 저하 상태로 전환됩니다. 블루/그린 배포와 모든 그린 데이터베이스를 삭제한 다음 다시 생성해야 합니다.
`GRANT` 및 `REVOKE`와 같은 데이터 제어 언어(DCL) 문은 블루 환경에서 그린 환경으로 복제되지 않습니다.	Aurora이 블루 환경에서 DCL 문 실행 시도를 감지하면 경고 메시지가 표시됩니다. 블루/그린 배포 프로세스의 제한 사항이므로 이 동작을 변경하는 데 사용할 수 있는 구성 또는 API는 없습니다.
시퀀스 객체에 대한 `NEXTVAL` 작업은 블루 환경과 그린 환경 간에 동기화되지 않습니다.	전환 중에 Aurora는 그린 환경의 시퀀스 값을 증가시켜 블루 환경의 시퀀스 값과 일치하도록 합니다. 수천 개의 시퀀스가 있는 경우 전환이 지연될 수 있습니다.
블루 환경의 대형 객체는 그린 환경에 복제되지 않습니다. 여기에는 기존 대형 객체와 블	Aurora가 블루 환경에서 **`pg_largeobject`** 시스템 테이블에 저장된 대형 객체의 생성 또는 수정을 감지하면 그린 데이터베이스는 복제 성능 저하 상태로 전환됩니다. 블루/그린 배포와 모든 그린 데이터베이스를 삭제한 다음 다시 생성해야 합니다.

제한 사항	설명
루/그린 배포 프로세스 중에 새로 생성되거나 수정된 대형 객체가 모두 포함됩니다.	
구체화된 뷰를 새로 고치면 복제가 중단됩니다.	블루 환경에서 구체화된 뷰를 새로 고치면 그린 환경에 대한 복제가 중단됩니다. 블루 환경에서 구체화된 뷰를 새로 고치지 마세요. 전환 후에는 REFRESH MATERIALIZED VIEW 명령을 사용하여 수동으로 새로 고치거나 새로 고침 일정을 예약할 수 있습니다.
프라이머리프라이머리 키가 없는 테이블에서는 UPDATE 및 DELETE 작업이 허용되지 않습니다.	블루/그린 배포를 생성하기 전에 모든 테이블에 프라이머리 키가 있거나 `REPLICA IDENTITY FULL`을 사용하는지 확인하세요. 그러나 프라이머리 키 또는 고유 키가 없는 경우 `REPLICA IDENTITY FULL`만 사용합니다. 복제 성능에 영향을 미치기 때문입니다. 자세한 내용은 PostgreSQL 설명서를 참조하세요.

블루/그린 배포 관련 고려 사항

Amazon RDS는 각 리소스의 `DbiResourceId` 및 `DbClusterResourceId`를 사용하여 블루/그린 배포의 리소스를 추적합니다. 이 리소스 ID는 AWS 리전별로 고유한, 리소스의 변경 불가능한 식별자입니다.

리소스 ID는 DB 클러스터 ID와 다릅니다. 각 항목은 RDS 콘솔의 데이터베이스 구성에 나열되어 있습니다.

블루/그린 배포로 전환하면 리소스 이름(클러스터 ID)이 변경되지만, 각 리소스는 동일한 리소스 ID를 유지합니다. 예를 들어 DB 클러스터 식별자는 블루 환경에서는 `mycluster`입니다. 전환이 끝나면 동일한 DB 인스턴스의 이름이 `mycluster-old1`으로 변경됩니다. 하지만 DB 클러스터의 리소스 ID는 전환 중에 변경되지 않습니다. 따라서 그린 리소스를 새 프로덕션 리소스로 전환하면 관련 리소스 ID는 이전에 프로덕션에 있었던 블루 리소스 ID와 일치하지 않게 됩니다.

블루/그린 배포로 전환한 후에는, 리소스 ID를 프로덕션 리소스와 함께 사용한 통합형 기능 및 서비스를 위해 새로 전환된 프로덕션 리소스의 ID로 업데이트하는 것을 고려해 보세요. 구체적으로는 다음 업데이트를 고려해 보십시오.

- RDS API 및 리소스 ID를 사용하여 필터링을 수행할 경우, 전환 후 필터링에 사용한 리소스 ID를 조정하세요.
- 리소스 감사에 CloudTrail을 사용한다면, 전환 후 새 리소스 ID를 추적하도록 CloudTrail의 소비자를 조정하십시오. 자세한 내용은 AWS CloudTrail에서 Amazon Aurora API 호출 모니터링 섹션을 참조하세요.
- 블루 환경의 리소스에 데이터베이스 활동 스트림을 사용한다면, 전환 후 새 스트림에 대한 데이터베이스 이벤트를 모니터링하도록 애플리케이션을 조정하십시오. 자세한 내용은 데이터베이스 활동 스트림을 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.
- Performance Insights API를 사용한다면, 전환 후 API 호출의 리소스 ID를 조정하십시오. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.

  전환 후 동일한 이름의 데이터베이스를 모니터링할 수 있지만, 전환 전의 데이터는 포함되지 않습니다.
- IAM 정책에서 리소스 ID를 사용한다면, 필요한 경우 새로 전환된 리소스의 리소스 ID를 추가해야 합니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.
- DB 클러스터와 연결된 IAM 역할이 있는 경우 전환 후 해당 역할을 다시 연결해야 합니다. 연결된 역할은 그린 환경으로 자동 복사되지 않습니다.
- IAM 데이터베이스 인증을 사용하여 DB 클러스터를 인증하는 경우, 데이터베이스 액세스에 사용되는 IAM 정책의 `Resource` 요소 아래에 블루 데이터베이스와 그린 데이터베이스가 모두 나열되어 있는지 확인하세요. 이는 전환 후 그린 데이터베이스에 연결하기 위해 필요합니다. 자세한 내용은 the section called "IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용" 섹션을 참조하세요.
- 블루/그린 배포의 일부였던 DB 클러스터의 수동 DB 클러스터 스냅샷을 복원하려면, 스냅샷을 촬영한 시간을 확인하여 올바른 DB 클러스터 스냅샷을 복원해야 합니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.
- 전환한 후에는 블루 환경의 체크포인트가 그린 환경에서 유효하지 않기 때문에 AWS Database Migration Service(AWS DMS) 복제 작업을 재개할 수 없습니다. 복제를 계속하려면 새 체크포인트로 DMS 작업을 다시 만들어야 합니다.
- Amazon Aurora는 블루 환경에서 기본 Aurora 스토리지 볼륨을 복제하여 그린 환경을 만듭니다. 그린 클러스터 볼륨은 그린 환경에 대한 증분 변경만 저장합니다. 블루 환경의 DB 클러스터를 삭제하면 그린 환경의 기본 Aurora 스토리지 볼륨 크기가 최대 크기로 커집니다. 자세한 내용은 the section called "Aurora DB 클러스터에 대한 볼륨 복제" 섹션을 참조하세요.

- 블루/그린 배포의 그린 환경에 있는 DB 인스턴스에 DB 인스턴스를 추가하면, 새 DB 인스턴스는 전환할 때 블루 환경의 읽기 전용 복제본을 대체하지 않습니다. 하지만 새 DB 인스턴스는 DB 클러스터에 유지되며 새 프로덕션 환경에서는 DB 인스턴스가 됩니다.
- 블루/그린 배포의 그린 환경에 있는 DB 클러스터에서 DB 인스턴스를 삭제하면, 블루/그린 배포에서 이를 대체할 새 DB 인스턴스를 만들 수 없습니다.

  삭제된 DB 인스턴스와 동일한 이름 및 ARN으로 새 DB 인스턴스를 생성하면, `DbiResourceId`가 다르기 때문에 그린 환경에 속하지 않게 됩니다.

  그린 환경의 DB 클러스터에서 DB 인스턴스를 삭제하면 다음과 같은 동작이 발생합니다.

  - 블루 환경에 이름이 같은 DB 인스턴스가 있는 경우, 이 인스턴스는 그린 환경의 DB 인스턴스로 전환되지 않습니다. 이 DB 인스턴스는 DB 인스턴스 이름에 `-oldn`을 추가하여 이름이 바뀌지 않습니다.
  - 블루 환경의 DB 인스턴스를 가리키는 모든 애플리케이션은 전환 후에도 동일한 DB 인스턴스를 계속 사용합니다.

Amazon Aurora 블루/그린 배포 모범 사례

다음은 블루/그린 배포 모범 사례입니다.

주제

- 블루/그린 배포 일반 모범 사례
- 블루/그린 배포에 대한 Aurora MySQL 모범 사례
- 블루/그린 배포 모범 사례에 대한 Aurora PostgreSQL 모범 사례

블루/그린 배포 일반 모범 사례

블루/그린 배포를 만들 때 다음 일반 모범 사례를 고려하세요.

- 전환하기 전에 Aurora DB 클러스터를 철저하게 테스트합니다.
- 그린 환경에서 데이터베이스를 읽기 전용으로 유지합니다. 복제 충돌이 발생할 수 있으므로 그린 환경에서 쓰기 작업을 사용 설정할 때는 신중해야 합니다. 전환 후 프로덕션 데이터베이스에 의도하지 않은 데이터가 저장될 수도 있습니다.
- 블루/그린 배포를 사용하여 스키마 변경을 구현할 때는 복제와 호환되는 변경만 적용합니다.

예를 들어 블루 배포에서 그린 배포로의 복제를 중단하지 않고 테이블 끝에 새 열을 추가할 수 있습니다. 그러나 열 이름 변경이나 테이블 이름 변경 같은 스키마 변경은 그린 배포로의 복제 중단을 유발합니다.

복제 호환 변경 사항에 대한 자세한 내용은 MySQL 설명서의 소스 및 복제본에서 서로 다른 테이블 정의를 사용하는 복제 및 PostgreSQL 논리적 복제 설명서의 제한 사항을 참조하세요.

- 두 환경의 모든 연결에 클러스터 엔드포인트, 리더 엔드포인트 또는 사용자 지정 엔드포인트를 사용합니다. 정적 또는 제외 목록과 함께 인스턴스 엔드포인트나 사용자 지정 엔드포인트를 사용하지 않습니다.
- 블루/그린 배포로 전환할 때는 전환 모범 사례를 따르세요. 자세한 내용은 the section called "전환 모범 사례" 섹션을 참조하세요.

블루/그린 배포에 대한 Aurora MySQL 모범 사례

Aurora MySQL DB 클러스터에서 블루/그린 배포를 만들 때 다음 모범 사례를 고려하세요.

- 그린 환경에 복제본 지연이 발생하는 경우 다음을 고려하세요.
  - 필요하지 않은 경우 이진 로그 보존을 비활성화하거나 복제가 회수될 때까지 일시적으로 비활성화합니다. 이렇게 하려면 `binlog_format` DB 클러스터 파라미터를 `0`으로 다시 설정하고 그린 라이터 DB 인스턴스를 재부팅합니다.
  - 그린 DB 파라미터 그룹에서 `innodb_flush_log_at_trx_commit` 파라미터를 일시적으로 `0`으로 설정합니다. 복제가 최신 상태로 동기화된 후 전환하기 전에 `1`의 기본값으로 되돌립니다. 임시 파라미터 값에서 예기치 않은 종료 또는 충돌이 발생하는 경우 탐지되지 않은 데이터 손상을 방지하기 위해 그린 환경을 다시 빌드합니다. 자세한 내용은 the section called "로그 버퍼를 플러시하는 빈도 구성" 단원을 참조하세요.

블루/그린 배포 모범 사례에 대한 Aurora PostgreSQL 모범 사례

Aurora PostgreSQL DB 클러스터에서 블루/그린 배포를 만들 때 다음 모범 사례를 고려하세요.

- Aurora PostgreSQL 논리적 복제 라이트-스루 캐시를 모니터링하고 필요한 경우 캐시 버퍼를 조정합니다. 자세한 내용은 the section called "논리적 복제 라이트-스루 캐시 모니터링" 섹션을 참조하세요.
- 블루 환경에서 `logical_decoding_work_mem` DB 파라미터의 값을 늘립니다. 이렇게 하면 디스크 디코딩이 줄어들고 대신 메모리가 사용됩니다. 자세한 내용은 the section called "논리적 디코딩을 위한 작업 메모리 조정" 섹션을 참조하세요.

  - `ReplicationSlotDiskUsage` CloudWatch 지표를 사용하여 디스크에 기록되는 트랜잭션 오버플로를 모니터링할 수 있습니다. 이 지표는 복제 슬롯의 디스크 사용량에 대한 인사이트를 제공하므로 트랜잭션 데이터가 메모리 용량을 초과하고 디스크에 저장되는 시점을 식별하는 데 도움이 됩니다. `FreeableMemory` CloudWatch 지표로 여유 메모리를 모니터링할 수 있습니다. 자세한 내용은 the section called “Amazon Aurora에 대한 인스턴스 수준 지표” 섹션을 참조하세요.
  - Aurora PostgreSQL 버전 14 이상에서는 `pg_stat_replication_slots` 시스템 뷰를 사용하여 논리적 오버플로 파일의 크기를 모니터링할 수 있습니다.
- 블루/그린 배포를 생성하기 전에 모든 PostgreSQL 확장을 최신 버전으로 업데이트합니다. 자세한 내용은 the section called “PostgreSQL 확장 버전 업그레이드” 섹션을 참조하세요.
- `aws_s3` 확장을 사용하는 경우 그린 환경이 만들어진 후 IAM 역할을 통해 그린 DB 클러스터에 Amazon S3에 대한 액세스 권한을 부여합니다. 이렇게 하면 전환 후에도 가져오기 및 내보내기 명령이 계속 작동합니다. 지침은 the section called “Amazon S3 버킷에 대한 액세스 권한 설정” 섹션을 참조하세요.
- 그린 환경에 더 높은 엔진 버전을 지정하는 경우 모든 데이터베이스에서 `ANALYZE` 작업을 실행하여 `pg_statistic` 테이블을 새로 고칩니다. 옵티마이저 통계는 메이저 버전 업그레이드 중에 전송되지 않으므로 성능 문제를 방지하려면 모든 통계를 다시 생성해야 합니다. 메이저 버전 업그레이드 중 추가 모범 사례에 대해 알아보려면 the section called “메이저 버전 업그레이드 수행” 섹션을 참조하세요.
- 데이터를 조작하기 위해 트리거를 소스에 사용하는 경우 트리거를 `ENABLE REPLICA` 또는 `ENABLE ALWAYS`로 구성하지 마세요. 이렇게 구성하면 복제 시스템이 변경 내용을 전파하고 트리거를 실행하므로 중복이 발생합니다.
- 트랜잭션이 오래 실행되면 심각한 복제 지연이 발생할 수 있습니다. 복제 지연을 줄이려면 다음과 같이 해 보세요.
  - 그린 환경이 블루 환경을 따라잡을 때까지 지연될 수 있는 장기 실행 트랜잭션 및 하위 트랜잭션을 줄이세요.
  - 그린 환경이 블루 환경을 따라잡을 때까지 블루 환경의 대량 작업을 줄이세요.
  - 블루/그린 배포를 생성하기 전에 사용량이 많은 테이블에서 수동 vacuum freeze 작업을 시작하세요. 지침은 수동 vacuum freeze 수행을 참조하세요.
  - PostgreSQL 버전 12 이상에서는 크거나 사용량이 많은 테이블에서 `index_cleanup` 파라미터를 비활성화하여 블루 데이터베이스의 정기 유지 관리의 효율성을 개선하세요. 자세한 내용은 테이블에 최대한 신속하게 vacuum 실행을 참조하세요.

> **Note**
>
> Vacuum 중에 인덱스 정리를 정기적으로 건너뛰면 인덱스 팽창이 발생하여 스캔 성능이 저하될 수 있습니다. 블루/그린 배포를 사용하는 동안에만 이 접근 방식을 사용하는 것이 가장 좋습니다. 배포가 완료되면 정기적인 인덱스 유지 관리 및 정리를 재개하는 것이 좋습니다.

  - 블루 및 그린 DB 인스턴스가 워크로드에 비해 크기가 작으면 복제 지연이 발생할 수 있습니다. DB 인스턴스가 인스턴스 유형에 대한 리소스 제한에 도달하지 않는지 확인합니다. 자세한 내용은 the section called “CloudWatch 지표를 사용한 리소스 사용량 분석” 섹션을 참조하세요.

- 복제 속도가 느리면 발신자와 수신자가 자주 다시 시작되어 동기화가 지연될 수 있습니다. 이들의 활성 상태를 유지하려면 블루 환경에서는 `wal_sender_timeout` 파라미터를 `0`으로 설정하고 그린 환경에서는 `wal_receiver_timeout` 파라미터를 `0`으로 설정하여 시간 초과를 비활성화하세요.
- UPDATE 및 DELETE 문 성능을 검토하고 WHERE 절에서 사용되는 열에 인덱스를 생성하여 이러한 쿼리를 최적화할 수 있는지 평가합니다. 이렇게 하면 그린 환경에서 작업을 재생할 때 성능이 향상될 수 있습니다. 자세한 내용은 the section called “대기를 생성하는 쿼리에 대한 술어 필터 확인” 단원을 참조하세요.
- 트리거를 사용하는 경우 이름이 'rds'로 시작하는 `pg_catalog.pg_publication`, `pg_catalog.pg_subscription` 및 `pg_catalog.pg_replication_slots` 객체를 만들고, 업데이트하고, 삭제하는 데 트리거가 방해가 되지 않는지 확인하세요.
- 소스 DB 클러스터에서 Babelfish를 활성화한 경우 다음 파라미터는 그린 환경의 대상 DB 클러스터 파라미터 그룹에서 소스 DB 클러스터 파라미터 그룹과 동일한 설정을 가져야 합니다.
  - rds.babelfish_status
  - babelfishpg_tds.tds_default_numeric_precision
  - babelfishpg_tds.tds_default_numeric_scale
  - babelfishpg_tsql.default_locale
  - babelfishpg_tsql.migration_mode
  - babelfishpg_tsql.server_collation_name

  이런 파라미터에 대한 자세한 내용은 the section called “Babelfish용 DB 클러스터 파라미터 그룹 설정 ” 섹션을 참조하세요.

Amazon Aurora에서 블루/그린 배포 생성

RDS는 블루 환경의 토폴로지와 기능을 스테이징 영역에 복사합니다. 블루 DB 인스턴스에 읽기 전용 복제본이 있는 경우, 그린 DB 인스턴스의 읽기 전용 복제본으로 복사됩니다. 모든 그린 복제본의 할당된 스토리지는 그린 기본 인스턴스와 일치하는 반면, 다른 스토리지 파라미터는 블루 복제본에서 상속됩니다.

블루/그린 배포를 만들 때는 배포에서 복사할 DB 인스턴스를 지정해야 합니다. 선택한 DB 클러스터는 프로덕션 DB 클러스터이며, 블루 환경에서는 이것이 DB 클러스터가 됩니다. RDS는 블루 환경의 토폴로지를 구성된 기능과 함께 스테이징 영역에 복사합니다. DB 클러스터는 그린 환경으로 복사되고, RDS는 블루 환경의 DB 클러스터에서 그린 환경의 DB 클러스터로의 복제를 구성합니다. 또한 RDS는 DB 클러스터에 있는 모든 DB 인스턴스를 복제합니다.

주제

- 블루/그린 배포 준비
- 블루/그린 배포 생성 시 변경 사항 지정
- 블루/그린 배포 생성
- 블루/그린 배포를 생성하기 위한 설정

블루/그린 배포 준비

Aurora DB 클러스터가 실행 중인 엔진에 따라 블루/그린 배포를 생성하기 전에 수행해야 하는 몇 가지 단계가 있습니다.

주제

- 블루/그린 배포를 위해 Aurora MySQL DB 클러스터 준비
- 블루/그린 배포를 위해 Aurora PostgreSQL DB 클러스터 준비

블루/그린 배포를 위해 Aurora MySQL DB 클러스터 준비

Aurora MySQL DB 클러스터의 블루/그린 배포를 생성하기 전에, DB 클러스터를 바이너리 로깅(`binlog_format`)을 켠 사용자 지정 클러스터 파라미터 그룹과 연결해야 합니다. 블루 환경에서 그린 환경으로 복제하려면 이진 로깅이 필요합니다. 모든 binlog 형식을 사용할 수 있지만 복제 불일치의 위험을 줄이기 위해 `ROW`를 사용하는 것이 좋습니다. 사용자 지정 DB 파라미터 그룹 생성에 대한 자세한 내용은 the section called “DB 클러스터 파라미터 그룹” 섹션을 참조하세요.

> **Note**
>
> 이진 로깅을 활성화하면 DB 클러스터에 대한 평균 디스크 쓰기 I/O 작업 수가 증가합니다. `VolumeWriteIOPs` CloudWatch 지표를 사용하여 IOPS 사용량을 모니터링할 수 있습니다.

이진 로깅을 활성화한 후에는 DB 클러스터를 재부팅하여 변경 사항을 적용해야 합니다. 블루/그린 배포의 요구 사항은 라이터 인스턴스가 DB 클러스터 파라미터 그룹과 동기화되는 것이며, 그렇지 않으면 생성에 실패합니다. 자세한 내용은 Aurora 클러스터 내의 DB 인스턴스 재부팅 섹션을 참조하세요.

또한 이진 로그 파일이 제거되지 않도록 이진 로그 보존 기간을 NULL 외의 다른 값으로 변경하는 것이 좋습니다. 자세한 내용은 the section called “이진수 로그 구성 설정 및 표시” 섹션을 참조하세요.

블루/그린 배포를 위해 Aurora PostgreSQL DB 클러스터 준비

Aurora PostgreSQL DB 클러스터에 블루/그린 배포를 만들기 전에 먼저 다음을 수행하세요.

- 논리적 복제(`rds.logical_replication`)가 활성화된 사용자 지정 DB 클러스터 파라미터 그룹과 클러스터를 연결합니다. 블루 환경에서 그린 환경으로 복제하려면 논리적 복제가 필요합니다.

  논리적 복제를 활성화할 때는 `max_replication_slots`, `max_logical_replication_workers`, `max_worker_processes` 같은 특정 클러스터 파라미터도 조정해야 합니다. 논리적 복제를 활성화하고 이러한 파라미터를 조정하는 방법에 대한 지침은 the section called “논리적 복제 설정” 섹션을 참조하세요.

  또한 `synchronous_commit` 파라미터가 on으로 설정되어 있어야 합니다.

  필수 파라미터를 구성한 후에 DB 클러스터를 재부팅하여 변경 사항을 적용합니다. 블루/그린 배포의 요구 사항은 라이터 인스턴스가 DB 클러스터 파라미터 그룹과 동기화되는 것이며, 그렇지 않으면 생성에 실패합니다. 자세한 내용은 Aurora 클러스터 내의 DB 인스턴스 재부팅 섹션을 참조하세요.
- DB 클러스터가 블루/그린 배포와 호환되는 Aurora PostgreSQL 버전을 실행하는지 확인합니다. 호환 가능한 버전 목록은 the section called “Aurora PostgreSQL을 사용한 블루/그린 배포” 섹션을 참조하세요.
- DB 클러스터의 모든 테이블에 프라이머리 키가 있는지 확인하세요. PostgreSQL 논리적 복제는 프라이머리 키가 없는 테이블에 대한 UPDATE 또는 DELETE 작업을 허용하지 않습니다.

블루/그린 배포 생성 시 변경 사항 지정

블루/그린 배포를 생성할 때 그린 환경의 DB 클러스터에 다음 변경 사항을 적용할 수 있습니다.

배포가 끝나면 그린 환경의 DB 클러스터와 관련 DB 인스턴스를 추가로 수정할 수 있습니다. 예를 들어 더 높은 엔진 버전 또는 다른 파라미터 그룹을 지정할 수 있습니다.

DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정](#) 섹션을 참조하세요.

주제

- [더 높은 엔진 버전 지정](#)
- [다른 DB 파라미터 그룹 지정](#)

더 높은 엔진 버전 지정

DB 엔진 업그레이드를 테스트하고 싶다면 더 높은 엔진 버전을 지정할 수 있습니다. 전환 시 데이터베이스는 지정한 메이저 또는 마이너 DB 엔진 버전으로 업그레이드됩니다.

다른 DB 파라미터 그룹 지정

DB 인스턴스에서 사용하는 그룹과 다른 DB 클러스터 파라미터 그룹을 지정합니다. 그린 환경에서 파라미터 변경이 DB 클러스터에 미치는 영향을 테스트하거나, 업그레이드할 때 새 메이저 DB 엔진 버전에 대한 파라미터 그룹을 지정할 수 있습니다.

다른 DB 클러스터 파라미터 그룹을 지정하면, 지정된 파라미터 그룹이 그린 환경의 DB 클러스터와 연결됩니다. 다른 DB 클러스터 파라미터 그룹을 지정하지 않으면 그린 환경의 DB 클러스터는 블루 DB 클러스터와 동일한 파라미터 그룹과 연결됩니다.

블루/그린 배포 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 블루/그린 배포를 생성할 수 있습니다.

콘솔

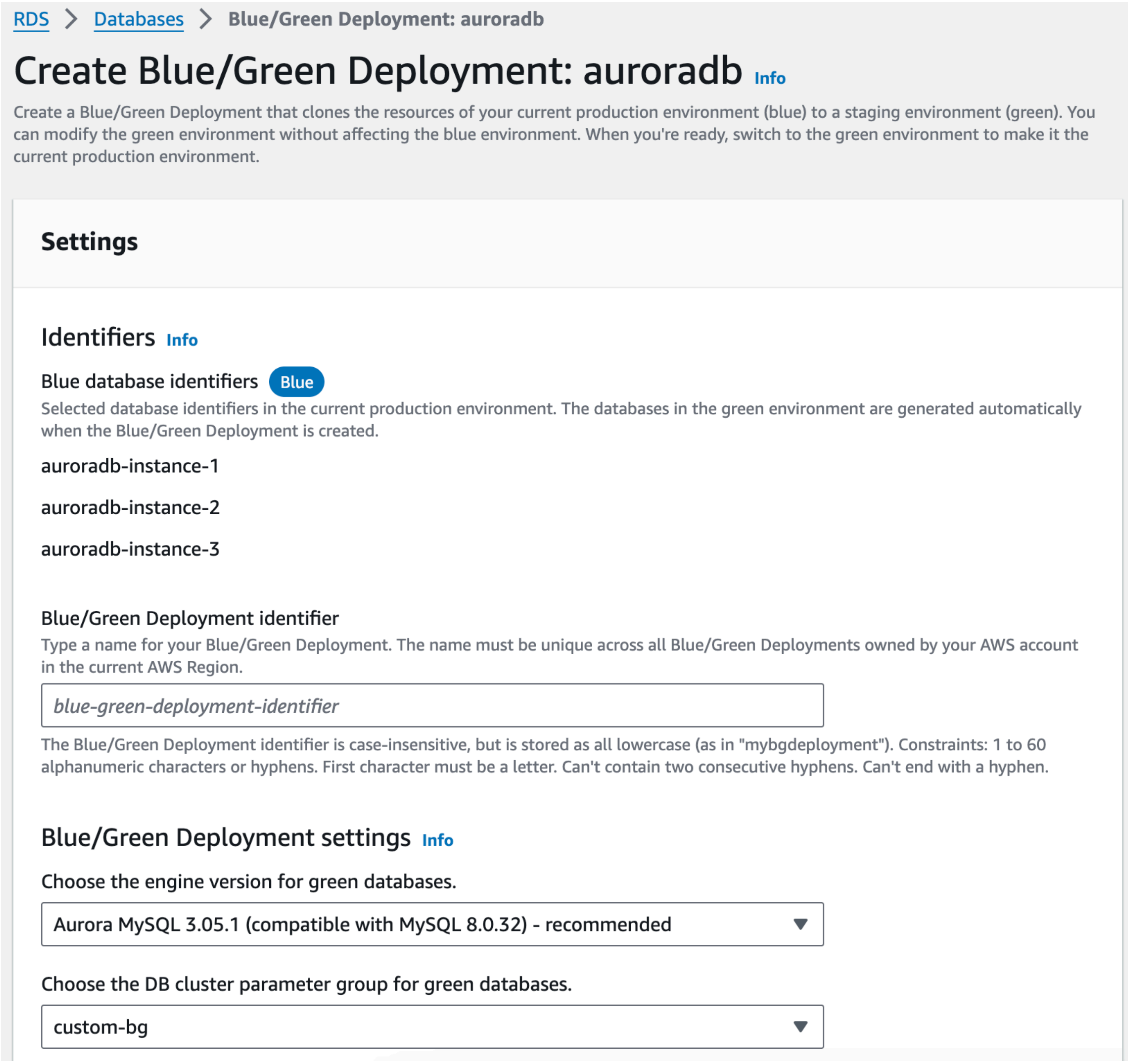
블루/그린 배포를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 그린 환경에 복사할 DB 클러스터를 선택합니다.

3. 작업, 블루/그린 배포 생성을 선택합니다.

   블루/그린 배포 생성 페이지가 표시됩니다.



4. 블루 데이터베이스 식별자를 검토합니다. 해당 식별자가 블루 환경에 있어야 하는 DB 인스턴스와 일치하는지 확인합니다. 일치하지 않는다면 Cancel(취소)를 선택합니다.

5. 블루/그린 배포 이름에 블루/그린 배포의 이름을 입력합니다.

6. 나머지 섹션에서 그린 환경에 대한 설정을 지정합니다. 각 설정에 대한 자세한 내용은 the section called “사용 가능한 설정” 단원을 참조하세요.

   배포가 끝나면 그린 환경의 데이터베이스를 추가로 수정할 수 있습니다.
7. 생성(Create)을 선택합니다.

AWS CLI

AWS CLI를 사용하여 블루/그린 배포를 생성하려면 create-blue-green-deployment 명령을 사용해야 합니다. 사용 가능한 모든 옵션에 대한 자세한 내용은 the section called “사용 가능한 설정” 섹션을 참조하세요.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

Windows의 경우:

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

RDS API

Amazon RDS API를 사용하여 블루/그린 배포를 생성하려면 `CreateBlueGreenDeployment` 작업을 사용해야 합니다. 각 옵션에 대한 자세한 내용은 the section called “사용 가능한 설정” 단원을 참조하세요.

블루/그린 배포를 생성하기 위한 설정

블루/그린 배포를 만들 때 선택할 수 있는 설정에 대한 설명은 다음 표에 나와 있습니다. AWS CLI 옵션에 대한 자세한 내용은 create-blue-green-deployment 섹션을 참조하세요. RDS API 파라미터에 대한 자세한 내용은 CreateBlueGreenDeployment를 참조하세요.

콘솔 설정	설정 설명	CLI 옵션 및 RDS API 파라미터

블루/그린 배포 식별자	블루/그린 배포의 이름	CLI 옵션:
		
		`--blue-green-deployment-name`
		
		API 파라미터:
		
		`BlueGreenDeploymentName`

블루 데이터베이스 식별자	그린 환경으로 복사할 클러스터의 식별자입니다. CLI 또는 API를 사용하는 경우 클러스터 Amazon 리소스 이름(ARN)을 지정합니다.	CLI 옵션:
		
		`--source`
		
		API 파라미터:
		
		`Source`

그린 데이터베이스를 위한 DB 클러스터 파라미터 그룹	그린 환경의 데이터베이스와 연결할 파라미터 그룹입니다.	CLI 옵션:
		
		`--target-db-cluster-parameter-group-name`
		
		API 파라미터:
		
		`TargetDBClusterParameterGroupName`

그린 데이터베이스를 위한 엔진 버전	그린 환경의 클러스터를 지정된 DB 엔진 버전으로 업그레이드합니다.
	
	Aurora PostgreSQL DB 클러스터를 선택하는 경우 논리적 복제 제한 사항을 검토하고 확인하세요. 자세한 내용은 the section called “블루/그린 배포의 논리적 복제 관련 제한 사항” 섹션을 참조하세요.
		CLI 옵션:
		
		`--target-engine-version`
		
		RDS API 파라미터:
		
		`TargetEngineVersion`

Amazon Aurora에서 블루/그린 배포 보기

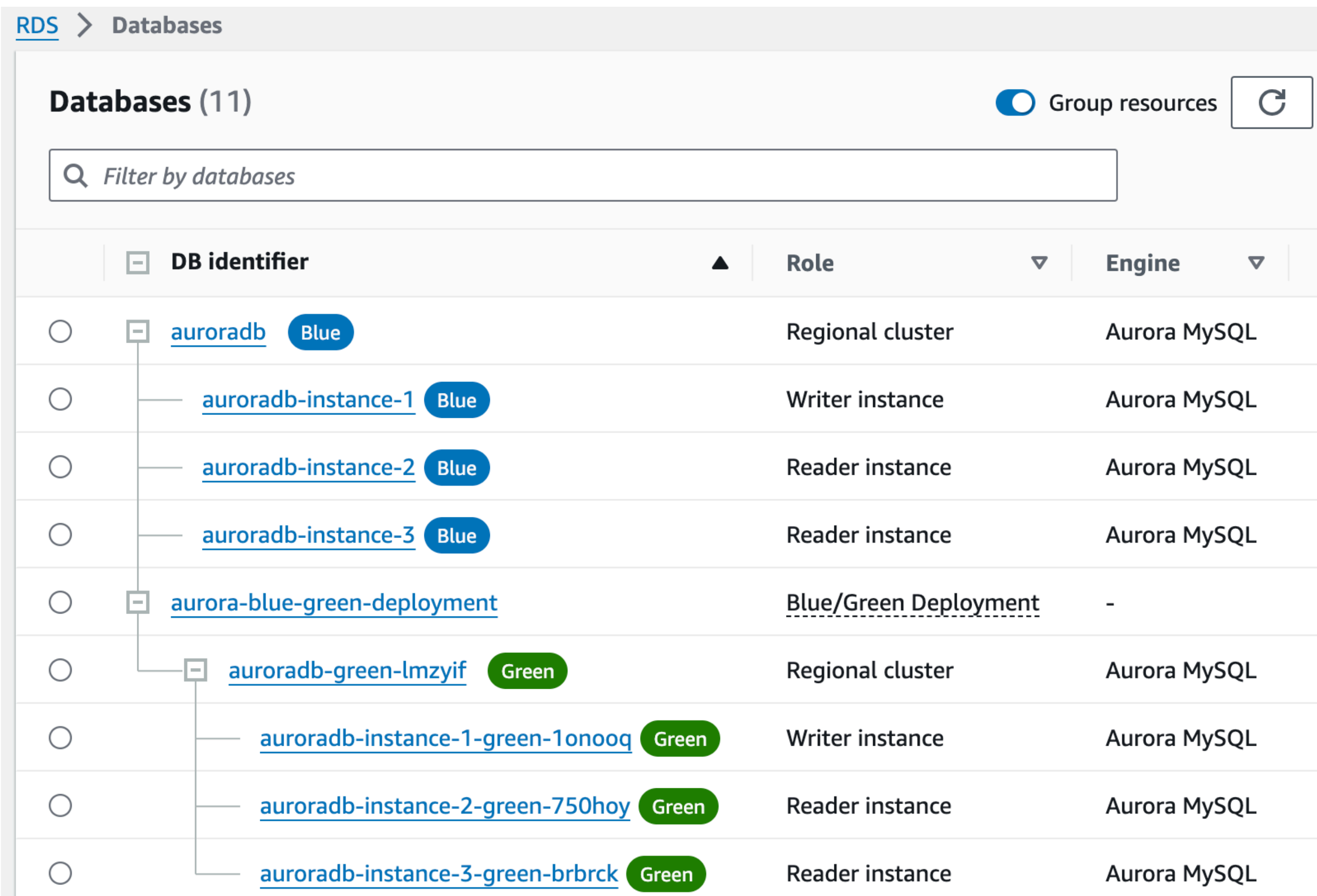
AWS Management Console, AWS CLI 또는 RDS API를 사용하여 블루/그린 배포의 세부 정보를 확인할 수 있습니다.

이벤트를 보고 구독하여 블루/그린 배포 관련 정보를 확인할 수도 있습니다. 자세한 내용은 블루/그린 배포 이벤트 단원을 참조하십시오.

콘솔

블루/그린 배포의 세부 정보를 확인하는 방법

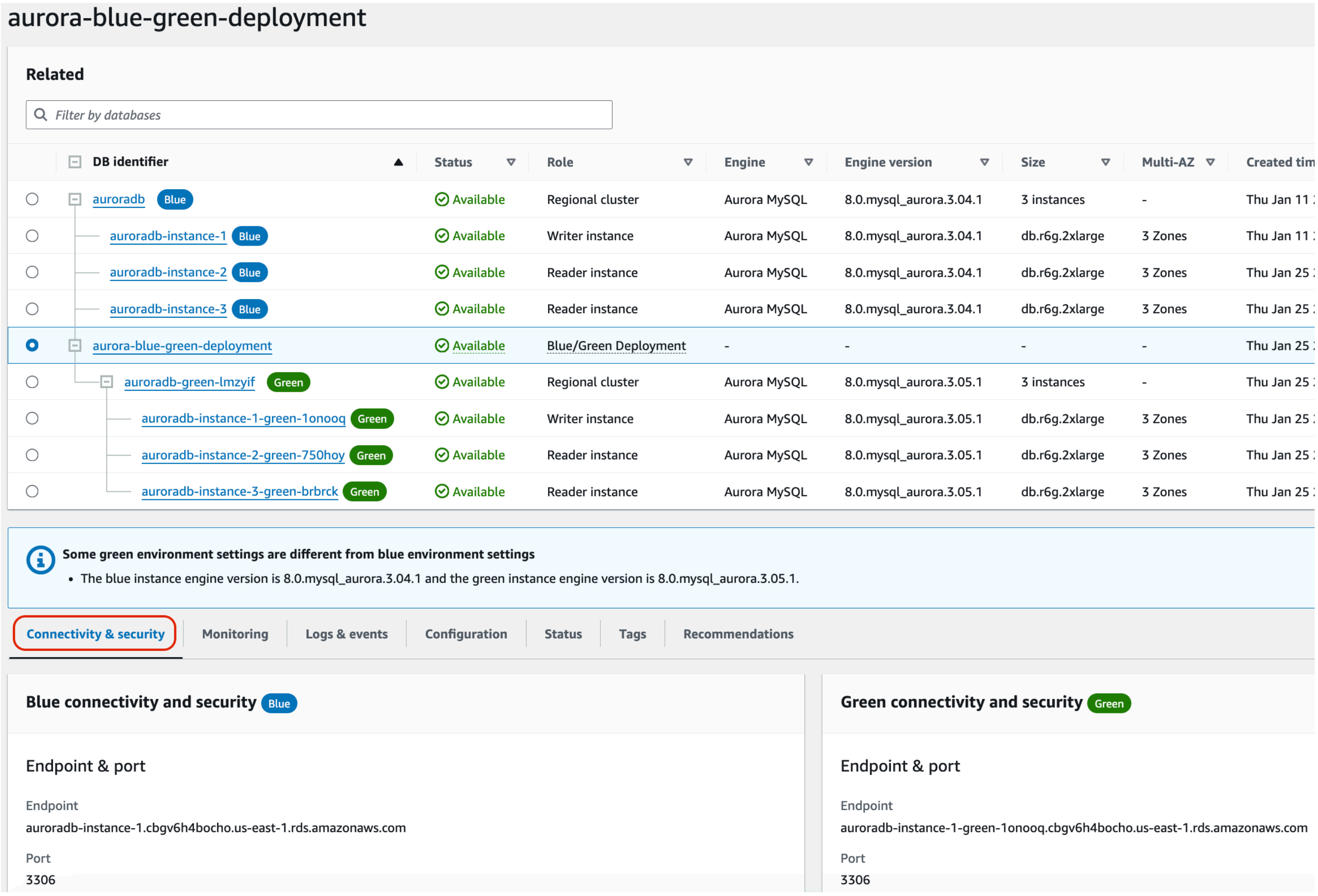
1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 목록에서 블루/그린 배포를 찾습니다.



   블루/그린 배포의 Role(역할) 값은 Blue/Green Deployment(블루/그린 배포)입니다.
3. 세부 정보를 표시할 블루/그린 배포의 이름을 선택합니다.

각 탭에는 블루 배포 섹션과 그린 배포 섹션이 있습니다. 예를 들어, 그린 환경에서 DB 엔진 버전을 업그레이드한다면 블루 환경과 그린 환경의 구성 탭에 표시되는 DB 엔진 버전이 다를 수 있습니다.

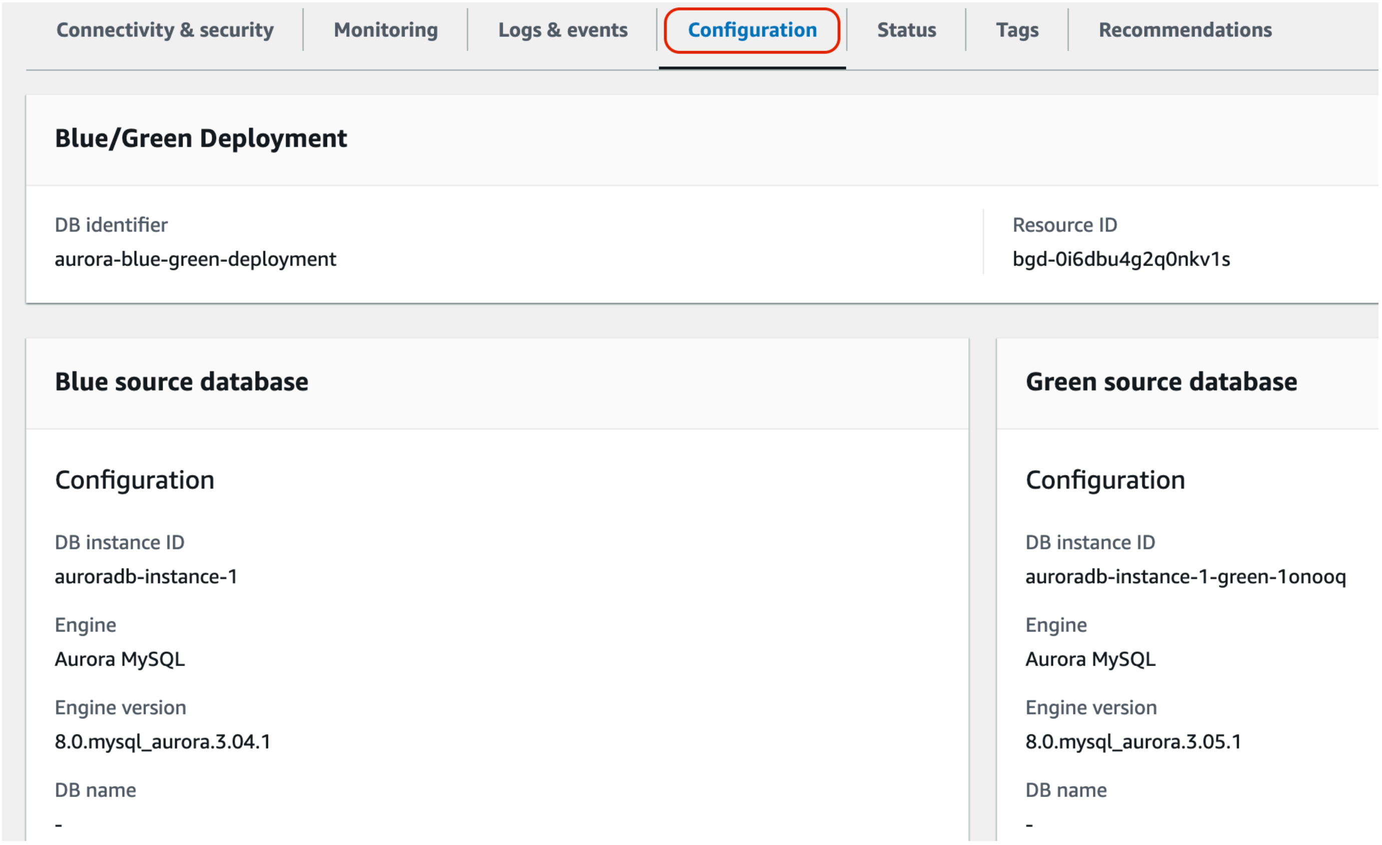
다음 이미지에서는 연결 및 보안 탭 예시를 확인할 수 있습니다.



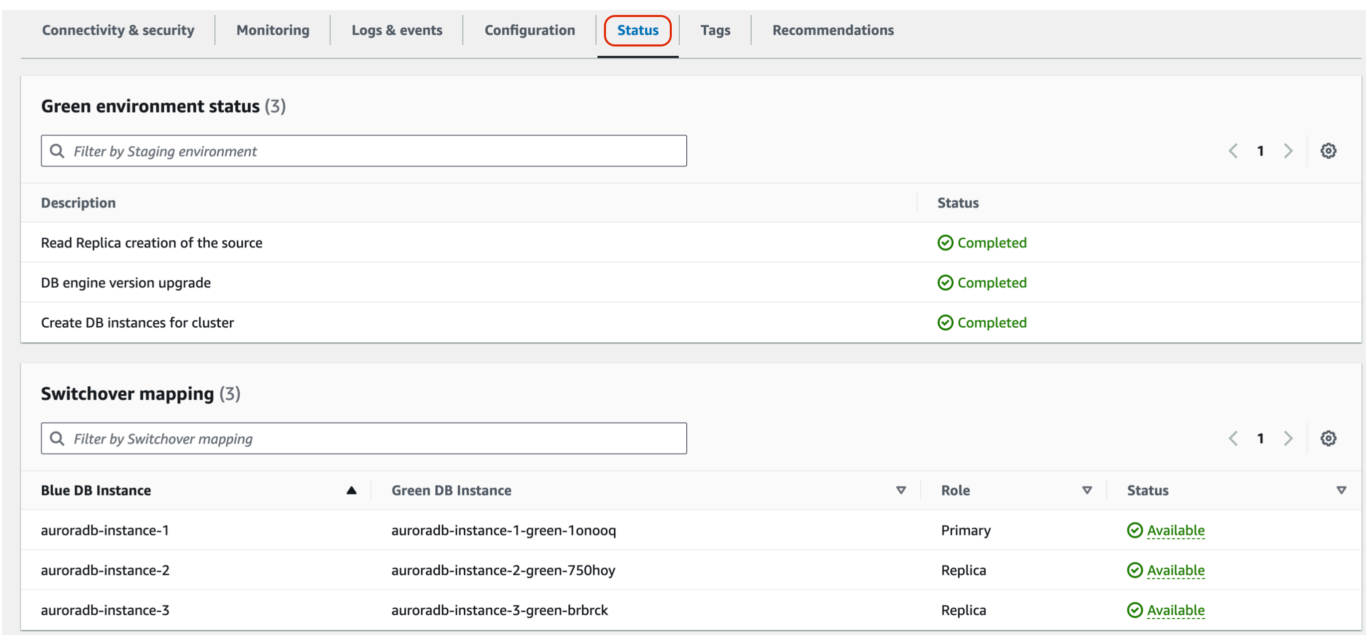
연결성 및 보안 탭에는 블루와 그린 환경 사이의 복제 및 복제 지연의 현재 상태를 보여주는 복제라는 섹션도 있습니다. 복제 상태가 `Replicating`인 경우 블루/그린 배포가 성공적으로 복제되고 있는 것입니다.

Aurora PostgreSQL 블루/그린 배포의 경우 블루 환경에서 지원되지 않는 DDL이나 대규모 객체를 변경하면 복제 상태가 `Replication degraded`로 변경될 수 있습니다. 자세한 내용은 the section called “블루/그린 배포의 논리적 복제 관련 제한 사항” 단원을 참조하십시오.

다음 이미지에서는 구성 탭 예시를 확인할 수 있습니다.



다음 이미지에서는 상태 탭 예시를 확인할 수 있습니다.



AWS CLI

AWS CLI를 사용하여 블루/그린 배포 세부 정보를 보려면 describe-blue-green-deployments 명령을 사용해야 합니다.

Example 이름을 필터링하여 블루/그린 배포 세부 정보 확인

describe-blue-green-deployments 명령을 사용하는 경우 `--blue-green-deployment-name`으로 필터링할 수 있습니다.

다음 예에서는 *my-blue-green-deployment*라는 블루/그린 배포의 세부 정보를 확인할 수 있습니다.

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example 식별자를 지정하여 블루/그린 배포 세부 정보 확인

describe-blue-green-deployments 명령을 사용하는 경우 `--blue-green-deployment-identifier` 옵션을 지정할 수 있습니다.

다음 예에서는 *bgd-1234567890abcdef*라는 식별자가 있는 블루/그린 배포의 세부 정보를 확인할 수 있습니다.

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

RDS API

Amazon RDS API를 사용하여 블루/그린 배포 세부 정보를 보려면 `DescribeBlueGreenDeployments` 작업을 사용하고 `BlueGreenDeploymentIdentifier`를 지정해야 합니다.

Amazon Aurora에서 블루/그린 배포 전환

전환은 그린 환경에 있는 DB 클러스터를 자체 DB 인스턴스와 함께 프로덕션 DB 클러스터로 전환합니다. 전환하기 전에는 프로덕션 트래픽이 블루 환경의 클러스터로 라우팅됩니다. 전환한 후에는 프로덕션 트래픽이 그린 환경의 DB 클러스터로 라우팅됩니다.

블루/그린 배포로의 전환은 블루/그린 배포 내에서 그린 DB 클러스터를 승격하는 것과 다릅니다. 작업 메뉴에서 승격을 선택하여 그린 DB 클러스터를 수동으로 승격하는 경우 블루 환경과 그린 환경 간의 복제가 중단되고 블루/그린 배포가 잘못된 구성 상태로 전환됩니다.

주제

전환 제한 시간

30초에서 3,600초(1시간) 사이의 전환 제한 시간을 지정할 수 있습니다. 전환이 지정된 기간보다 오래 걸린다면, 모든 변경 사항이 롤백되고 어느 환경도 변경되지 않습니다. 기본 제한 시간은 300초(5분)입니다.

전환 가드레일

전환을 시작하면 Amazon RDS는 몇 가지 기본 검사를 실행하여 블루 및 그린 환경이 전환 준비가 되었는지 테스트합니다. 이러한 검사를 전환 가드레일이라고 합니다. 이러한 전환 가드레일은 환경이 준비되지 않았다면 전환이 진행되지 않게 합니다. 따라서 예상보다 긴 다운타임을 방지하고, 전환이 시작되면 발생할 수 있는 블루 및 그린 환경 간의 데이터 손실을 방지할 수 있습니다.

Amazon RDS는 그린 환경에서 다음 가드레일 검사를 실행합니다.

- 복제 상태 – 그린 DB 클러스터 복제 상태가 정상인지 확인합니다. 그린 DB 클러스터는 블루 DB 클러스터의 복제본입니다.
- 복제 지연 - 그린 DB 클러스터의 복제 지연이 전환의 허용 한계 이내인지 확인합니다. 허용 한계는 지정된 제한 시간을 기준으로 합니다. 복제 지연은 그린 DB 클러스터가 자신의 블루 DB 클러스터보다 얼마나 뒤쳐져 있는지를 나타냅니다. 자세한 내용은 the section called “전환 전 복제 지연 모니터링” 단원을 참조하십시오.

- 활성 쓰기 - 그린 DB 클러스터에서 활성 쓰기가 없는지 확인합니다.

Amazon RDS는 블루 환경에서 다음 가드레일 검사를 실행합니다.

- 외부 복제 - Aurora PostgreSQL의 경우 블루 환경이 자체 관리되는 논리적 소스(게시자) 또는 복제본(구독자)이 아니어야 합니다. 자체 관리되는 논리적 소스 또는 복제본인 경우 블루 환경의 모든 데이터베이스에서 자체 관리형 복제 슬롯과 구독을 삭제하고 전환을 진행한 다음 다시 생성하여 복제를 재개하는 것이 좋습니다. Aurora MySQL의 경우 파란색 데이터베이스가 외부 binlog 복제본이 아닌지 확인합니다. 있는 경우 활발하게 복제하고 있지 않은지 확인합니다.
- 장기 실행 활성 쓰기 - 블루 DB 클러스터에 복제 지연을 늘릴 수 있는 장기 실행 활성 쓰기가 없는지 확인합니다.
- 장기 실행 DDL 문 - 블루 DB 클러스터에 복제 지연을 늘릴 수 있는 장기 실행 DDL 문이 없는지 확인합니다.
- 지원되지 않는 PostgreSQL 변경 - Aurora PostgreSQL의 경우 블루 환경에서 DDL 변경이나 대형 객체의 추가 또는 수정이 수행되지 않았는지 확인합니다. 자세한 내용은 the section called “블루/그린 배포의 논리적 복제 관련 제한 사항” 단원을 참조하십시오.

  Amazon RDS가 지원되지 않는 PostgreSQL 변경을 감지하면 복제 상태를 `Replication degraded`로 변경하고 블루/그린 배포에서 전환을 사용할 수 없음을 알립니다. 전환을 진행하려면 블루/그린 배포와 모든 그린 데이터베이스를 삭제하고 다시 생성하는 것이 좋습니다. 이렇게 하려면 작업, 그린 데이터베이스를 사용한 삭제를 선택합니다.

전환 작업

블루/그린 배포로 전환하면 RDS는 다음 작업을 수행합니다.

1. 가드레일 검사를 실행하여 블루 및 그린 환경이 전환 준비가 되었는지 확인합니다.
2. 두 환경 모두에서 DB 클러스터에 대한 신규 쓰기 작업을 중지합니다.
3. 두 환경 모두에서 DB 인스턴스에 대한 연결을 끊고 새 연결을 허용하지 않습니다.
4. 그린 환경이 블루 환경과 동기화되도록, 그린 환경에서 복제가 충분히 진행될 때까지 기다립니다.
5. 두 환경 모두에서 DB 클러스터와 DB 인스턴스의 이름을 바꿉니다.

   RDS는 그린 환경의 DB 클러스터와 DB 인스턴스의 이름을 블루 환경의 대응하는 DB 클러스터 및 DB 인스턴스와 일치하도록 변경합니다. 예를 들어 블루 환경의 DB 인스턴스 이름이 `mydb`라고 가정하겠습니다. 또한 그린 환경의 대응하는 DB 인스턴스의 이름은 `mydb-green-abc123`이라고 가정하겠습니다. 전환 중에는 그린 환경의 DB 인스턴스 이름이 `mydb`로 변경됩니다.

RDS는 블루 환경의 DB 클러스터 및 DB 인스턴스의 현재 이름에 `-old`*n*을 추가하여 이름을 변경합니다(*n*은 숫자입니다). 예를 들어 블루 환경의 DB 인스턴스 이름이 `mydb`라고 가정하겠습니다. 전환이 끝나면 DB 인스턴스의 이름은 `mydb-old1`이 됩니다.

또한 RDS는 블루 환경의 대응하는 엔드포인트와 일치하도록 그린 환경의 엔드포인트 이름을 변경하므로, 애플리케이션은 변경하지 않아도 됩니다.

6. 두 환경 모두에서 데이터베이스 연결을 허용합니다.
7. 새 프로덕션 환경에서 DB 클러스터에 대한 쓰기 작업을 허용합니다.

   전환이 끝나면 이전 프로덕션 DB 클러스터는 읽기 작업만 허용합니다. DB 클러스터에서 쓰기를 활성화하더라도 블루/그린 배포를 삭제하기 전까지는 읽기 전용으로 유지됩니다.

Amazon EventBridge를 사용하여 전환 상태를 모니터링할 수 있습니다. 자세한 내용은 the section called “블루/그린 배포 이벤트” 단원을 참조하십시오.

블루 환경에서 태그를 구성했다면 이러한 태그는 전환 중에 새 프로덕션 환경으로 복사됩니다. 태그에 대한 자세한 내용은 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 섹션을 참조하세요.

어떤 이유로든 전환이 시작된 후 종료되기 전에 중지되면, 모든 변경 사항이 롤백되고 두 환경 모두 변경되지 않습니다.

전환 모범 사례

전환하기 전에 다음 작업을 완료하여 모범 사례를 준수하는 것이 좋습니다.

- 그린 환경에서 리소스를 철저히 테스트합니다. 정상적이고 효율적으로 작동하는지 확인합니다.
- 관련 Amazon CloudWatch 지표를 모니터링합니다. 자세한 내용은 the section called “전환 전 CloudWatch 지표 확인” 단원을 참조하십시오.
- 전환에 가장 적합한 시간을 확인합니다.

  전환 중에는 두 환경 모두의 데이터베이스에서 쓰기가 차단됩니다. 프로덕션 환경에서 트래픽이 가장 적은 시간을 확인합니다. 활성 DDL 같이 오래 실행되는 트랜잭션은 전환 시간이 길며, 그 결과 프로덕션 워크로드의 다운타임이 길어질 수 있습니다.

  DB 클러스터에 다수의 연결이 있는 경우 블루/그린 배포로 전환하기 전에 애플리케이션에 필요한 최소한의 개수로 연결을 수동으로 줄이는 것이 좋습니다. 이를 위한 한 가지 방법은 블루/그린 배포의 상태를 모니터링하고 상태가 `SWITCHOVER_IN_PROGRESS`로 변경되었음을 감지하면 연결 정리를 시작하는 스크립트를 만드는 것입니다.

- 두 환경 모두에서 DB 클러스터 및 DB 인스턴스가 `Available` 상태인지 확인합니다.
- 그린 환경의 DB 클러스터가 정상 상태이고 복제 중인지 확인합니다.
- 네트워크 및 클라이언트 구성으로 인해 DNS 캐시 TTL(Time-to-Live)이 Aurora DNS 영역의 기본값인 5초 이상으로 늘어나지 않도록 해야 합니다.

  그렇지 않으면 애플리케이션은 전환 후에도 계속해서 쓰기 트래픽을 블루 환경으로 전송합니다.
- Aurora PostgreSQL 블루/그린 배포의 경우 다음을 수행합니다.
  - 논리적 복제 제한을 검토하고 전환하기 전에 필요한 조치를 취합니다. 자세한 내용은 the section called "블루/그린 배포의 논리적 복제 관련 제한 사항" 단원을 참조하십시오.
  - `ANALYZE` 작업을 실행하여 `pg_statistics` 테이블을 새로 고칩니다. 이렇게 하면 전환 후 성능 문제가 발생할 위험이 줄어듭니다.

> **Note**
>
> 전환 중에는 전환에 포함된 DB 클러스터를 수정할 수 없습니다.

전환 전 CloudWatch 지표 확인

블루/그린 배포를 전환하기 전에 Amazon CloudWatch 내에서 다음 지표를 확인하는 것이 좋습니다.

- `DatabaseConnections` - 이 지표를 사용하여 블루/그린 배포의 작업 수준을 예측하고 전환하기 전에 값이 배포에 적합한 수준인지 확인합니다. 성능 개선 도우미가 활성화되어 있으면 `DBLoad`가 더 정확한 지표입니다.
- `ActiveTransactions` - DB 인스턴스의 DB 파라미터 그룹에서 `innodb_monitor_enable`이 `all`로 설정된 경우 이 지표를 사용하여 전환을 차단할 수 있는 활성 트랜잭션이 많은지 확인합니다.

자세한 내용은 the section called "Aurora의 CloudWatch 지표" 단원을 참조하십시오.

전환 전 복제 지연 모니터링

블루/그린 배포를 전환하기 전에 가동 중지 시간을 줄이기 위해 복제 지연이 0에 가까운지 확인합니다.

Aurora MySQL

MySQL 블루/그린 배포의 경우 그린 환경에서 `AuroraBinlogReplicaLag` CloudWatch 지표를 확인하여 현재 복제본 지연을 식별합니다. 자세한 내용은 the section called "읽기 전용 복제본 간 지연 문제 진단 및 해결" 단원을 참조하십시오.

Aurora PostgreSQL

PostgreSQL 블루/그린 배포의 경우 블루 환경에서 `OldestReplicationSlotLag` CloudWatch 지표를 확인하여 현재 복제본 지연을 식별합니다. 자세한 내용은 the section called “Amazon Aurora에 대한 인스턴스 수준 지표” 단원을 참조하십시오.

또한 블루 환경에서 다음 SQL 쿼리를 실행할 수 있습니다.

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed     | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

`confirmed_flush_lsn`은 복제본으로 전송된 마지막 로그 시퀀스 번호(LSN)을 나타냅니다. `pg_current_wal_lsn`은 데이터베이스의 현재 위치를 나타냅니다. `lsn_distance`가 0이라는 것은 최신 복제본까지 처리되었음을 의미합니다.

블루/그린 배포로의 전환

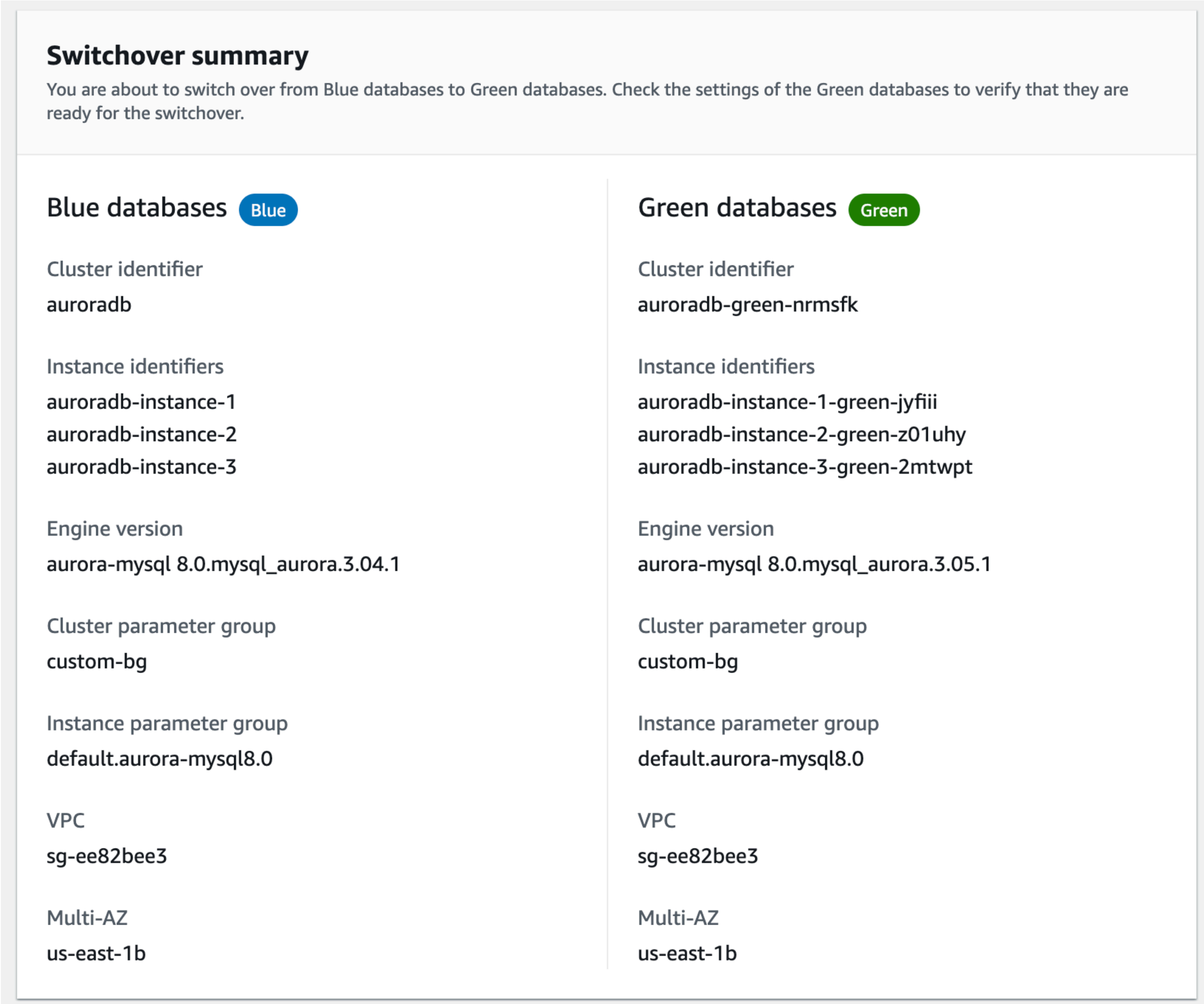
AWS Management Console, AWS CLI 또는 RDS API를 사용하여 블루/그린 배포로 전환할 수 있습니다.

콘솔

블루/그린 배포로 전환하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 후 전환할 블루/그린 배포를 선택합니다.
3. Actions(작업)에서 Switch over(전환)을 선택합니다.

   Switch over(전환) 페이지가 표시됩니다.



4. Switch over(전환) 페이지에서 전환 요약을 확인합니다. 두 환경의 리소스가 예상과 일치하는지 확인합니다. 일치하지 않는다면 Cancel(취소)를 선택합니다.

5. 제한 시간 설정에 전환 제한 시간을 입력합니다.

6. 클러스터에서 Aurora PostgreSQL을 실행하는 경우 전환 전 권장 사항을 검토하고 확인하세요. 자세한 내용은 the section called “블루/그린 배포의 논리적 복제 관련 제한 사항” 단원을 참조하십시오.

7. Switch over(전환)을 선택합니다.

AWS CLI

AWS CLI를 사용하여 블루/그린 배포로 전환하려면 다음 옵션을 지정한 [switchover-blue-green-deployment](#) 명령을 사용해야 합니다.

- `--blue-green-deployment-identifier` - 블루/그린 배포의 리소스 ID를 지정합니다.
- `--switchover-timeout` - 전환의 제한 시간을 초 단위로 지정합니다. 기본값은 300입니다.

Example 블루/그린 배포로 전환

대상 LinuxmacOS, 또는Unix:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

Windows의 경우:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS API

Amazon RDS API를 사용하여 블루/그린 배포로 전환하려면 다음 파라미터를 적용한 [`SwitchoverBlueGreenDeployment`](#) 작업을 사용해야 합니다.

- `BlueGreenDeploymentIdentifier` - 블루/그린 배포의 리소스 ID를 지정합니다.
- `SwitchoverTimeout` - 전환의 제한 시간을 초 단위로 지정합니다. 기본값은 300입니다.

전환 후

전환한 후에는 이전 블루 환경의 DB 클러스터 및 DB 인스턴스가 유지됩니다. 이러한 리소스에는 표준 요금이 적용됩니다. 블루와 그린 환경 간의 복제 및 바이너리 로깅이 중지됩니다.

RDS는 블루 환경의 DB 클러스터 및 DB 인스턴스의 현재 이름에 `-old`*n*을 추가하여 이름을 변경합니다(*n*은 숫자입니다). 블루 DB 클러스터는 강제로 읽기 전용 상태가 됩니다. 쓰기 작업을 활성화하더라도 블루/그린 배포를 삭제하기 전까지는 읽기 전용으로 유지됩니다. RDS는 그린 환경 `-new`*n*에서 DB 클러스터 및 DB 인스턴스의 이름을 지정합니다.

블루/그린 배포 리소스를 삭제하면 RDS는 `-old`*n* 및 `-new`*n* 리소스를 유지합니다.

DB identifier	Role	Engine
auroradb-old1 Old Blue	Regional cluster	Aurora MySQL
auroradb-instance-1-old1 Old Blue	Writer instance	Aurora MySQL
auroradb-instance-2-old1 Old Blue	Reader instance	Aurora MySQL
auroradb-instance-3-old1 Old Blue	Reader instance	Aurora MySQL
aurora-blue-green-deployment	Blue/Green Deployment	-
auroradb New Blue	Regional cluster	Aurora MySQL
auroradb-instance-1 New Blue	Writer instance	Aurora MySQL
auroradb-instance-2 New Blue	Reader instance	Aurora MySQL
auroradb-instance-3 New Blue	Reader instance	Aurora MySQL

소비자를 위한 상위 노드 업데이트

RDS는 완전 관리형 읽기 전용 복제본을 제공합니다. 그러나 외부 복제본이라고도 하는 자체 관리형 복제본을 설정하는 옵션도 제공합니다. 외부 복제본을 사용하면 타사 리소스를 복제 대상으로 사용할 수 있습니다.

Aurora MySQL 블루/그린 배포로 전환한 후, 블루 DB 클러스터에 전환 이전의 외부 복제본 또는 이진 로그 소비자가 있었던 경우, 전환 후에 상위 노드를 업데이트해야 복제 연속성을 유지할 수 있습니다.

상위 노드를 업데이트하려면

1. 전환 후 이전에 그린 환경에 있었던 라이터 DB 인스턴스는 마스터 로그 파일 이름 및 마스터 로그 위치가 포함된 이벤트를 내보냅니다. 이벤트를 찾으려면 RDS 콘솔로 이동하여 왼쪽 탐색 창에서 이벤트를 선택합니다.
2. 전환하기 전에 소스가 이전 그린 라이터 DB 인스턴스의 이름인 이벤트를 기준으로 필터링합니다.
3. 이진 로그 좌표가 포함된 이벤트를 찾습니다. 이벤트 메시지는 `Binary log coordinates in green environment after switchover: file mysql-bin-changelog.`*`000003`* `and position` *`40134574`*와 유사합니다.

4. 소비자 또는 복제본이 이전 블루 환경의 모든 이진 로그를 적용했는지 확인하세요. 그런 다음 제공된 이진 로그 좌표를 사용하여 소비자에 대한 복제를 재개하세요. 예를 들어, EC2에서 MySQL 복제본을 실행하는 경우 다음 명령을 사용할 수 있습니다.

   MySQL 8.0.22 이하 메이저 및 마이너 버전

   ```
   CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-changelog.000003', MASTER_LOG_POS=40134574;
   ```

   MySQL 8.0.23 이상 메이저 및 마이너 버전

   ```
   CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
    SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
   ```

Amazon Aurora에서 블루/그린 배포 삭제

블루/그린 배포는 전환 전이나 후에 삭제할 수 있습니다.

전환하기 전에 블루/그린 배포를 삭제하면, Amazon RDS는 배포를 그린 환경에서 선택적으로 DB 클러스터를 삭제합니다.

- 그린 환경(`--delete-target`)에서 DB 클러스터를 삭제하기로 한 경우, 클러스터에는 삭제 방지가 꺼져 있어야 합니다.
- 그린 환경(`--no-delete-target`)에서 DB 클러스터를 삭제하지 않으면 클러스터는 그대로 유지되지만, 더 이상 블루/그린 배포에 속하지 않게 됩니다. Aurora MySQL의 경우 환경 간에 복제가 계속됩니다. Aurora PostgreSQL의 경우 그린 환경이 독립 실행형 환경으로 승격되므로 복제가 중지됩니다.

녹색 데이터베이스를 삭제하는 옵션은 전환 후에는 콘솔에서 사용할 수 없습니다. AWS CLI를 사용하여 블루/그린 배포를 삭제할 때 배포 상태가 `SWITCHOVER_COMPLETED`인 경우 `--delete-target` 옵션을 지정할 수 없습니다.

> **Important**
>
> 블루/그린 배포를 삭제하면 RDS는 이전 프로덕션 DB 클러스터에서 읽기 전용 보호를 제거합니다. DB 클러스터에 대해 `read_only` 파라미터를 비활성화하면 쓰기 작업이 다시 허용되기 시작합니다.

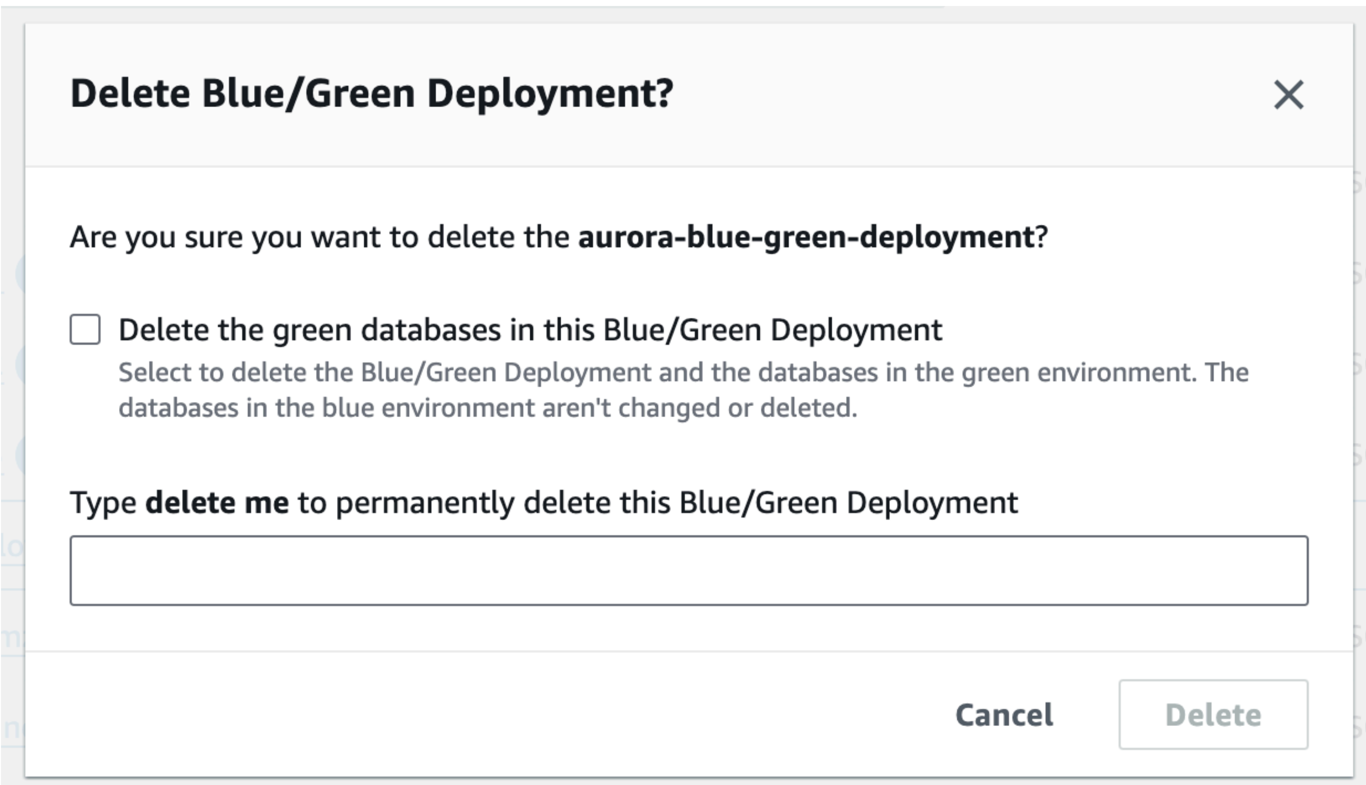
AWS Management Console, AWS CLI 또는 RDS API를 사용하여 블루/그린 배포를 삭제할 수 있습니다.

콘솔

블루/그린 배포를 삭제하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 후 삭제하려는 블루/그린 배포를 선택합니다.
3. 작업에 대해 삭제를 선택합니다.

   블루/그린 배포를 삭제하시겠습니까? 창이 나타납니다.

   그린 데이터베이스를 삭제하려면 이 블루/그린 배포에서 그린 데이터베이스 삭제를 선택합니다.
4. 상자에 **`delete me`**를 입력합니다.
5. 삭제를 선택합니다.

AWS CLI

AWS CLI를 사용하여 블루/그린 배포를 삭제하려면 다음 옵션을 적용한 delete-blue-green-deployment 명령을 사용해야 합니다.

- `--blue-green-deployment-identifier` - 삭제할 블루/그린 배포의 리소스 ID입니다.
- `--delete-target` – 그린 환경의 DB 클러스터가 삭제된다고 지정합니다. 블루/그린 배포가 `SWITCHOVER_COMPLETED` 상태인 경우 이 옵션을 지정할 수 없습니다.
- `--no-delete-target` – 그린 환경의 DB 클러스터가 유지된다고 지정합니다.

Example 그린 환경에서 블루/그린 배포 및 DB 클러스터를 삭제합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

Windows의 경우:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example 그린 환경에서 블루/그린 배포를 삭제하지만 DB 클러스터는 유지합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

Windows의 경우:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS API

Amazon RDS API를 사용하여 블루/그린 배포를 삭제하려면 다음 파라미터를 적용한 `DeleteBlueGreenDeployment` 작업을 사용해야 합니다.

- `BlueGreenDeploymentIdentifier` - 삭제할 블루/그린 배포의 리소스 ID입니다.
- `DeleteTarget` – `TRUE`로 지정하여 그린 환경의 DB 클러스터를 삭제하거나 `FALSE`로 설정하여 유지합니다. 블루/그린 배포 상태가 `SWITCHOVER_COMPLETED`인 경우 `TRUE`일 수 없습니다.

Amazon Aurora DB 클러스터 백업 및 복구

이 주제에서는 Amazon Aurora DB 클러스터 백업 및 복원에 대한 정보를 제공합니다.

> **Tip**
>
> Aurora 고가용성 기능과 자동 백업 기능을 사용하면 광범위한 설정 없이도 데이터를 안전하게 보호할 수 있습니다. 백업 전략을 구현하기 전에 Aurora에서 여러 데이터 복사본을 유지 관리하고 여러 DB 인스턴스와 AWS 리전에서 데이터에 액세스하도록 지원하는 방법에 대해 알아보세요. 자세한 내용은 Amazon Aurora의 고가용성 단원을 참조하세요.

주제

Aurora DB 클러스터 백업 및 복원에 대한 개요

다음 주제에서는 Aurora 백업과 Aurora DB 클러스터를 복원하는 방법에 대해 설명합니다.

목차

백업

Aurora는 클러스터 볼륨을 자동으로 백업한 후 백업 보관 기간 동안 복원 데이터를 보관합니다. Aurora 자동 백업은 연속적 및 증분식으로 이루어지기 때문에 백업 보관 기간 내에 어떤 시점으로든 신속하게 복구가 가능합니다. 백업 데이터를 쓰는 중에도 성능에 미치는 영향이나 데이터베이스 서비스 중단은 일어나지 않습니다. 백업 보관 기간은 DB 클러스터를 생성 또는 수정할 때 1일에서 35일까지 지정할 수 있습니다. Aurora 자동 백업은 Amazon S3에 저장됩니다. 자동 백업 보존에 대한 자세한 내용은 자동 백업 보존 섹션을 참조하세요.

백업 보관 기간을 넘겨서 데이터를 유지하고 싶을 때는 클러스터 볼륨의 데이터 스냅샷을 캡처할 수 있습니다. Aurora DB 클러스터 스냅샷은 만료되지 않습니다. 새로운 DB 클러스터를 스냅샷에서 생성할 수 있기 때문입니다. 자세한 내용은 DB 클러스터 스냅샷 생성 단원을 참조하십시오.

> Note
>
> - Amazon Aurora DB 클러스터의 경우, 기본 백업 보관 기간은 DB 클러스터 생성 방법과 상관없이 1일이 됩니다.
> - Aurora에서 자동 백업을 비활성화할 수 없습니다. Aurora에 대한 백업 보관 기간은 DB 클러스터에서 관리합니다.

백업 스토리지의 비용은 유지하는 Aurora 백업 및 스냅샷 데이터의 양과 유지 기간에 따라 달라집니다. Aurora 백업 및 스냅샷과 연결된 스토리지에 대한 자세한 내용은 Amazon Aurora 백업 스토리지 사

용량 파악 단원을 참조하십시오. Aurora 백업 스토리지에 대한 요금 정보는 Amazon RDS for Aurora RDS 요금을 참조하십시오. 스냅샷과 연결된 Aurora 클러스터가 삭제된 후 해당 스냅샷을 저장하면 Aurora에 대해 표준 백업 스토리지 요금이 발생합니다.

AWS Backup 사용하기

AWS Backup을 사용해 Amazon Aurora DB 클러스터의 백업 데이터를 관리할 수 있습니다.

AWS Backup에서 관리하는 스냅샷은 수동 DB 클러스터 스냅샷으로 간주되지만 Aurora에 대한 DB 클러스터 스냅샷 할당량에 포함되지는 않습니다. AWS Backup에서 생성된 스냅샷의 이름은 `awsbackup:job-`*`AWS-Backup-job-number`*입니다. AWS Backup에 대한 자세한 내용은 AWS 백업 개발자 안내서를 참조하세요.

또한 AWS Backup을 사용해 Amazon Aurora DB 클러스터의 자동 백업을 관리할 수 있습니다. DB 클러스터가 AWS Backup의 백업 계획에 연결되어 있는 경우, 시점 복구에 해당 백업 계획을 사용할 수 있습니다. AWS Backup에서 관리하는 자동(연속) 백업의 이름은 `continuous:cluster-`*`AWS-Backup-job-number`*입니다. 자세한 내용은 AWS Backup을 사용해 지정된 시간으로 DB 클러스터 복원 단원을 참조하십시오.

백업 기간

자동 백업은 기본 백업 기간 동안 매일 실행됩니다. 백업 시간이 백업 기간에 할당된 시간보다 오래 걸릴 경우 백업은 백업 기간이 종료한 후에도 완료 시까지 계속 실행됩니다. 백업 기간은 해당 DB 클러스터에 대한 주간 유지 보수 기간과 겹칠 수 없습니다.

Aurora 자동 백업은 연속 및 증분 백업이지만 백업 기간은 백업 유지 기간 내에 보존되는 일일 시스템 백업을 만드는 데 사용됩니다. 백업을 복사하여 유지 기간이 지난 뒤에도 보존할 수 있습니다.

> **Note**
>
> AWS Management Console를 사용하여 DB 클러스터를 생성하는 경우 백업 윈도우를 지정할 수 없습니다. 그러나 AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 생성할 때, 백업 기간을 지정할 수 있습니다.

클러스터를 생성할 때 원하는 백업 기간을 지정하지 않으면 Aurora이 기본 30분 백업 기간을 할당합니다. 이 기간은 각 AWS 리전에 대해 8시간의 시간 블록 중에서 임의로 선택됩니다. 다음 테이블은 기본 백업 기간이 할당된 각 AWS 리전별 시간 블록 목록입니다.

리전 이름	리전	시간 블록
미국 동부(버지니아 북부)	us-east-1	03:00~11:00 UTC
미국 동부(오하이오)	us-east-2	03:00~11:00 UTC
미국 서부(캘리포니아 북부 지역)	us-west-1	06:00~14:00 UTC
미국 서부(오리건)	us-west-2	06:00~14:00 UTC
Africa (Cape Town)	af-south-1	03:00~11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00~14:00 UTC
아시아 태평양(하이데라바드)	ap-south-2	06:30~14:30 UTC
아시아 태평양(자카르타)	ap-southeast-3	08:00~16:00 UTC
아시아 태평양(말레이시아)	ap-southeast-5	09:00~17:00 UTC
아시아 태평양(멜버른)	ap-southeast-4	11:00~19:00 UTC
아시아 태평양(뭄바이)	ap-south-1	16:30~00:30 UTC
Asia Pacific (Osaka)	ap-northeast-3	00:00~08:00 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00~21:00 UTC
아시아 태평양(싱가포르)	ap-southeast-1	14:00~22:00 UTC

리전 이름	리전	시간 블록
아시아 태평양(시드니)	ap-southeast-2	12:00~20:00 UTC
아시아 태평양(도쿄)	ap-northeast-1	13:00~21:00 UTC
Canada (Central)	ca-central-1	03:00~11:00 UTC
캐나다 서부(캘거리)	ca-west-1	18:00~02:00 UTC
중국(베이징)	cn-north-1	06:00~14:00 UTC
China (Ningxia)	cn-northwest-1	06:00~14:00 UTC
Europe (Frankfurt)	eu-central-1	20:00~04:00 UTC
유럽(아일랜드)	eu-west-1	22:00~06:00 UTC
Europe (London)	eu-west-2	22:00~06:00 UTC
유럽(밀라노)	eu-south-1	02:00~10:00 UTC
유럽(파리)	eu-west-3	07:29~14:29 UTC
유럽(스페인)	eu-south-2	02:00~10:00 UTC
Europe (Stockholm)	eu-north-1	23:00~07:00 UTC
유럽(취리히)	eu-central-2	02:00~10:00 UTC
이스라엘(텔아비브)	il-central-1	03:00~11:00 UTC
중동(바레인)	me-south-1	06:00~14:00 UTC
중동(UAE)	me-central-1	05:00~13:00 UTC
남아메리카(상파울루)	sa-east-1	23:00~07:00 UTC
AWS GovCloud(미국 동부)	us-gov-east-1	17:00~01:00 UTC

리전 이름	리전	시간 블록
AWS GovCloud(미국 서부)	us-gov-west-1	06:00~14:00 UTC

데이터 복구

Aurora에서 보존되는 백업 데이터, 이전에 저장한 DB 클러스터 스냅샷 또는 보존된 자동 백업에서 새 Aurora DB 클러스터를 생성하여 데이터를 복구할 수 있습니다. 백업 데이터에서 생성된 DB 클러스터의 새 사본을 백업 보관 기간 중 임의 시점으로 빨리 복구할 수 있습니다. 백업 보존 기간 중 Aurora 백업은 연속 및 증분 방식으로 백업을 수행하므로 복원 시간을 줄이기 위해 데이터 스냅샷을 자주 캡처할 필요가 없습니다.

DB 클러스터의 최근 복원 가능 시간은 DB 클러스터를 복원할 수 있는 가장 최근 시점을 나타냅니다. 일반적으로 활성 DB 클러스터의 경우 현재 시간으로부터 5분 이내, 보존된 자동 백업의 경우 클러스터 삭제 시간으로부터 5분 이내입니다.

가장 빠른 복원 가능 시간은 백업 보존 기간 내에서 클러스터 볼륨을 복원할 수 있는 가장 오래된 시점입니다.

DB 클러스터의 최근 또는 가장 빠른 복구 시간을 알아보려면 RDS 콘솔에서 `Latest restorable time` 또는 `Earliest restorable time` 값을 확인합니다. 이러한 값을 보는 방법은 [보존된 백업 보기](#) 단원을 참조하십시오.

DB 클러스터의 복구가 언제 완료되었는지는 `Latest restorable time` 및 `Earliest restorable time` 값을 사용하여 확인할 수 있습니다. 이 값은 복원 작업이 완료되기 전에는 NULL을 반환합니다. `Latest restorable time` 또는 `Earliest restorable time`이 NULL을 반환하면 백업 또는 복원 작업을 요청할 수 없습니다.

DB 클러스터를 특정 시점으로 복원에 대한 자세한 내용은 [지정된 시간으로 DB 클러스터 복원](#) 단원을 참조하십시오.

Aurora에 대한 데이터베이스 복제

DB 클러스터 스냅샷을 새 DB 클러스터로 복원하는 대신, 데이터베이스 복제를 이용해 Aurora DB 클러스터의 데이터베이스를 복제할 수도 있습니다. 복제 데이터베이스는 최초 생성 시 최소한의 추가 공간만 사용합니다. 데이터는 데이터가 변경된 경우에만 원본 데이터베이스 또는 복제 데이터베이스에 복사됩니다. 동일한 DB 클러스터에 대해 여러 복제본을 생성할 수 있고, 다른 복제본에서 추가 복제본을 추가할 수도 있습니다. 자세한 내용은 [Aurora DB 클러스터에 대한 볼륨 복제](#) 단원을 참조하십시오.

역추적

이제 Aurora MySQL은 백업에서 데이터를 복구하지 않고도 특정 시간으로 DB 클러스터 "되감기"를 지원합니다. 자세한 내용은 Aurora DB 클러스터 역추적 섹션을 참조하세요.

자동 백업 보존

프로비저닝된 DB 클러스터 또는 Aurora Serverless v2 DB 클러스터를 삭제할 때 자동 백업을 보존할 수 있습니다. 이렇게 하면 클러스터를 삭제한 후에도 DB 클러스터를 백업 보존 기간 내의 특정 시점으로 복원할 수 있습니다.

보존된 자동 백업에는 DB 클러스터의 시스템 스냅샷 및 트랜잭션 로그가 포함되어 있습니다. 또한 활성 클러스터로 복원하는 데 필요한 DB 인스턴스 클래스와 같은 DB 클러스터 속성도 포함됩니다.

AWS Management Console, RDS API 및 AWS CLI를 사용하여 보존된 자동 백업을 복원하거나 제거할 수 있습니다.

> **Note**
>
> Aurora Serverless v1 DB 클러스터의 자동 백업은 유지할 수 없습니다.

주제

보존 기간

보존된 자동 백업의 시스템 스냅샷 및 트랜잭션 로그는 원본 DB 클러스터에 대해 만료될 때와 동일한 방식으로 만료됩니다. 원본 클러스터의 보존 기간 설정은 자동 백업에도 적용됩니다. 이 클러스터에 대해 새 스냅샷이나 로그가 생성되지 않았으므로 보존된 자동 백업은 결국 완전히 만료됩니다. 보존 기간이 끝난 후에도 수동 DB 클러스터 스냅샷은 계속 보존되지만 자동 백업은 모두 만료됩니다.

콘솔이나 AWS CLI 또는 RDS API를 사용하여 보존된 자동 백업을 제거할 수 있습니다. 자세한 내용은 보관된 자동 백업 삭제 단원을 참조하십시오.

보존된 자동 백업과 달리 최종 스냅샷은 만료되지 않습니다. 보존된 자동 백업도 결국 만료되므로 자동 백업을 보존하더라도 가급적이면 최종 스냅샷을 생성하는 것이 좋습니다

보존된 백업 보기

보존된 자동 백업을 보려면 RDS 콘솔의 탐색 창에서 자동 백업을 선택한 다음 보존됨을 선택합니다. 보존된 자동 백업에 연결된 개별 스냅샷을 보려면 탐색 창에서 [스냅샷(Snapshots)]을 선택합니다. 또는 보존된 자동 백업과 연결된 개별 스냅샷을 설명할 수 있습니다. 그런 다음 이러한 스냅샷 중 하나에서 DB 인스턴스를 직접 복원할 수 있습니다.

AWS CLI를 사용하여 보존된 자동 백업을 설명하려면 다음 명령 중 하나를 사용합니다.

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

RDS API를 사용하여 보존된 자동 백업을 설명하려면 `DbClusterResourceId` 파라미터로 `DescribeDBClusterAutomatedBackups` 작업을 호출합니다.

보존 비용

각 Aurora DB 클러스터에 대한 총 Aurora 데이터베이스 스토리지의 최대 100%까지 백업 스토리지에 대한 추가 비용은 없습니다. 또한 DB 클러스터를 삭제한 후 자동 백업을 보관하는 경우 최대 1일까지는 추가 요금이 부과되지 않습니다. 하루 이상 보존된 백업에는 요금이 부과됩니다.

트랜잭션 로그나 인스턴스 메타데이터에 대해 추가 요금이 부과되지 않습니다. 백업에 대한 기타 모든 요금 규칙이 복원 가능한 클러스터에 적용됩니다. 자세한 내용은 Amazon Aurora 요금 페이지를 참조하세요.

제한 사항

다음 제한은 보존된 자동 백업에 적용됩니다.

- 한 AWS 리전에서 보존된 자동 백업의 최대 개수는 40개입니다. DB 클러스터의 할당량에는 포함되지 않습니다. 최대 40개의 실행 중인 DB 클러스터, 40개의 실행 중인 DB 인스턴스 및 DB 클러스터에 대한 40개의 보존된 자동 백업을 동시에 보유할 수 있습니다.

  자세한 내용은 Amazon Aurora의 할당량 단원을 참조하십시오.
- 보존된 자동 백업에는 파라미터나 옵션 그룹에 대한 정보가 포함되지 않습니다.
- 삭제된 클러스터를 삭제할 때의 보존 기간 내 특정 시점으로 복원할 수 있습니다.
- 보존된 자동 백업은 원본 클러스터를 삭제할 때 존재했던 시스템 백업, 트랜잭션 로그 및 DB 클러스터 속성으로 구성되므로 수정할 수 없습니다.

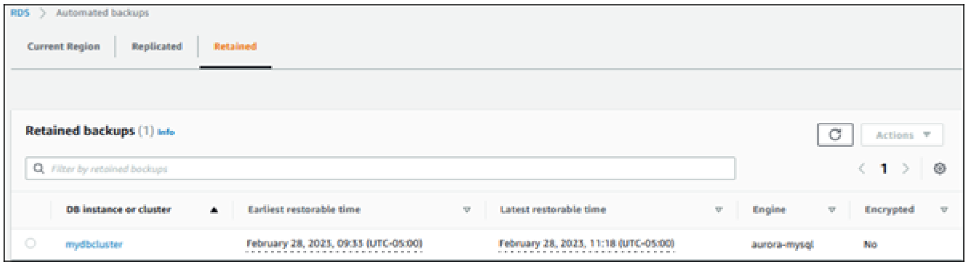
보관된 자동 백업 삭제

보관된 자동 백업이 더 이상 필요하지 않으면 삭제할 수 있습니다.

콘솔

보관된 자동 백업을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 자동 백업(Automated backups)을 선택합니다.
3. 보존됨 탭을 선택합니다.



4. 삭제하려는 보관된 자동 백업을 선택합니다.
5. [Actions]에 대해 [Delete]를 선택합니다.
6. 확인 페이지에서 **`delete me`**를 입력하고 삭제를 선택합니다.

AWS CLI

delete-db-cluster-automated-backup이라는 AWS CLI 명령을 다음 옵션과 함께 사용하여 보존된 자동 백업을 삭제할 수 있습니다.

- `--db-cluster-resource-id` – 원본 DB 클러스터의 리소스 식별자.

  describe-db-cluster-automated-backups라는 AWS CLI 명령을 실행하여 보존된 자동 백업의 원본 DB 클러스터에 대한 리소스 식별자를 찾을 수 있습니다.

Example

이 예시에서는 리소스 ID가 `cluster-123ABCEXAMPLE`인 원본 DB 클러스터의 보존된 자동 백업을 삭제합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-cluster-automated-backup \
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

Windows의 경우:

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

RDS API

DeleteDBClusterAutomatedBackup이라는 Amazon RDS API 작업을 다음 파라미터와 함께 사용하여 보존된 자동 백업을 삭제할 수 있습니다.

- `DbClusterResourceId` – 원본 DB 클러스터의 리소스 식별자.

  DescribeDBClusterAutomatedBackups라는 Amazon RDS API 작업을 사용하여 보존된 자동 백업의 원본 DB 인스턴스에 대한 리소스 식별자를 찾을 수 있습니다.

Amazon Aurora 백업 스토리지 사용량 파악

Amazon Aurora는 자동(연속) 백업과 스냅샷이라는 2가지 유형의 백업을 유지 관리합니다.

자동 백업 스토리지

클러스터의 자동(연속) 백업은 지정된 보존 기간 내의 모든 데이터베이스 변경 사항을 증분 저장하여 해당 보존 기간 내의 어느 시점으로든 복원할 수 있습니다. 보존 기간은 1~35일 사이일 수 있습니다. 자동 백업은 증분 백업이며, 보존 기간 내 원하는 시점으로 복원하는 데 필요한 스토리지의 양에 따라 요금이 청구됩니다.

Aurora는 무료 백업 사용량도 제공합니다. 이 무료 사용량은 최신 클러스터 볼륨 크기(`VolumeBytesUsed` Amazon CloudWatch 지표로 표시됨)와 동일합니다. 이 양은 계산된 자동 백업 사용량에서 차감됩니다. 또한 보존 기간이 1일에 불과한 자동 백업에 대해서는 요금이 부과되지 않습니다.

예를 들어 자동 백업의 보존 기간이 7일인데 클러스터를 4일 전의 상태로 복원하려고 합니다. Aurora는 자동 백업에 저장된 증분 데이터를 사용하여 정확히 4일 전의 클러스터 상태를 재생성합니다.

자동 백업은 보존 기간의 어느 시점으로든 클러스터를 복원할 수 있도록 필요한 모든 정보를 저장합니다. 즉, 새 정보 쓰기 또는 기존 정보 삭제를 포함하여 보존 기간 동안 이루어진 모든 변경 사항을 저장합니다. 변경이 많은 데이터베이스의 경우 시간이 지남에 따라 자동 백업 크기가 커집니다. 데이터베이스 변경이 중지되면 이전에 저장한 변경 사항의 보존 기간이 종료되므로, 자동 백업의 크기가 줄어들 것으로 예상할 수 있습니다.

자동 백업의 총 청구 사용량은 보존 기간 동안 누적된 클러스터 볼륨 크기를 초과하지 않습니다. 예를 들어 보존 기간이 7일이고 클러스터 볼륨이 매일 100GB인 경우 청구되는 자동 백업 사용량은 700GB(100GBx7)를 초과하지 않습니다.

스냅샷 스토리지

DB 클러스터 스냅샷은 항상 클러스터 볼륨을 만들 때 클러스터 볼륨의 크기를 캡처하는 전체 백업입니다. 스냅샷을 수동으로 만들든 AWS 백업 계획을 통해 만들든 Aurora는 스냅샷을 수동 스냅샷으로 취급합니다. Aurora는 자동 백업 보존 기간 내에 있는 모든 스냅샷에 대해 무제한 무료 스토리지를 제공합니다. 수동 스냅샷이 이 기간을 벗어나면 월별로 GB당 요금이 발생합니다. 자동 시스템 스냅샷은 복사하지 않는 한 무료로 유지됩니다. 자동 백업은 스냅샷 복사본을 다루지 않으므로 AWS는 항상 스냅샷 복사본에 대한 요금을 청구합니다.

Aurora 백업에 대한 일반적인 정보는 백업을 참조하세요. Aurora 백업 스토리지에 대한 요금 정보는 Amazon Aurora 요금 페이지를 참조하시기 바랍니다.

Aurora 백업 스토리지에 대한 Amazon CloudWatch 지표

CloudWatch 콘솔을 통해 Amazon CloudWatch 지표를 사용하여 Aurora 클러스터를 모니터링하고 보고서를 작성할 수 있습니다. 다음 CloudWatch 지표를 사용하여 Aurora 백업에서 사용하는 스토리지 양을 검토하고 모니터링할 수 있습니다. 이러한 지표는 각 Aurora DB 클러스터에 대해 독립적으로 계산됩니다.

- `BackupRetentionPeriodStorageUsed` - 현재 자동 백업을 저장하는 데 사용되는 백업 스토리지 양(바이트)을 나타냅니다.
  - 값은 클러스터 볼륨의 크기와 보존 기간 동안 DB 클러스터에 적용된 변경 사항(쓰기 및 업데이트) 수에 따라 달라집니다. 이는 자동 백업이 클러스터에 적용된 모든 증분 변경 사항을 저장해야 원하는 시점으로 복원할 수 있기 때문입니다.
  - 이 지표는 Aurora가 제공하는 백업 프리 티어 사용량을 차감하지 않습니다.
  - 이 지표는 해당 날짜에 기록된 자동 백업 사용량에 대한 일일 데이터 포인트를 1개 생성합니다.
- `SnapshotStorageUsed` - 자동 백업 보존 기간 이후에 수동 스냅샷을 저장하는 데 사용되는 백업 스토리지의 양(바이트)을 나타냅니다.
  - 값은 자동 백업의 보존 기간을 초과하여 보관하는 스냅샷 수와 각 스냅샷의 크기에 따라 달라집니다.
  - 각 스냅샷의 크기는 스냅샷을 생성할 당시의 클러스터 볼륨 크기입니다.
  - 스냅샷은 증분 백업이 아니라 전체 백업입니다.
  - 이 지표는 청구되는 각 스냅샷에 대해 일일 데이터 포인트 1개를 생성합니다. 일일 총 스냅샷 사용량을 검색하려면 1일 동안 이 지표의 합계를 구하면 됩니다.
- `TotalBackupStorageBilled` - 해당 클러스터에 대해 청구된 모든 백업 사용량 지표(바이트)를 나타냅니다.

  ```
  BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier
  ```

  - 이 지표는 `BackupRetentionPeriodStorageUsed` 값에서 Aurora가 제공하는 백업 프리 티어 사용량을 뺀 값에 대해 일일 데이터 포인트 1개를 생성합니다. 이 프리 티어는 가장 최근에 기록된 DB 클러스터 볼륨 크기와 동일합니다. 이 데이터 포인트는 자동 백업의 실제 청구 사용량을 나타냅니다.
  - 이 지표는 모든 `SnapshotStorageUsed` 값에 대해 개별 일일 데이터 포인트를 생성합니다.
  - 일일 청구된 총 백업 사용량을 검색하려면 1일 동안 이 지표의 합계를 구하면 됩니다. 청구된 모든 스냅샷 사용량과 청구된 자동 백업 사용량을 합산하여 청구된 총 백업 사용량을 산출합니다.

CloudWatch 지표를 사용하는 방법에 대한 자세한 내용은 Amazon RDS 콘솔의 Aurora 지표 가용성 섹션을 참조하세요.

백업 스토리지 사용량 계산

자동 백업의 사용량은 백업 보존 기간 내의 어느 시점으로든 복원할 수 있도록 저장해야 하는 모든 증분 레코드를 검토하여 계산됩니다. 이러한 변경 사항에는 쓰기 작업 수뿐만 아니라 데이터 수정의 크기와 범위도 포함됩니다. 각 작업 유형(INSERT, UPDATE, DELETE)은 특정 시점 복구를 위해 보존해야 하는 변경 레코드를 생성합니다. 따라서 두 데이터베이스의 쓰기 작업(IOPS) 수가 같을 수 있지만 각 트랜잭션에서 수정되는 데이터의 양에 따라 백업 스토리지 요구 사항이 크게 다를 수 있습니다.

예를 들어 보존 기간이 7일인 자동 백업이 있습니다. 보존 기간 직전의 클러스터 볼륨 크기는 Aurora가 저장해야 하는 최소 용량인 100GB였습니다. 그런 다음 다음 7일 동안 활동을 진행합니다. 여기서 증분 레코드 크기는 데이터베이스의 쓰기 및 업데이트에서 발생하는 변경 레코드를 저장하는 데 필요한 스토리지의 양입니다.

일	증분 레코드 크기(GB)
1	10
2	15
3	25
4	20
5	10
6	25
7	30
합계	135

이 데이터는 백업에 대해 계산된 자동 백업 사용량이 다음과 같음을 의미합니다.

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

이후 청구된 사용량에서 프리 티어 사용량을 차감합니다. 볼륨의 최신 크기가 200GB라고 가정해 보겠습니다.

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

FAQ

스냅샷 요금은 언제 청구되나요?

자동 백업의 보존 기간이 지난(이전) 수동 스냅샷에 대해 요금이 청구됩니다.

수동 스냅샷이란 무엇인가요?

수동 스냅샷은 다음 조건 중 하나가 적용되는 스냅샷입니다.

- 사용자가 직접 요청함
- AWS Backup과 같은 자동 백업 서비스에서 생성함
- 자동 시스템 스냅샷에서 복사하여 보존 기간 이후에도 보존됨

DB 클러스터를 삭제하면 수동 스냅샷은 어떻게 되나요?

수동 스냅샷은 사용자가 삭제할 때까지 만료되지 않습니다.

DB 클러스터를 삭제해도 이전에 생성한 수동 스냅샷은 계속 존재합니다. 이전에 이러한 스냅샷이 자동 백업 보존 기간 내에 있었기 때문에 요금이 청구되지 않았다면, 더 이상 무료로 사용할 수 없으며 모든 사용량에 대해 전체 용량으로 요금이 청구되기 시작합니다.

백업 스토리지 비용을 줄이려면 어떻게 해야 하나요?

백업 사용 관련 비용을 줄일 수 있는 몇 가지 방법이 있습니다.

- 자동 백업의 보존 기간이 지난 수동 스냅샷을 삭제합니다. 여기에는 직접 생성한 스냅샷과 AWS Backup 계획에서 생성했을 수 있는 스냅샷이 포함됩니다. AWS Backup 계획을 살펴보고 보존 기간이 지난 스냅샷을 예상치 못하게 보관하지 있지 않은지 확인하세요.
- 데이터베이스에 대한 쓰기 및 업데이트를 평가하여 변경 횟수를 줄일 수 있는지 확인하세요. 자동 백업은 보존 기간 내의 모든 증분 변경 사항을 저장하므로, 업데이트 횟수를 줄이면 자동 백업 비용도 절감됩니다.
- 자동 백업의 보존 기간을 줄이는 것이 합리적인지 평가하세요. 보존 기간을 줄이면 백업에 더 적은 일수의 증분 데이터가 저장되어 전체 백업 비용이 줄어들 수 있습니다. 하지만 보존 기간을 줄이면 보존 기간이 지난 일부 스냅샷에 대해 요금이 청구되기 시작할 수도 있습니다. 이 조치가 적절한지 판단하기 전에 발생할 수 있는 추가 스냅샷 비용을 모두 확인해야 합니다.

백업 스토리지 비용은 어떻게 청구되나요?

스토리지는 GB/월별로 결제됩니다.

즉, 백업 스토리지 사용량은 해당 월의 사용량에 대한 가중 평균으로 요금이 청구됩니다. 다음은 한 달(30일)의 몇 가지 예입니다.

- 청구된 백업 사용량은 매월 30일 동안 매일 100GB입니다. 요금은 다음과 같습니다.

```
(100 GB * 30) / 30 = 100 GB-month
```

- 청구된 백업 사용량은 매월 첫 15일은 100GB이고, 마지막 15일은 0GB입니다. 요금은 다음과 같습니다.

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- 청구된 백업 사용량은 매월 첫 10일 동안은 50GB, 다음 10일 동안은 100GB, 마지막 10일 동안은 150GB입니다. 요금은 다음과 같습니다.

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

DB 클러스터의 역추적 설정은 백업 스토리지 사용량에 어떤 영향을 미치나요?

Aurora DB 클러스터의 역추적 설정은 해당 클러스터의 백업 데이터 볼륨에 영향을 미치지 않습니다. Amazon은 역추적 데이터용 스토리지에 별도로 요금을 부과합니다. Aurora 백업 스토리지에 대한 요금 정보는 Amazon Aurora 요금 페이지를 참조하세요.

스토리지 비용은 공유 스냅샷에 어떻게 적용되나요?

스냅샷을 다른 사용자와 공유하는 경우 여전히 스냅샷 소유자입니다. 스토리지 비용은 스냅샷 소유자에게 적용됩니다. 고객님이 소유한 공유 스냅샷을 삭제하면 아무도 이 스냅샷에 액세스할 수 없습니다.

다른 누군가가 소유한 공유 스냅샷에 대한 액세스를 유지하려면 해당 스냅샷을 복사하면 됩니다. 이렇게 하면 고객님이 새로운 스냅샷의 소유자가 됩니다. 복사한 스냅샷에 대한 모든 스토리지 비용은 고객님의 계정에 적용됩니다.

스냅샷 공유에 대한 자세한 내용은 DB 클러스터 스냅샷 공유 단원을 참조하세요. 스냅샷 복사에 대한 자세한 내용은 DB 클러스터 스냅샷 복사 단원을 참조하세요.

DB 클러스터 스냅샷 생성

Amazon RDS는 개별 데이터베이스가 아닌 전체 DB 클러스터를 백업하여 DB 클러스터의 스토리지 볼륨 스냅샷을 생성합니다. DB 클러스터 스냅샷을 생성할 때는 백업할 DB 클러스터를 구분한 다음 나중에 복구할 수 있도록 DB 클러스터 스냅샷에 이름을 지정해야 합니다. DB 클러스터 스냅샷을 생성하는 데 걸리는 시간은 데이터베이스 크기에 따라 다릅니다. 스냅샷에는 전체 스토리지 볼륨이 포함되기 때문에 임시 파일 같은 파일들의 크기가 스냅샷을 생성하는 데 걸리는 시간에 영향을 미치기도 합니다.

> **Note**
>
> DB 클러스터 스냅샷을 생성하려면 DB 인스턴스가 `available` 상태여야 합니다.

자동 백업과 달리 수동 스냅샷에는 백업 보존 기간이 적용되지 않습니다. 스냅샷이 만료되지 않습니다.

매우 장기간 백업하려면 스냅샷 데이터를 Amazon S3로 내보내는 것이 좋습니다. DB 엔진의 메이저 버전이 더 이상 지원되지 않는 경우에는 스냅샷에서 해당 버전으로 복원할 수 없습니다. 자세한 내용은 [Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기](#) 단원을 참조하십시오.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷을 생성할 수 있습니다.

콘솔

DB 클러스터 스냅샷을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.

   [수동 스냅샷(Manual snapshots)] 목록이 나타납니다.
3. [스냅샷 생성(Take Snapshot)]을 선택합니다.

   [DB 스냅샷 생성(Take DB Snapshot)] 창이 나타납니다.
4. 스냅샷 유형에서 DB 클러스터를 선택합니다.
5. 스냅샷을 생성할 DB 클러스터를 선택합니다.
6. 스냅샷 이름을 입력합니다.
7. [스냅샷 생성(Take Snapshot)]을 선택합니다.

수동 스냅샷 목록이 나타나고 새로운 DB 클러스터 스냅샷 상태가 `Creating`으로 표시됩니다. 스냅샷 상태가 `Available`이 되면 스냅샷 생성 시간을 볼 수 있습니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터 스냅샷을 생성할 때는 백업할 DB 클러스터를 구분한 다음 나중에 복구할 수 있도록 DB 클러스터 스냅샷에 이름을 지정해야 합니다. 이를 위해서는 AWS CLI [`create-db-cluster-snapshot`] 명령을 다음 파라미터와 함께 사용하면 됩니다.

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

이 예에서는 *`mydbcluster`*라고 하는 DB 클러스터에*`mydbclustersnapshot`*이라는 DB 클러스터 스냅샷을 생성합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Windows의 경우:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터 스냅샷을 생성할 때는 백업할 DB 클러스터를 구분한 다음 나중에 복구할 수 있도록 DB 클러스터 스냅샷에 이름을 지정해야 합니다. 이를 위해서는 Amazon RDS API [`CreateDBClusterSnapshot`] 명령을 다음 파라미터와 함께 사용하면 됩니다.

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

DB 클러스터 스냅샷의 사용 가능 여부 확인

AWS Management Console에서 클러스터에 대한 세부 정보 페이지에 있는 유지 관리 및 백업(Maintenance & backups) 탭의 스냅샷(Snapshots) 아래를 확인하거나, `describe-db-cluster-snapshots` CLI 명령을 사용하거나, `DescribeDBClusterSnapshots` API 작업을 사용하여 DB 클러스터 스냅샷 사용 가능 여부를 확인할 수 있습니다.

또한 `wait db-cluster-snapshot-available` CLI 명령을 사용하여 스냅샷이 사용 가능할 때까지 30초마다 API를 폴링할 수 있습니다.

DB 클러스터 스냅샷에서 복원

Amazon RDS는 개별 데이터베이스가 아닌 전체 DB 클러스터를 백업하여 DB 클러스터의 스토리지 볼륨 스냅샷을 생성합니다. DB 스냅샷에서 복원하여 새 DB 클러스터를 생성할 수 있습니다. 복원 원본으로 사용할 DB 클러스터 스냅샷의 이름을 입력한 다음 복원 작업에서 생성되는 새 DB 클러스터의 이름을 입력합니다. DB 클러스터 스냅샷에서 기존 DB 클러스터로 복원할 수 없습니다. 복원하면 새 DB 클러스터가 생성됩니다.

> **Important**
>
> 더 이상 사용되지 않는 DB 엔진 버전으로 스냅샷을 복원하려고 하면 최신 엔진 버전으로 즉시 업그레이드됩니다. 더 이상 사용되지 않는 엔진 버전에서 복원된 DB 클러스터는 최신 메이저 버전으로 업그레이드할 때까지 액세스할 수 없습니다.
> 또한 해당 버전에 추가 지원이 적용되거나 표준 지원 종료 시점에 도달한 경우 추가 지원 요금이 부과될 수 있습니다. 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 섹션을 참조하세요.

복원된 DB 클러스터는 `available` 상태가 되면 바로 사용할 수 있습니다.

DB 클러스터를 DB 클러스터 스냅샷에서 복원할 때 AWS CloudFormation을 사용할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::RDS::DBCluster를 참조하세요.

> **Note**
>
> 암호화되었거나 암호화되지 않은 수동 DB 클러스터 스냅샷을 공유하면 권한이 있는 AWS 계정에서 스냅샷의 복사본을 만든 후 복원하는 대신에 해당 스냅샷에서 DB 클러스터를 직접 복원할 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷 공유 섹션을 참조하세요.

RDS 추가 지원 버전을 사용하는 Aurora DB 클러스터 또는 글로벌 클러스터를 복원하는 방법에 대한 자세한 내용은 Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 복원 섹션을 참조하세요.

파라미터 그룹 고려 사항

복원된 DB 클러스터를 올바른 파라미터 그룹과 연결할 수 있도록 생성한 DB 클러스터 스냅샷에 대한 DB 파라미터 그룹과 DB 클러스터 파라미터 그룹을 유지하는 것이 좋습니다.

기본 DB 파라미터 그룹과 DB 클러스터 파라미터 그룹은 다른 그룹을 선택하지 않는 한 복원된 클러스터와 연결됩니다. 기본 파라미터 그룹에서는 사용자 정의 파라미터 설정을 사용할 수 없습니다.

DB 클러스터를 복원할 때 파라미터 그룹을 지정할 수 있습니다.

DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

보안 그룹 고려 사항

DB 클러스터를 복원할 때 다른 항목을 선택하지 않는 한 기본 Virtual Private Cloud(VPC), DB 서브넷 그룹 및 VPC 보안 그룹이 복원된 인스턴스와 연결됩니다.

- Amazon RDS 콘솔을 사용하고 있는 경우 사용자 정의 VPC 보안 그룹을 지정하여 클러스터와 연결하거나 새 VPC 보안 그룹을 생성할 수 있습니다.
- AWS CLI를 사용 중이라면 `restore-db-cluster-from-snapshot` 명령에 `--vpc-security-group-ids` 옵션을 포함하여 클러스터에 연결할 사용자 정의 VPC 보안 그룹을 지정할 수 있습니다.
- Amazon RDS API를 사용 중이라면 `VpcSecurityGroupIds.VpcSecurityGroupId.N` 작업에 `RestoreDBClusterFromSnapshot` 파라미터를 포함할 수 있습니다.

복원이 완료되고 새 DB 클러스터를 사용할 수 있게 되면 DB 클러스터를 수정하여 VPC 설정을 변경할 수도 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

Amazon Aurora 고려 사항

Aurora에서 DB 클러스터 스냅샷을 DB 클러스터로 복원합니다.

또한 Aurora MySQL과 Aurora PostgreSQL에서는 DB 클러스터 스냅샷을 Aurora Serverless DB 클러스터로 복원할 수 있습니다. 자세한 내용은 Aurora Serverless v1 DB 클러스터 복원 섹션을 참조하세요.

Aurora MySQL에서는 병렬 쿼리가 없는 클러스터에서 병렬 쿼리가 있는 클러스터로 DB 클러스터 스냅샷을 복원할 수 있습니다. 병렬 쿼리는 일반적으로 매우 큰 테이블에 사용되기 때문에 스냅샷 메커니즘은 큰 볼륨의 데이터를 Aurora MySQL 병렬 쿼리 지원 클러스터로 수집하는 가장 빠른 방법입니다. 자세한 내용은 Amazon Aurora MySQL용 병렬 쿼리 섹션을 참조하세요.

스냅샷에서 복원

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷에서 DB 클러스터를 복원할 수 있습니다.

콘솔

DB 클러스터 스냅샷에서 DB 클러스터를 복원하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 복원 원본으로 사용할 DB 클러스터 스냅샷을 선택합니다.
4. 작업에서 스냅샷 복원을 선택합니다.

   스냅샷 복원 페이지가 표시됩니다.
5. DB 클러스터를 복원하려는 DB 엔진 버전을 선택합니다.

   기본적으로 스냅샷은 소스 DB 클러스터와 동일한 DB 엔진 버전으로 복원됩니다(해당 버전이 사용 가능한 경우).
6. DB instance identifier(DB 인스턴스 식별자)에 복원된 DB 인스턴스의 이름을 입력합니다. Amazon RDS는 지정한 DB 인스턴스 식별자에서 DB 클러스터 식별자를 파생합니다.
7. DB 클러스터 스토리지 구성과 같은 다른 설정을 지정합니다.

   각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.
8. DB 클러스터 복원(Restore DB cluster)을 선택합니다.

AWS CLI

DB 클러스터 스냅샷에서 DB 클러스터를 복원하려면 AWS CLI 명령 restore-db-cluster-from-snapshot을 사용합니다.

이 예에서는 `mydbclustersnapshot`이라는 이전에 생성된 DB 클러스터 스냅샷에서 복원합니다. `mynewdbcluster`라는 새 DB 클러스터로 복원합니다.

DB 엔진 버전 등 다른 설정을 지정할 수 있습니다. 엔진 버전을 지정하지 않으면 DB 클러스터가 기본 엔진 버전으로 복원됩니다.

각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

DB 클러스터가 복원된 후 이전 DB 클러스터와 동일한 기능을 원할 경우 DB 클러스터 스냅샷을 생성하는 데 사용된 DB 클러스터가 사용하는 보안 그룹에 DB 클러스터를 추가해야 합니다.

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터를 위한 프라이머리 DB 인스턴스(라이터)를 생성합니다. AWS CLI를 사용하여 DB 클러스터를 복원할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들지 않아도 DB 클러스터 엔드포인트는 `creating` 상태를 유지합니다.
>
> create-db-instanceAWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하세요. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하십시오.

RDS API

DB 클러스터 스냅샷에서 DB 클러스터를 복원하려면 다음 파라미터와 함께 RDS API 작업 RestoreDBClusterFromSnapshot 을 호출합니다.

- `DBClusterIdentifier`
- `SnapshotIdentifier`

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터를 위한 프라이머리 DB 인스턴스(라이터)를 생성합니다. RDS API를 사용하여 DB 클러스터를 복원할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다. 프라이머리 DB 인스턴스를 만들지 않아도 DB 클러스터 엔드포인트는 `creating` 상태를 유지합니다.
>
> RDS API 작업 [CreateDBInstance](#)를 호출하여 DB 클러스터의 기본 인스턴스를 만듭니다. DB 클러스터의 이름을 `DBClusterIdentifier` 파라미터 값으로 포함합니다.

DB 클러스터 스냅샷 복사

Amazon Aurora에서는 자동 백업 또는 수동으로 DB 클러스터 스냅샷을 복사할 수 있습니다. 스냅샷을 복사한 후 사본은 수동 스냅샷입니다. 자동 백업 또는 수동 스냅샷의 복사본을 여러 개 만들 수 있지만 각 복사본에는 고유한 식별자가 있어야 합니다.

동일한 AWS 리전 내의 스냅샷을 복사할 수 있고, AWS 리전 간에 스냅샷을 복사할 수 있고, 공유 스냅샷을 복사할 수 있습니다.

리전과 계정 간에 DB 클러스터 스냅샷을 단일 단계로 복사할 수는 없습니다. 복사 작업마다 한 번씩 단계를 수행해야 합니다. 복사하는 대신 수동 스냅샷을 다른 AWS 계정과 공유할 수도 있습니다. 자세한 내용은 DB 클러스터 스냅샷 공유 섹션을 참조하세요.

> **Note**
>
> Amazon은 유지하는 Amazon Aurora 백업 및 스냅샷 데이터의 양과 유지 기간에 따라 요금을 청구합니다. Aurora 백업 및 스냅샷과 연결된 스토리지에 대한 자세한 내용은 Amazon Aurora 백업 스토리지 사용량 파악 단원을 참조하십시오. Aurora 스토리지에 대한 요금 정보는 Amazon RDS for Aurora 요금을 참조하세요.

DB 클러스터 스냅샷 복사에 대한 제한 및 고려 사항을 검토하세요. DB 클러스터 스냅샷을 복사하려면 다음 주제 중 하나를 참조하세요.

- AWS Management Console을 사용한 DB 클러스터 스냅샷 복사
- AWS CLI 또는 Amazon RDS API를 사용하여 암호화되지 않은 DB 클러스터 스냅샷 복사
- AWS CLI 또는 Amazon RDS API를 사용하여 암호화된 DB 클러스터 스냅샷 복사
- 계정 간에 DB 클러스터 스냅샷 복사

AWS Management Console을 사용한 DB 클러스터 스냅샷 복사

DB 클러스터 스냅샷을 복사하려면 이 항목의 절차를 사용합니다. 원본 데이터베이스 엔진이 Aurora인 경우 사용자의 스냅샷은 DB 클러스터 스냅샷입니다.

각 AWS 계정에 대해 AWS 리전 간에 DB 클러스터 스냅샷을 한 번에 5개까지 복사할 수 있습니다. 암호화된 DB 클러스터 스냅샷 및 암호화되지 않은 DB 클러스터 스냅샷 복사가 모두 지원됩니다. DB 클러스터 스냅샷을 다른 AWS 리전에 복사하면 해당 AWS 리전에 유지되는 수동 DB 클러스터 스냅샷이

생성됩니다. 소스 AWS 리전 외부로 DB 클러스터 스냅샷을 복사하면 Amazon RDS 데이터 전송 요금이 발생합니다.

데이터 전송 요금에 대한 자세한 내용은 Amazon RDS 요금을 참조하세요.

새 AWS 리전에 DB 클러스터 스냅샷 사본이 생성된 후 DB 클러스터 스냅샷 사본은 해당 AWS 리전의 다른 모든 DB 클러스터 스냅샷과 똑같이 동작합니다.

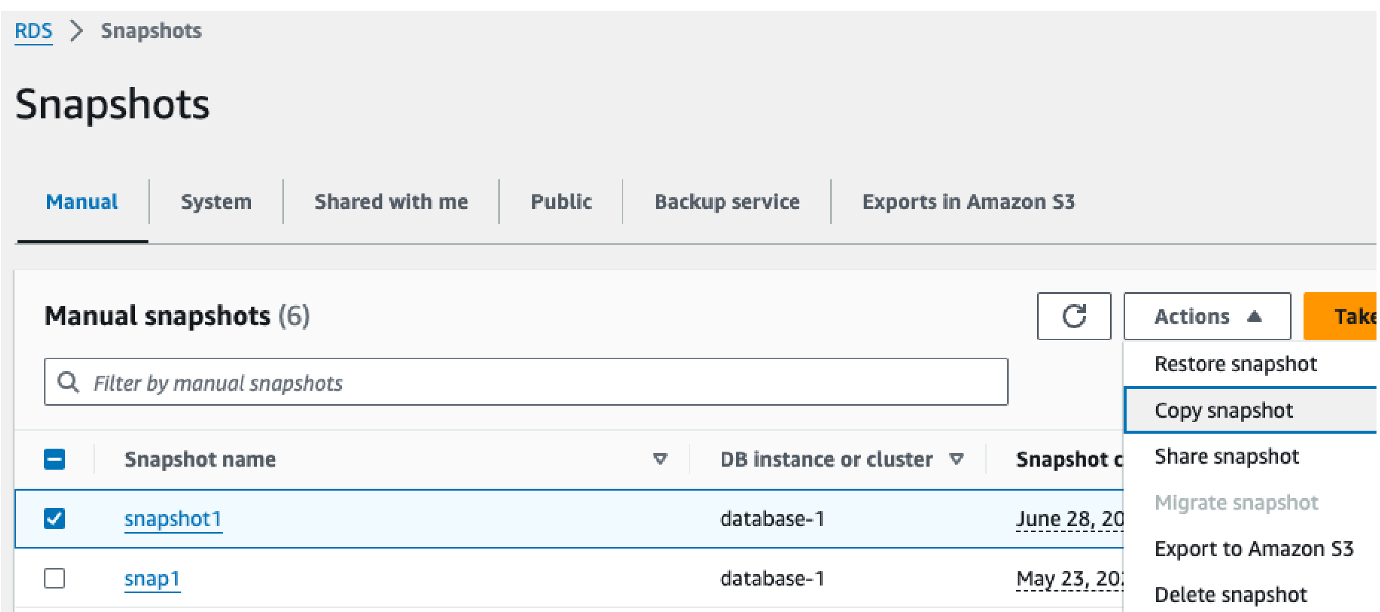
이 절차는 동일한 AWS 리전 또는 리전 간에 암호화된 DB 클러스터 스냅샷 또는 암호화되지 않은 DB 클러스터 스냅샷을 복사하는 데 사용됩니다.

진행 중인 복사 작업을 취소하려면 대상 DB 클러스터 스냅샷이 copying 상태에 있는 동안 해당 DB 클러스터 스냅샷을 삭제합니다.

DB 클러스터 스냅샷을 복사하기 전에 제한 사항 및 스냅샷 복사 고려 사항의 내용을 검토하세요.

DB 클러스터 스냅샷을 복사하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 복사하려는 DB 클러스터 스냅샷을 선택합니다.
4. 작업을 선택한 다음 스냅샷 복사를 선택합니다.



5. (선택 사항) DB 클러스터 스냅샷을 다른 AWS 리전에 복사하려면 대상 리전에서 해당 AWS 리전을 선택합니다.

6. 새 DB 스냅샷 식별자에 DB 클러스터 스냅샷 복사본의 이름을 입력합니다.
7. 스냅샷의 태그와 값들을 스냅샷 사본에 복사하려면 [Copy Tags]를 선택합니다.
8. [Copy Snapshot]을 선택합니다.

제한 사항

다음은 스냅샷을 복사할 때 적용되는 몇몇 제한 사항입니다.

- 다음 AWS 리전으로/에서는 스냅샷을 복사할 수 없습니다.
  - 중국(베이징)
  - 중국(닝샤)
- AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 간에 스냅샷을 복사할 수 있습니다. 그러나 이러한 AWS GovCloud (US) 리전과 상용 AWS 리전 사이에는 스냅샷을 복사할 수 없습니다.
- 대상 스냅샷이 제공되기 전에 소스 스냅샷을 삭제하면 스냅샷 복사가 실패할 수 있습니다. 원본 스냅샷을 삭제하기 전에 대상 스냅샷의 상태가 `AVAILABLE`인지 확인하십시오.
- 계정당 단일 대상 리전으로 최대 5개의 스냅샷 복사 요청이 진행될 수 있습니다.
- 동일한 소스 DB 인스턴스에 대해 여러 스냅샷 복사본을 요청하면 내부적으로 대기열에 들어갑니다. 나중에 요청한 복사본은 이전 스냅샷 복사본이 완료될 때까지 시작되지 않습니다. 자세한 내용은 AWS 지식 센터의 EC2 AMI 또는 EBS 스냅샷 생성 속도가 느린 이유는 무엇입니까? 단원을 참조하세요.
- 관련된 AWS 리전과 복사할 데이터 양에 따라 교차 리전 스냅샷 복사를 완료하는 데 몇 시간이 걸릴 수 있습니다. 경우에 따라 지정된 소스 리전으로부터 대량의 리전 간 스냅샷 복사 요청이 있을 수 있습니다. 이러한 경우 진행 중인 복사본 중 일부가 완료될 때까지 Amazon RDS가 해당 소스 리전의 새로운 교차 리전 복사 요청을 대기열에 넣을 수 있습니다. 대기열에 있는 복사 요청에 대한 진행 정보는 표시되지 않습니다. 복사가 시작되면 진행 정보가 표시됩니다.
- Aurora에서는 증분 스냅샷을 지원하지 않습니다. Aurora DB 클러스터 스냅샷 복사본은 항상 전체 복사본으로 저장됩니다. 전체 스냅샷 복사에는 DB 클러스터 복원에 필요한 모든 데이터 및 메타데이터가 포함됩니다.

스냅샷 복사 고려 사항

스냅샷을 복사할 때 고려해야 할 사항은 다음과 같습니다.

주제

- [공유 스냅샷 복사 시 고려 사항]
- [암호화된 DB 클러스터 스냅샷 복사 시 고려 사항]
- [리전 간 스냅샷 복사 시 고려 사항]
- [파라미터 그룹에 대한 고려 사항]

공유 스냅샷 복사 시 고려 사항

다른 AWS 계정이 공유하는 스냅샷을 복사할 수 있습니다. 경우에 따라 다른 AWS 계정에서 공유된 암호화된 스냅샷을 복사할 수 있습니다. 이 경우 스냅샷을 암호화하는 데 사용된 AWS KMS key에 대한 액세스 권한이 있어야 합니다.

동일한 AWS 리전에서는 암호화 여부와 관계없이 공유 DB 클러스터 스냅샷만 복사할 수 있습니다. 자세한 내용은 암호화된 스냅샷 공유 섹션을 참조하세요.

암호화된 DB 클러스터 스냅샷 복사 시 고려 사항

KMS 키를 사용하여 암호화된 스냅샷을 복사할 수 있습니다. 암호화된 스냅샷을 복사할 경우 스냅샷의 사본도 암호화해야 합니다. 동일한 AWS 리전 내에서 암호화된 스냅샷을 복사하는 경우 원본 스냅샷과 동일한 KMS 키를 사용하여 사본을 암호화할 수 있습니다. 또는 다른 KMS 키를 지정할 수 있습니다.

리전 간에 암호화된 스냅샷을 복사하는 경우 대상 AWS 리전에서 유효한 KMS 키를 지정해야 합니다. 리전별 KMS 키 또는 다중 리전 키일 수 있습니다. 다중 리전 KMS 키에 대한 자세한 내용은 AWS KMS에서 다중 리전 키 사용을 참조하세요.

Amazon RDS의 AWS KMS 키 관리에 대한 자세한 내용은 AWS KMS key 관리 섹션을 참조하세요.

소스 스냅샷은 복사 프로세스 전체에서 암호화를 유지합니다. 자세한 내용은 Amazon Aurora 암호화된 DB 클러스터의 제한 섹션을 참조하세요.

> **Note**
>
> Amazon Aurora DB 클러스터 스냅샷의 경우 암호화되지 않은 DB 클러스터 스냅샷은 스냅샷을 복사할 때 암호화할 수 없습니다.

암호화된 DB 클러스터 스냅샷을 복사하려면 다음 주제를 참조하세요.

- AWS CLI 또는 Amazon RDS API를 사용하여 암호화된 DB 클러스터 스냅샷 복사

- 계정 간에 DB 클러스터 스냅샷 복사

리전 간 스냅샷 복사 시 고려 사항

AWS 리전 간에 DB 클러스터 스냅샷을 복사할 수 있습니다. 하지만 리전 간 스냅샷 복사와 관련한 특정 제약 및 고려 사항이 있습니다.

관련된 AWS 리전과 복사할 데이터 양에 따라 교차 리전 스냅샷 복사를 완료하는 데 몇 시간이 걸릴 수 있습니다.

경우에 따라 지정된 소스 AWS 리전으로부터 대량의 교차 리전 스냅샷 복사 요청이 있을 수 있습니다. 이러한 경우 진행 중인 복사본 중 일부가 완료될 때까지 Amazon RDS가 해당 소스 AWS 리전의 새로운 교차 리전 복사 요청을 대기열에 넣을 수 있습니다. 대기열에 있는 복사 요청에 대한 진행 정보는 표시되지 않습니다. 복사가 시작되면 진행 정보가 표시됩니다.

교차 리전 스냅샷 복사를 사용하면 대상 데이터에 전체 복사본이 생성되지만, 데이터 전송 요금이 증분됩니다. 증분 데이터에는 마지막 복사 이후 고객 데이터베이스에 추가된 새 데이터와 기존 데이터에 대한 변경 사항이 모두 포함됩니다. 자세한 내용은 AWS Backup 개발자 안내서의 AWS 리전 전체에 백업 복사본 생성을 참조하세요.

> **Note**
>
> Aurora는 대상 리전에서 스냅샷의 전체 사본을 만드는 데 필요한 최소량의 데이터를 복사합니다. 리전 간에 스냅샷을 복사할 때 데이터 전송 요금이 부과됩니다.

파라미터 그룹에 대한 고려 사항

리전 간에 스냅샷을 복사하는 경우 복사에는 원래 DB 클러스터에서 사용된 파라미터 그룹이 포함되지 않습니다. 스냅샷을 복원하여 새 DB 클러스터를 만들면 해당 DB 클러스터가 생성된 AWS 리전에 대한 기본 파라미터 그룹을 얻습니다. 새 DB 클러스터에 오리지널과 같은 파라미터를 지정하려면 다음을 수행합니다.

1. 대상 AWS 리전에서 원래 DB 클러스터와 동일한 설정으로 DB 클러스터 파라미터 그룹을 생성합니다. 새 AWS 리전에 이미 옵션 그룹이 있는 경우 이 그룹을 사용할 수 있습니다.
2. 대상 AWS 리전에 스냅샷을 복원한 후 새 DB 클러스터를 수정하여 이전 단계의 신규 또는 기존 파라미터 그룹을 추가합니다.

AWS CLI 또는 Amazon RDS API를 사용하여 암호화되지 않은 DB 클러스터 스냅샷 복사

다음 섹션의 절차에 따라 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 암호화되지 않은 DB 클러스터 스냅샷을 복사합니다.

진행 중인 복사 작업을 취소하려면 대상 DB 클러스터 스냅샷이 copying 상태에 있는 동안 `--target-db-cluster-snapshot-identifier` 또는 `TargetDBClusterSnapshotIdentifier`로 식별된 해당 DB 클러스터 스냅샷을 삭제합니다.

콘솔

AWS Management Console을 사용하여 DB 클러스터 스냅샷을 복사하려면 AWS Management Console을 사용한 DB 클러스터 스냅샷 복사 섹션을 참조하세요.

AWS CLI

AWS CLI copy-db-cluster-snapshot 명령을 사용하여 DB 클러스터 스냅샷을 복사할 수 있습니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 스냅샷을 복사할 AWS 리전에서 명령을 실행합니다.

다음 옵션을 사용하여 암호화되지 않은 DB 클러스터 스냅샷을 복사할 수 있습니다.

- `--source-db-cluster-snapshot-identifier` – 복사할 DB 클러스터 스냅샷의 식별자입니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 식별자는 소스 AWS 리전의 ARN 형식이어야 합니다.
- `--target-db-cluster-snapshot-identifier` – DB 클러스터 스냅샷의 새 사본의 식별자입니다.

다음 코드는 명령이 실행되는 AWS 리전에 `myclustersnapshotcopy`라는 DB 클러스터 스냅샷 `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805`의 사본을 만듭니다. 복사가 완료되면 원래 스냅샷의 모든 태그가 스냅샷 사본에 복사됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds copy-db-cluster-snapshot \
```

```
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

Windows의 경우:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS API

DB 클러스터 스냅샷을 복사하려면 Amazon RDS API CopyDBClusterSnapshot 작업을 사용합니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 스냅샷을 복사할 AWS 리전에서 작업을 수행합니다.

다음 파라미터를 사용하여 암호화되지 않은 DB 클러스터 스냅샷을 복사할 수 있습니다.

- `SourceDBClusterSnapshotIdentifier` – 복사할 DB 클러스터 스냅샷의 식별자입니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 식별자는 소스 AWS 리전의 ARN 형식이어야 합니다.
- `TargetDBClusterSnapshotIdentifier` – DB 클러스터 스냅샷의 새 사본의 식별자입니다.

다음 코드는 미국 서부(캘리포니아 북부 지역) 리전에 `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805`라는 이름으로 스냅샷 `myclustersnapshotcopy`의 복사본을 생성합니다. 복사가 완료되면 원래 스냅샷의 모든 태그가 스냅샷 사본에 복사됩니다.

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
```

```
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

AWS CLI 또는 Amazon RDS API를 사용하여 암호화된 DB 클러스터 스냅샷 복사

다음 섹션의 절차에 따라 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 암호화된 DB 클러스터 스냅샷을 복사합니다.

진행 중인 복사 작업을 취소하려면 대상 DB 클러스터 스냅샷이 copying 상태에 있는 동안 `--target-db-cluster-snapshot-identifier` 또는 `TargetDBClusterSnapshotIdentifier`로 식별된 해당 DB 클러스터 스냅샷을 삭제합니다.

콘솔

AWS Management Console을 사용하여 DB 클러스터 스냅샷을 복사하려면 AWS Management Console을 사용한 DB 클러스터 스냅샷 복사 섹션을 참조하세요.

AWS CLI

AWS CLI copy-db-cluster-snapshot 명령을 사용하여 DB 클러스터 스냅샷을 복사할 수 있습니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 스냅샷을 복사할 AWS 리전에서 명령을 실행합니다.

다음 옵션을 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있습니다.

- `--source-db-cluster-snapshot-identifier` – 복사할 암호화된 DB 클러스터 스냅샷의 식별자입니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 식별자는 소스 AWS 리전의 ARN 형식이어야 합니다.
- `--target-db-cluster-snapshot-identifier` – 암호화된 DB 클러스터 스냅샷의 새 사본의 식별자입니다.
- `--kms-key-id` - DB 클러스터 스냅샷의 복사본을 암호화하는 데 사용할 키의 KMS 키 식별자입니다.

  DB 클러스터 스냅샷이 암호화되어 있으며, 동일한 AWS 리전에 스냅샷을 복사하고, 복사본을 암호화할 새 KMS 키를 지정하려는 경우 이 옵션을 사용해도 됩니다. 그렇지 않으면 DB 클러스터 스냅샷의 사본이 원본 DB 클러스터 스냅샷과 동일한 KMS 키로 암호화됩니다.

DB 클러스터 스냅샷이 암호화되어 있으며 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 옵션을 사용해야 합니다. 이 경우 대상 AWS 리전에 대해 KMS 키를 지정해야 합니다.

다음 코드 예시에서는 암호화된 DB 클러스터 스냅샷을 미국 서부(오레곤) 리전에서 US East (N. Virginia) 리전으로 복사합니다. 이 명령은 US East (N. Virginia) 리전에서 호출됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

Windows의 경우:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

이 `--source-region` 파라미터는 AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전 간에 암호화된 DB 클러스터 스냅샷을 복사할 때 필요합니다. `--source-region`의 경우 소스 DB 인스턴스의 AWS 리전을 지정합니다. `source-db-cluster-snapshot-identifier`에 지정된 AWS 리전은 `--source-region`에 지정된 AWS 리전과 일치해야 합니다.

`--source-region`이 지정되지 않은 경우에는 `--pre-signed-url` 값을 지정합니다. 미리 서명된 URL은 소스 AWS 리전에서 호출되는 `copy-db-cluster-snapshot` 명령에 대한 서명 버전 4의 서명된 요청이 포함된 URL입니다. `pre-signed-url` 옵션에 대한 자세한 내용은 AWS CLI 명령 참조에서 copy-db-cluster-snapshot을 참조하세요.

RDS API

DB 클러스터 스냅샷을 복사하려면 Amazon RDS API CopyDBClusterSnapshot 작업을 사용합니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 스냅샷을 복사할 AWS 리전에서 작업을 수행합니다.

다음 파라미터를 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있습니다.

- `SourceDBClusterSnapshotIdentifier` – 복사할 암호화된 DB 클러스터 스냅샷의 식별자입니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 식별자는 소스 AWS 리전의 ARN 형식이어야 합니다.
- `TargetDBClusterSnapshotIdentifier` – 암호화된 DB 클러스터 스냅샷의 새 사본의 식별자입니다.
- `KmsKeyId` - DB 클러스터 스냅샷의 복사본을 암호화하는 데 사용할 키의 KMS 키 식별자입니다.

  DB 클러스터 스냅샷이 암호화되어 있으며, 동일한 AWS 리전에 스냅샷을 복사하고, 복사본을 암호화하는 데 사용할 새 KMS 키를 지정하는 경우 이 파라미터를 사용해도 됩니다. 그렇지 않으면 DB 클러스터 스냅샷의 사본이 원본 DB 클러스터 스냅샷과 동일한 KMS 키로 암호화됩니다.

  DB 클러스터 스냅샷이 암호화되어 있으며 스냅샷을 다른 AWS 리전에 복사하려는 경우 이 파라미터를 사용해야 합니다. 이 경우 대상 AWS 리전에 대해 KMS 키를 지정해야 합니다.
- `PreSignedUrl` - 스냅샷을 다른 AWS 리전에 복사하려는 경우 `PreSignedUrl` 파라미터를 지정해야 합니다. `PreSignedUrl` 값은 DB 클러스터 스냅샷을 복사해올 소스 AWS 리전에서 `CopyDBClusterSnapshot` 작업이 호출되도록 하는 서명 버전 4의 서명된 요청이 포함된 URL이어야 합니다. 미리 서명된 URL을 사용하는 방법에 대해 자세히 알아보려면 [CopyDBClusterSnapshot](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_CopyDBClusterSnapshot.html)을 참조하세요.

다음 코드 예시에서는 암호화된 DB 클러스터 스냅샷을 미국 서부(오레곤) 리전에서 US East (N. Virginia) 리전으로 복사합니다. 이 작업은 미국 동부(버지니아 북부) 리전에서 호출됩니다.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
```

```
        %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
        %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
        %2526X-Amz-Date%253D20161117T215409Z
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

이 `PreSignedUrl` 파라미터는 AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전 간에 암호화된 DB 클러스터 스냅샷을 복사할 때 필요합니다. `PreSignedUrl` 값은 DB 클러스터 스냅샷을 복사해올 소스 AWS 리전에서 `CopyDBClusterSnapshot` 작업이 호출되도록 하는 서명 버전 4의 서명된 요청이 포함된 URL이어야 합니다. 미리 서명된 URL 사용에 대한 자세한 내용은 Amazon RDS API Reference(Amazon RDS API 참조)의 CopyDBClusterSnapshot을 참조하세요.

미리 서명된 URL을 수동이 아닌 자동으로 생성하려면 `--source-region` 옵션 대신 AWS CLIcopy-db-cluster-snapshot 명령을 사용합니다.

계정 간에 DB 클러스터 스냅샷 복사

Amazon RDS API `ModifyDBClusterSnapshotAttribute` 및 `CopyDBClusterSnapshot` 작업을 사용하여 다른 AWS 계정에서 지정한 DB 클러스터 스냅샷을 복사할 수 있습니다. 동일한 AWS 리전의 계정 간에만 DB 클러스터 스냅샷을 복사할 수 있습니다. 계정 간 복사 프로세스는 다음과 같이 작동합니다. 여기에서 계정 A는 스냅샷을 복사할 수 있도록 하고, 계정 B는 이를 복사합니다.

1. 계정 A에서 `ModifyDBClusterSnapshotAttribute` 파라미터에 **`restore`**를 지정하고 `AttributeName` 파라미터에 계정 B의 ID를 지정하여 `ValuesToAdd`를 호출합니다.

2. (스냅샷이 암호화되어 있는 경우) 계정 A를 사용하여 KMS 키의 키 정책을 업데이트합니다. 먼저 계정 B의 ARN을 `Principal`로 추가한 다음 `kms:CreateGrant` 작업을 허용합니다.
3. (스냅샷이 암호화되어 있는 경우) 계정 B를 사용하여 사용자를 선택하거나 만들고, 해당 사용자에 IAM 정책을 연결합니다. 그러면 사용자가 KMS 키를 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있습니다.
4. 계정 B를 사용하여 `CopyDBClusterSnapshot`을 호출하고, `SourceDBClusterSnapshotIdentifier` 파라미터를 사용하여 복사할 DB 클러스터 스냅샷의 ARN을 지정합니다. 이 ARN에는 계정 A의 ID가 포함되어야 합니다.

DB 클러스터 스냅샷을 복원할 수 있도록 허용된 모든 AWS 계정을 나열하려면 DescribeDBSnapshotAttributes 또는 DescribeDBClusterSnapshotAttributes API 작업을 사용합니다.

AWS 계정의 공유 권한을 제거하려면 `ModifyDBSnapshotAttribute` 또는 `ModifyDBClusterSnapshotAttribute` 작업을 사용합니다. 이때 `AttributeName`은 `restore`로 설정하고, `ValuesToRemove` 파라미터에는 제거할 계정의 ID를 지정합니다.

암호화되지 않은 DB 클러스터 스냅샷을 다른 계정에 복사

다음 절차를 사용하여 암호화되지 않은 DB 클러스터 스냅샷을 동일한 AWS 리전의 다른 계정에 복사할 수 있습니다.

1. DB 클러스터 스냅샷의 소스 계정에서 `ModifyDBClusterSnapshotAttribute` 파라미터에 **`restore`**를 지정하고 `AttributeName` 파라미터에 대상 계정의 ID를 지정하여 `ValuesToAdd`를 호출합니다.

   계정 `987654321`을 사용하는 다음 예를 실행하면 AWS 및 `123451234512`라는 두 개의 `123456789012` 계정 식별자가 `manual-snapshot1`라는 DB 클러스터 스냅샷을 복원할 수 있습니다.

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
```

```
&X-Amz-Date=20150922T220515Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. 대상 계정에서 `CopyDBClusterSnapshot`을 호출하고, `SourceDBClusterSnapshotIdentifier` 파라미터를 사용하여 복사할 DB 클러스터 스냅샷의 ARN을 지정합니다. 이 ARN에는 원본 계정의 ID가 포함되어야 합니다.

   계정 `123451234512`를 사용하는 다음 예를 실행하면 계정 `aurora-cluster1-snapshot-20130805`에서 DB 클러스터 스냅샷 `987654321`가 복사되고, `dbclustersnapshot1`라는 DB 클러스터 스냅샷이 생성됩니다.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

암호화된 DB 클러스터 스냅샷을 다른 계정에 복사

다음 절차를 사용하여 암호화된 DB 클러스터 스냅샷을 동일한 AWS 리전의 다른 계정에 복사할 수 있습니다.

1. DB 클러스터 스냅샷의 소스 계정에서 `ModifyDBClusterSnapshotAttribute` 파라미터에 **`restore`**를 지정하고 `AttributeName` 파라미터에 대상 계정의 ID를 지정하여 `ValuesToAdd`를 호출합니다.

   계정 `987654321`을 사용하는 다음 예를 실행하면 AWS 및 `123451234512`라는 두 개의 `123456789012` 계정 식별자가 `manual-snapshot1`라는 DB 클러스터 스냅샷을 복원할 수 있습니다.

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. DB 클러스터 스냅샷의 소스 계정에서 암호화된 DB 클러스터 스냅샷과 동일한 AWS 리전으로 사용자 지정 KMS 키를 생성합니다. 고객 관리 키를 생성할 때는 대상 AWS 계정에 키에 대한 액세스 권한을 부여합니다. 자세한 내용은 고객 관리 키를 생성하고 액세스 권한 부여 단원을 참조하십시오.
3. 스냅샷을 복사하여 대상 AWS 계정에 공유합니다. 자세한 내용은 소스 계정에서 스냅샷 복사 및 공유 단원을 참조하십시오.
4. 대상 계정에서 `CopyDBClusterSnapshot`을 호출하고, `SourceDBClusterSnapshotIdentifier` 파라미터를 사용하여 복사할 DB 클러스터 스냅샷의 ARN을 지정합니다. 이 ARN에는 원본 계정의 ID가 포함되어야 합니다.

   계정 `123451234512`를 사용하는 다음 예를 실행하면 계정 `aurora-cluster1-snapshot-20130805`에서 DB 클러스터 스냅샷 `987654321`가 복사되고, `dbclustersnapshot1`라는 DB 클러스터 스냅샷이 생성됩니다.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
```

```
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
    &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

DB 클러스터 스냅샷 공유

Amazon RDS을 사용하면 다른 방법으로 수동 DB 클러스터 스냅샷을 공유할 수 있습니다.

- 암호화되었거나 암호화되지 않은 수동 DB 클러스터 스냅샷을 공유하면 권한이 있는 AWS 계정에서 해당 스냅샷을 복사할 수 있습니다.
- 암호화되었거나 암호화되지 않은 수동 DB 클러스터 스냅샷을 공유하면 권한이 있는 AWS 계정에서 스냅샷의 복사본을 만든 후 복원하는 대신에 해당 스냅샷에서 DB 클러스터를 직접 복원할 수 있습니다.

> **Note**
>
> 자동 DB 클러스터 스냅샷을 공유하려면 자동 스냅샷을 복사하여 수동 DB 클러스터 스냅샷을 생성한 다음 해당 복사본을 공유합니다. 이 프로세스는 AWS Backup에서 생성된 리소스에도 적용됩니다.

스냅샷 복사에 대한 자세한 정보는 DB 클러스터 스냅샷 복사 단원을 참조하십시오. DB 클러스터 스냅샷에서 DB 인스턴스를 복원하는 방법에 대한 자세한 정보는 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.

DB 클러스터 스냅샷에서 DB 클러스터를 복원하는 방법에 대한 자세한 정보는 Aurora DB 클러스터 백업 및 복원에 대한 개요을 참조하십시오.

최대 20개의 다른 AWS 계정 계정과 수동 스냅샷을 공유할 수 있습니다.

수동 스냅샷을 다른 AWS 계정와 공유할 경우 다음과 같은 제한이 적용됩니다.

- AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 공유 스냅샷에서 DB 클러스터를 복원할 때는 공유 스냅샷의 Amazon 리소스 이름(ARN)을 스냅샷 식별자로 지정해야 합니다.

다음 섹션에서 스냅샷, 퍼블릭 스냅샷, 암호화된 스냅샷을 공유하는 방법을 알아봅니다. 스냅샷 공유 중지 방법도 알아볼 수 있습니다.

주제

- [퍼블릭 스냅샷 공유]
- [암호화된 스냅샷 공유]
- [스냅샷 공유 중지]

스냅샷 공유

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷을 공유할 수 있습니다.

콘솔

Amazon RDS 콘솔을 사용하여 최대 20개의 AWS 계정와 수동 DB 클러스터 스냅샷을 공유할 수 있습니다. 콘솔을 사용하여 하나 이상의 계정에 대한 수동 스냅샷 공유를 중지할 수도 있습니다.

Amazon RDS 콘솔을 사용하여 수동 DB 클러스터 스냅샷을 공유하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 공유할 수동 스냅샷을 선택합니다.
4. Actions(작업)에서 Share snapshot(스냅샷 공유)을 선택합니다.
5. DB 스냅샷 공개 여부에 대해 다음 옵션 중 하나를 선택합니다.
   - 소스가 암호화되어 있지 않은 경우 퍼블릭을 선택하여 모든 AWS 계정가 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하도록 허용하거나, 프라이빗을 선택하여 지정한 AWS 계정만 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하도록 허용합니다.

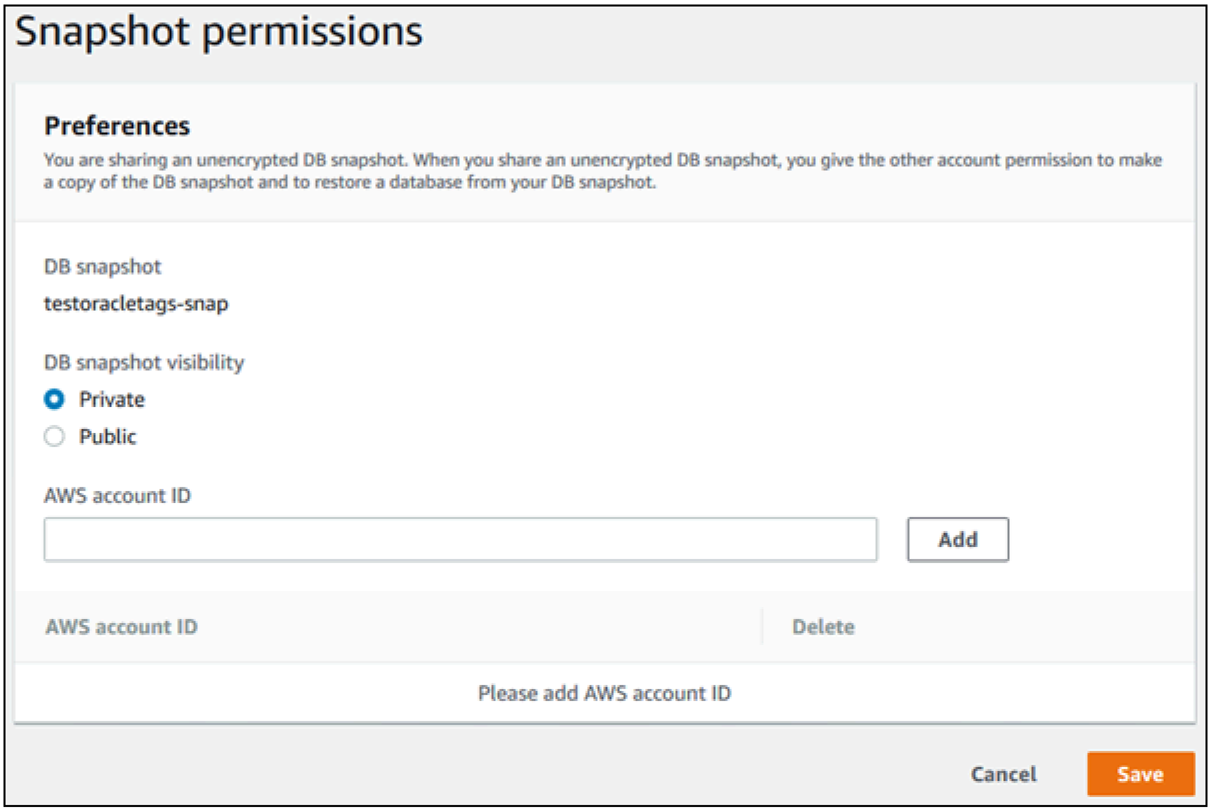
     > **Warning**
     >
     > DB 스냅샷 가시성을 퍼블릭으로 설정한 경우 모든 AWS 계정는 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하고 사용자 데이터에 액세스할 수 있습니다. 프라이빗 정보가 포함된 수동 DB 클러스터 스냅샷을 퍼블릭으로 공유하지 마십시오.
     > 자세한 내용은 [퍼블릭 스냅샷 공유] 단원을 참조하십시오.

   - 원본 DB 클러스터가 암호화되어 있는 경우 암호화된 스냅샷을 퍼블릭으로 공유할 수 없으므로 DB snapshot visibility(DB 스냅샷 가시성)는 Private(프라이빗)으로 설정됩니다.

> **Note**
>
> 기본 AWS KMS key로 암호화된 스냅샷은 공유할 수 없습니다. 이 문제를 해결하는 방법에 대한 자세한 내용은 암호화된 스냅샷 공유 섹션을 참조하세요.

6. AWS 계정 ID에 수동 스냅샷에서 DB 클러스터를 복원하도록 허용할 계정의 AWS 계정 계정 식별자를 입력한 다음 추가를 선택합니다. 이 과정을 반복하여 최대 20개의 AWS 계정까지 AWS 계정 식별자를 추가합니다.

   허용된 계정 목록에 AWS 계정 식별자를 추가하다가 실수하는 경우, 잘못된 AWS 계정 식별자의 오른쪽에 있는 삭제를 선택하여 목록에서 해당 식별자를 삭제할 수 있습니다.



7. 수동 스냅샷을 복원할 수 있도록 허용할 모든 AWS 계정의 식별자를 추가한 후 저장을 선택하여 변경 내용을 저장합니다.

AWS CLI

DB 클러스터 스냅샷을 공유하려면 `aws rds modify-db-cluster-snapshot-attribute` 명령을 사용합니다. `--values-to-add` 파라미터를 사용하여 수동 스냅샷을 복원할 권한이 있는 AWS 계정 계정의 ID 목록을 추가합니다.

Example 단일 계정과 스냅샷 공유

다음 예제에서는 123456789012라는 AWS 계정 식별자가 `cluster-3-snapshot`이라는 DB 클러스터 스냅샷을 복원할 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

Windows의 경우:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example 여러 계정과 스냅샷 공유

다음 예제에서는 111122223333 및 444455556666이라는 두 개의 AWS 계정 식별자가 `manual-cluster-snapshot1`이라는 DB 클러스터 스냅샷을 복원할 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

Windows의 경우:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> Windows 명령 프롬프트를 사용하는 경우 백슬래시(\)를 접두사로 추가하여 JSON 코드에서 큰 따옴표(")를 이스케이프해야 합니다.

스냅샷을 복원할 수 있는 AWS 계정를 나열하려면 [describe-db-cluster-snapshot-attributes](#) AWS CLI 명령을 사용합니다.

RDS API

Amazon RDS API를 사용하여 수동 DB 클러스터 스냅샷을 다른 AWS 계정와 공유할 수도 있습니다. 이렇게 하려면 [ModifyDBClusterSnapshotAttribute](#) 작업을 호출합니다. `AttributeName`에 대해 `restore`를 지정하고 `ValuesToAdd` 파라미터를 사용하여 수동 스냅샷의 복원 권한이 있는 AWS 계정를 위한 ID 목록을 추가합니다.

수동 스냅샷을 퍼블릭으로 설정하고 모든 AWS 계정에서 복원할 수 있도록 하려면 `all` 값을 사용합니다. 단, 일부 AWS 계정에만 제공할 비공개 정보가 포함된 수동 스냅샷에 대해 `all` 값을 추가하지 않도록 유의합니다. 또한 암호화된 스냅샷에 대해 `all`을 지정하지 마십시오. 그러한 스냅샷을 퍼블릭으로 설정하는 것은 지원되지 않습니다.

스냅샷을 복원할 수 있도록 허용된 모든 AWS 계정를 나열하려면 [DescribeDBClusterSnapshotAttributes](#) API 작업을 사용합니다.

퍼블릭 스냅샷 공유

암호화되지 않은 수동 스냅샷을 퍼블릭으로 공유할 수 있습니다. 그러면 모든 AWS 계정에서 해당 스냅샷을 사용할 수 있습니다. 스냅샷을 퍼블릭으로 공유할 경우, 퍼블릭 스냅샷에 비공개 정보가 포함되지 않도록 유의하세요.

스냅샷이 공개적으로 공유될 경우 스냅샷을 복사하고 해당 스냅샷에서 DB 클러스터를 생성할 수 있는 권한을 모든 AWS 계정에 부여합니다.

다른 계정이 소유한 퍼블릭 스냅샷의 백업 스토리지에 대해서는 요금이 청구되지 않습니다. 본인이 소유한 스냅샷에 대해서만 요금이 청구됩니다.

퍼블릭 스냅샷을 복사할 경우 해당 복사본을 소유하게 됩니다. 스냅샷 복사본의 백업 스토리지에 대한 요금이 청구됩니다. 퍼블릭 스냅샷에서 DB 클러스터를 생성할 경우 해당 DB 클러스터에 대한 요금이 청구됩니다. Amazon Aurora 요금에 대한 자세한 내용은 [Aurora 요금 페이지](#)를 참조하세요.

본인이 소유한 퍼블릭 스냅샷만 삭제할 수 있습니다. 공유 또는 퍼블릭 스냅샷을 삭제하려면 해당 스냅샷을 소유한 AWS 계정에 로그인해야 합니다.

다른 AWS 계정 계정이 소유한 퍼블릭 스냅샷 보기

Amazon RDS 콘솔의 스냅샷 페이지에 있는 퍼블릭 탭에서 특정 AWS 리전의 다른 계정이 소유한 퍼블릭 스냅샷을 볼 수 있습니다. 사용 중인 계정이 소유한 스냅샷은 이 탭에 표시되지 않습니다.

퍼블릭 스냅샷을 보려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. [퍼블릭(Public)] 탭을 선택합니다.

   퍼블릭 스냅샷이 표시됩니다. [소유자(Owner)] 열에서 퍼블릭 스냅샷을 소유한 계정을 확인할 수 있습니다.

   > **Note**
   >
   > 이 열을 보려면 [퍼블릭 스냅샷(Public snapshots)] 목록의 오른쪽 상단에 있는 톱니바퀴 아이콘을 선택하여 페이지 기본 설정을 수정해야 할 수도 있습니다.

본인이 소유한 퍼블릭 스냅샷 보기

다음 AWS CLI 명령(Unix에만 해당)을 사용하여 특정 AWS 리전의 AWS 계정가 소유한 퍼블릭 스냅샷을 볼 수 있습니다.

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

퍼블릭 스냅샷이 있는 경우 반환되는 출력은 다음 예제와 유사합니다.

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

지원 중단된 DB 엔진 버전의 퍼블릭 스냅샷 공유

더 이상 사용되지 않는 DB 엔진 버전에서 퍼블릭 스냅샷을 복원하거나 복사하는 것은 지원되지 않습니다. 지원되지 않는 기존 퍼블릭 스냅샷을 복원하거나 복사할 수 있게 하려면 다음 단계를 수행하세요.

1. 스냅샷을 프라이빗으로 표시합니다.
2. 스냅샷을 복원합니다.
3. 복원된 DB 클러스터를 지원되는 엔진 버전으로 업그레이드합니다.
4. 새로운 스냅샷을 생성합니다.
5. 스냅샷을 공개적으로 다시 공유합니다.

암호화된 스냅샷 공유

Amazon Aurora 리소스 암호화에 설명된 대로 AES-256 암호화 알고리즘을 사용하여 '저장 상태'에서 암호화된 DB 클러스터 스냅샷을 공유할 수 있습니다.

다음 제한은 암호화된 스냅샷 공유에 적용됩니다.

- 암호화된 스냅샷을 퍼블릭으로는 공유할 수 없습니다.
- 스냅샷을 공유한 AWS 계정의 기본 KMS 키를 사용하여 암호화된 스냅샷은 공유할 수 없습니다.

  Amazon RDS의 AWS KMS 키 관리에 대한 자세한 내용은 AWS KMS key 관리 섹션을 참조하세요.

기본 KMS 키 문제를 해결하려면 다음 작업을 수행하세요.

1. 고객 관리 키를 생성하고 액세스 권한 부여.
2. 소스 계정에서 스냅샷 복사 및 공유.
3. 공유된 스냅샷을 대상 계정에 복사.

고객 관리 키를 생성하고 액세스 권한 부여

먼저 암호화된 DB 클러스터 스냅샷과 동일한 AWS 리전에 있는 사용자 지정 KMS 키를 생성합니다. 고객 관리 키를 생성할 때는 다른 AWS 계정에 키에 대한 액세스 권한을 부여합니다.

고객 관리 키를 생성하고 액세스 권한을 부여하는 방법

1. AWS 계정 소스에서 AWS Management Console에 로그인합니다.

2. AWS KMS 콘솔(https://console.aws.amazon.com/kms)을 엽니다.
3. AWS 리전을 변경하려면 페이지의 오른쪽 상단 모서리에 있는 리전 선택기를 사용합니다.
4. 탐색 창에서 고객 관리형 키를 선택합니다.
5. 키 생성을 선택합니다.
6. 키 구성 페이지에서 다음 단계를 따릅니다.

   a. 키 유형에서 대칭을 선택합니다.
   b. 키 사용에서 암호화 및 해독을 선택합니다.
   c. 고급 옵션을 확장합니다.
   d. 키 구성 요소 원본에서 KMS를 선택합니다.
   e. 리전 특성에서 단일 리전 키를 선택합니다.
   f. Next(다음)를 선택합니다.
7. 레이블 추가 페이지에서 다음 단계를 따릅니다.

   a. 별칭에 KMS 키의 표시 이름을 입력합니다(예: **`share-snapshot`**).
   b. (선택 사항) KMS 키에 대한 설명을 입력합니다.
   c. (선택 사항) KMS 키에 태그를 추가합니다.
   d. Next(다음)를 선택합니다.
8. 키 관리 권한 정의 페이지에서 다음을 선택합니다.
9. 키 사용 권한 정의 페이지에서 다음 단계를 따릅니다.

   a. 기타 AWS 계정에서 다른 AWS 계정 추가를 선택합니다.
   b. 액세스 권한을 부여할 AWS 계정 ID를 입력합니다.

      여러 AWS 계정에 액세스 권한을 부여할 수 있습니다.
   c. Next(다음)를 선택합니다.
10. KMS 키를 검토한 다음 완료를 선택합니다.

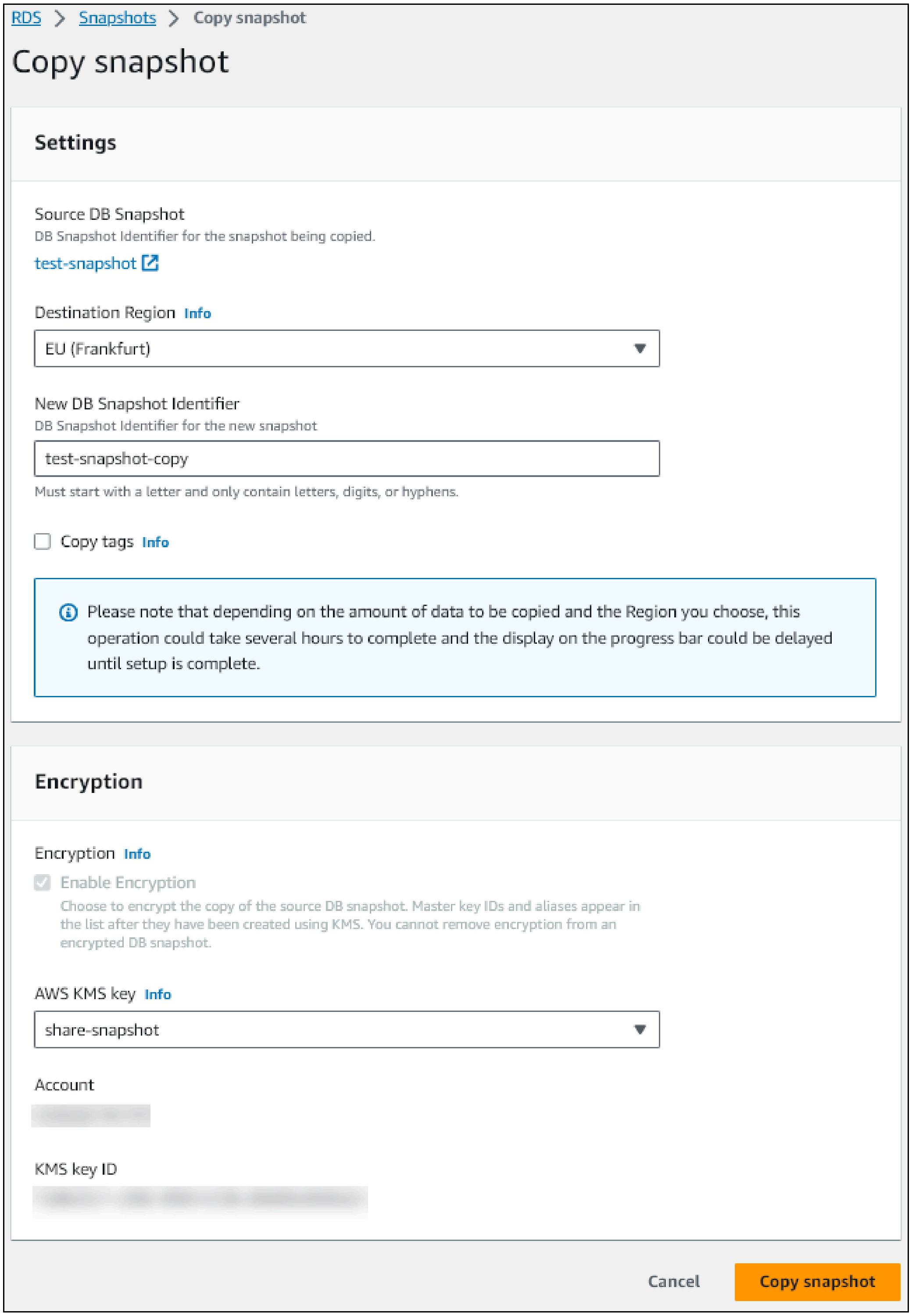
소스 계정에서 스냅샷 복사 및 공유

다음으로 고객 관리 키를 사용하여 소스 DB 클러스터 스냅샷을 새 스냅샷에 복사합니다. 그런 다음 대상 AWS 계정에 공유합니다.

스냅샷을 복사하여 공유하는 방법

1. AWS 계정 소스에서 AWS Management Console에 로그인합니다.
2. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
3. 탐색 창에서 [스냅샷(Snapshots)]을 선택합니다.
4. 복사하려는 DB 클러스터 스냅샷을 선택합니다.
5. [작업(Actions)]에서 [스냅샷 복사(Copy snapshot)]를 선택합니다.
6. 스냅샷 복사 페이지에서 다음 단계를 따릅니다.

   a. 대상 리전에 이전 절차에서 고객 관리 키를 생성한 AWS 리전을 선택합니다.
   b. 새 DB 스냅샷 식별자에 DB 클러스터 스냅샷 복사본의 이름을 입력합니다.
   c. AWS KMS key에 생성한 고객 관리 키를 선택합니다.



d. 스냅샷 복사를 선택합니다.

7. 스냅샷 사본을 사용할 수 있는 경우 해당 복사본을 선택합니다.

8. Actions(작업)에서 Share snapshot(스냅샷 공유)을 선택합니다.

9. 스냅샷 권한 페이지에서 다음 단계를 따릅니다.

a. 스냅샷 사본을 공유하는 데 사용할 AWS 계정 ID를 입력한 다음 추가를 선택합니다.

b. Save(저장)를 선택합니다.

스냅샷이 복사됩니다.

공유된 스냅샷을 대상 계정에 복사

이제 대상 AWS 계정에서 공유 스냅샷을 복사할 수 있습니다.

공유된 스냅샷을 복사하는 방법

1. 대상 AWS 계정에서 AWS Management Console에 로그인합니다.
2. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
3. 탐색 창에서 [스냅샷(Snapshots)]을 선택합니다.
4. 나와 공유됨 탭을 선택합니다.
5. 공유된 스냅샷을 선택합니다.
6. [작업(Actions)]에서 [스냅샷 복사(Copy snapshot)]를 선택합니다.
7. 이전 절차에서와 같이 스냅샷 복사 설정을 선택하고 대상 계정에 속한 AWS KMS key를 사용합니다.

   스냅샷 복사를 선택합니다.

스냅샷 공유 중지

DB 클러스터 스냅샷 공유를 중지하려면 대상 AWS 계정에서 권한을 제거해야 합니다.

콘솔

AWS 계정에 대한 수동 DB 클러스터 스냅샷 공유를 중지하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 공유를 중지할 수동 스냅샷을 선택합니다.
4. Actions(작업)를 선택한 다음 Share snapshot(스냅샷 공유)을 선택합니다.

5. AWS 계정의 권한을 제거하려면 권한이 있는 계정 목록에서 해당 계정의 AWS 계정 식별자에 대해 삭제를 선택합니다.
6. 저장을 선택하여 변경 사항을 저장합니다.

CLI

목록에서 AWS 계정 식별자를 제거하려면 `--values-to-remove` 파라미터를 사용합니다.

Example 스냅샷 공유 중지

다음은 AWS 계정 ID 444455556666의 스냅샷 복원을 방지하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

Windows의 경우:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

AWS 계정의 공유 권한을 제거하려면 `AttributeName`을 `restore`로 설정한 `ModifyDBClusterSnapshotAttribute` 작업과 `ValuesToRemove` 파라미터를 사용합니다. 수동 스냅샷을 비공개로 하려면 `all` 속성에 대한 값 목록에서 `restore` 값을 제거합니다.

Amazon S3로 DB 클러스터 데이터 내보내기

라이브 Amazon Aurora DB 클러스터의 데이터를 Amazon S3 버킷으로 내보낼 수 있습니다. 내보내기 프로세스는 백그라운드에서 실행되며 활성 DB 클러스터의 성능에는 영향을 주지 않습니다.

기본적으로 DB 클러스터의 모든 데이터를 내보냅니다. 그러나 특정한 데이터베이스, 스키마 또는 테이블 집합을 내보내도록 선택할 수 있습니다.

Amazon Aurora는 DB 클러스터를 복제하고, 복제복에서 데이터를 추출하고, Amazon S3 버킷에 데이터를 저장합니다. 데이터는 압축되고 일관된 Apache Parquet 형식으로 저장됩니다. 개별 Parquet 파일은 일반적으로 크기가 1~10MB입니다.

Aurora MySQL 버전 2 및 버전 3용 스냅샷 데이터를 내보낼 때 제공되는 더 빠른 성능은 DB 클러스터 데이터 내보내기에는 적용되지 않습니다. 자세한 내용은 Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기 단원을 참조하십시오.

전체 데이터를 내보내든 일부 데이터를 내보내든 관계없이 전체 DB 클러스터를 내보내는 데 요금이 부과됩니다. 자세한 내용은 Amazon Aurora 요금 페이지를 참조하세요.

데이터를 내보낸 후에는 Amazon Athena 또는 Amazon Redshift Spectrum 같은 도구를 통해 직접 내보낸 데이터를 분석할 수 있습니다. Parquet 데이터를 읽는 데 Athena를 사용하는 방법에 대한 자세한 내용은 Amazon Athena 사용 설명서의 Parquet SerDe를 참조하세요. Parquet 데이터를 읽는 데 Redshift Spectrum을 사용하는 방법에 대한 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서에서 열 기반 데이터 형식의 COPY를 참조하세요.

기능 가용성 및 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. DB 클러스터 데이터를 S3로 내보낼 때 사용할 수 있는 버전 및 리전에 대한 자세한 내용은 Amazon S3로 클러스터 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진을 참조하세요.

다음 프로세스를 사용하여 DB 클러스터 데이터를 Amazon S3 버킷으로 내보냅니다. 자세한 내용은 다음 섹션을 참조하십시오.

DB 클러스터 데이터 내보내기 개요

1. 데이터를 내보낼 DB 클러스터를 식별합니다.
2. Amazon S3 버킷에 대한 액세스를 설정합니다.

   버킷은 Amazon S3 객체 또는 파일에 대한 컨테이너입니다. 버킷에 액세스하기 위한 정보를 제공하려면 다음 단계를 수행하십시오.

   a. DB 클러스터 데이터를 내보낼 S3 버킷을 식별합니다. S3 버킷이 DB 클러스터와 동일한 AWS 리전에 있어야 합니다. 자세한 내용은 내보낼 Amazon S3 버킷 식별 단원을 참조하십시오.
   b. DB 클러스터 내보내기 작업에 S3 버킷에 대한 액세스 권한을 부여하는 AWS Identity and Access Management(IAM) 역할을 생성합니다. 자세한 내용은 IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공 단원을 참조하십시오.
3. 서버 측 암호화를 위한 대칭 암호화 AWS KMS key를 생성합니다. KMS 키는 클러스터 내보내기 작업에서 S3에 내보내기 데이터를 기록할 때 AWS KMS 서버 측 암호화를 설정하는 데 사용됩니다.

   KMS 키 정책에는 `kms:CreateGrant` 및 `kms:DescribeKey` 권한이 모두 포함되어야 합니다. Amazon Aurora에서 KMS 키를 사용하는 자세한 방법은 AWS KMS key 관리 섹션을 참조하세요.

   KMS 키 정책에 거부 문이 있는 경우 AWS 서비스 보안 주체 `export.rds.amazonaws.com`을 명시적으로 제외해야 합니다.

   AWS 계정 안에서 KMS 키를 사용할 수 있으며, 또는 교차 계정 KMS 키를 사용할 수 있습니다. 자세한 내용은 교차 계정 AWS KMS key 사용 단원을 참조하십시오.
4. 콘솔 또는 `start-export-task` CLI 명령을 사용하여 Amazon S3로 DB 클러스터를 내보냅니다. 자세한 내용은 DB 클러스터 내보내기 작업 생성 단원을 참조하십시오.
5. Amazon S3 버킷에서 내보낸 데이터에 액세스하려면 Amazon Simple Storage Service 사용 설명서의 객체 업로드, 다운로드 및 관리를 참조하세요.

다음 섹션에서 DB 클러스터 내보내기 작업을 설정, 내보내기, 모니터링 및 취소하고 문제를 해결하는 방법을 알아봅니다.

주제

- DB 클러스터 내보내기 고려 사항
- Amazon S3 버킷에 대한 액세스 권한 설정
- DB 클러스터 내보내기 작업 생성
- DB 클러스터 내보내기 작업 모니터링
- DB 클러스터 내보내기 작업 취소
- DB 클러스터 내보내기 문제 해결

DB 클러스터 내보내기 고려 사항

다음 섹션을 바탕으로 DB 클러스터 데이터를 Amazon S3로 내보낼 때의 제한, 파일 명명 규칙, 데이터 변환 및 스토리지에 대해 알아보세요.

주제

제한 사항

DB 클러스터 데이터를 Amazon S3로 내보내는 데는 다음과 같은 제한 사항이 적용됩니다.

- 동일한 DB 클러스터에 대해 여러 내보내기 작업을 동시에 실행할 수 없습니다. 이는 전체 및 부분 내보내기에 모두 적용됩니다.
- AWS 계정당 최대 5개의 DB 스냅샷 내보내기 작업을 동시에 수행할 수 있습니다.
- Aurora Serverless v1 DB 클러스터는 S3로의 내보내기를 지원하지 않습니다.
- Aurora MySQL 및 Aurora PostgreSQL은 프로비저닝된 엔진 모드에서만 S3로의 내보내기를 지원합니다.
- S3로 내보내는 경우 콜론(:)이 포함된 S3 접두사를 지원하지 않습니다.
- 내보내는 동안 S3 파일 경로의 다음 문자는 밑줄(_)로 변환됩니다.

```
\ ` " (space)
```

- 데이터베이스, 스키마 또는 테이블의 이름에 다음 문자가 아닌 문자가 있으면 부분 내보내기가 지원되지 않습니다. 그러나 전체 DB 클러스터를 내보낼 수는 있습니다.
  - 라틴 문자(A–Z)
  - 숫자(0–9)
  - 달러 기호($)
  - 밑줄(_)
- 공백( )과 특정 문자는 데이터베이스 테이블 열 이름에서 지원되지 않습니다. 열 이름에 다음 문자가 포함되어 있는 테이블은 내보내기를 수행하는 동안 건너뛰기가 됩니다.

```
, ; { } ( ) \n \t = (space)
```

- 이름에 슬래시(/)가 포함되어 있는 테이블은 내보내기를 수행하는 동안 생략됩니다.
- Aurora PostgreSQL 임시 테이블과 로깅되지 않는 테이블은 내보내기 중에 건너뛰게 됩니다.
- 데이터에 500MB에 근접하거나 이보다 큰 객체(예: BLOB 또는 CLOB)가 포함되어 있으면 내보내기가 실패합니다.
- 테이블에 2GB에 가깝거나 그보다 큰 행이 있으면 내보내기 중 테이블을 건너뜁니다.
- 부분 내보내기의 경우 `ExportOnly` 목록의 최대 크기는 200KB입니다.
- 각 내보내기 작업에 고유한 이름을 사용하는 것이 좋습니다. 고유한 작업 이름을 사용하지 않으면 다음 오류 메시지가 표시될 수 있습니다.

  ExportTaskAlreadyExistsFault: StartExportTask 작업을 호출하는 동안 오류(ExportTaskAlreadyExists)가 발생했습니다. ID가 *xxxxx*인 내보내기 작업이 이미 존재합니다.
- 일부 테이블은 건너뛸 수 있으므로 내보낸 후 데이터의 행 및 테이블 수를 확인하는 것이 좋습니다.

파일 명명 규칙

특정 테이블에 대해 내보낸 데이터는 *base_prefix*/*files* 형식으로 저장됩니다. 여기서 기본 접두사는 다음과 같습니다.

```
export_identifier/database_name/schema_name.table_name/
```

예:

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

출력 파일은 다음과 같은 명명 규칙을 사용합니다. 여기서 *partition_index*는 영숫자입니다.

```
partition_index/part-00000-random_uuid.format-based_extension
```

예:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
    a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

파일 명명 규칙은 변경될 수 있습니다. 따라서 대상 테이블을 읽을 때 테이블의 기본 접두사 내에 있는 모든 내용을 읽는 것이 좋습니다.

데이터 변환 및 저장 형식

DB 클러스터를 Amazon S3 버킷으로 내보낼 때 Amazon Aurora는 데이터를 Parquet 형식으로 변환하고 내보내며 저장합니다. 자세한 내용은 Amazon S3 버킷으로 내보내기를 할 때 데이터 변환 단원을 참조하십시오.

Amazon S3 버킷에 대한 액세스 권한 설정

Amazon S3 버킷을 식별한 다음 버킷에 액세스할 수 있는 권한을 DB 클러스터 작업에 부여합니다.

주제

- 내보낼 Amazon S3 버킷 식별
- IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공
- 교차 계정 Amazon S3 버킷 사용하기

내보낼 Amazon S3 버킷 식별

DB 클러스터 데이터를 내보낼 Amazon S3 버킷을 식별합니다. 기존 S3 버킷을 사용하거나 새 S3 버킷을 생성합니다.

> **Note**
> S3 버킷이 DB 클러스터와 동일한 AWS 리전에 있어야 합니다.

Amazon S3 버킷 작업에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 다음을 참조하세요.

- S3 버킷에 대한 속성을 보려면 어떻게 해야 합니까?
- Amazon S3 버킷의 기본 암호화를 활성화하려면 어떻게 해야 합니까?
- S3 버킷을 생성하려면 어떻게 해야 합니까?

IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공

DB 클러스터 데이터를 Amazon S3로 내보내기 전에 내보내기 작업에 Amazon S3 버킷에 대한 쓰기-액세스 권한을 부여하십시오.

이 권한을 부여하려면 버킷에 대한 액세스 권한을 부여하는 IAM 정책을 생성한 다음 IAM 역할을 생성하고 정책을 역할에 연결해야 합니다. 나중에 DB 클러스터 내보내기 작업에 IAM 역할을 할당할 수 있습니다.

> **Important**
>
> AWS Management Console을 사용하여 DB 클러스터를 내보내려는 경우 DB 클러스터를 내보낼 때 IAM 정책 및 역할을 자동으로 생성하도록 선택할 수 있습니다. 지침은 DB 클러스터 내보내기 작업 생성 단원을 참조하십시오.

작업에 Amazon S3에 대한 액세스 권한을 부여하는 방법

1. IAM 정책을 생성합니다. 이 정책은 DB 클러스터 내보내기 작업에서 Amazon S3 액세스를 허용하는 버킷 및 객체 권한을 제공합니다.

   정책에 다음 필수 작업을 포함하여 Amazon Aurora에서 S3 버킷으로의 파일 전송을 허용합니다.

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   정책에 다음 리소스를 포함하여 버킷에 있는 S3 버킷과 객체를 식별합니다. 다음 리소스 목록은 Amazon S3에 액세스하기 위한 Amazon 리소스 이름(ARN) 형식을 보여 줍니다.

   - `arn:aws:s3:::`*`amzn-s3-demo-bucket`*
   - `arn:aws:s3:::`*`amzn-s3-demo-bucket`*`/*`

   Amazon Aurora에 대한 IAM 정책을 생성하는 방법에 대한 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 섹션을 참조하세요. IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결도 참조하세요.

   다음 AWS CLI 명령은 이 옵션으로 `ExportPolicy`라는 IAM 정책을 만듭니다. *`amzn-s3-demo-bucket`*이라는 버킷에 대한 액세스 권한을 부여합니다.

> **Note**
>
> 정책을 생성한 후 정책의 ARN을 기록해 둡니다. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Aurora가 사용자 대신 이 IAM 역할을 수임하여 Amazon S3 버킷에 액세스할 수 있도록 IAM 역할을 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자에게 권한을 위임하기 위한 역할 생성](#)을 참조하세요.

   다음 예제에서는 AWS CLI 명령을 사용해 `rds-s3-export-role`이라는 역할을 생성하는 방법을 보여줍니다.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
```

```
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 생성한 IAM 역할에 생성한 IAM 정책을 연결합니다.

   다음 AWS CLI 명령은 앞서 생성한 정책을 `rds-s3-export-role`이라는 역할에 연결합니다. *`your-policy-arn`*을 이전 단계에서 기록해 둔 정책 ARN으로 바꿉니다.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

교차 계정 Amazon S3 버킷 사용하기

S3 버킷 교차 AWS 계정을 사용할 수 있습니다. 자세한 내용은 교차 계정 Amazon S3 버킷 사용하기 단원을 참조하십시오.

DB 클러스터 내보내기 작업 생성

내보내기 작업을 생성하여 Aurora DB 클러스터의 데이터를 Amazon S3 버킷으로 내보냅니다. AWS 계정당 최대 5개의 DB 클러스터 내보내기 작업을 동시에 수행할 수 있습니다.

> **Note**
>
> DB 클러스터 데이터 내보내기는 데이터베이스 유형 및 크기에 따라 다소 시간이 걸릴 수 있습니다. 내보내기 작업은 먼저 전체 데이터베이스를 복제하고 크기를 조정한 다음 Amazon S3로 데이터를 추출합니다. 이 단계 동안의 작업 진행 상황은 Starting으로 표시됩니다. 작업이 S3로 데이터 내보내기로 전환되면 진행 상황이 진행 중(In progress)으로 표시됩니다.
> 내보내기를 완료하는 데 걸리는 시간은 데이터베이스에 저장된 데이터에 따라 다릅니다. 예를 들어 숫자로 된 기본 키 또는 인덱스 열이 잘 분산되어 있는 테이블은 가장 빠르게 내보냅니다. 분할에 적합한 열을 포함하지 않는 테이블과 문자열 기반 열에 인덱스가 하나만 있는 테이블은 내보내기가 더 느린 단일 스레드 프로세스를 사용하기 때문에 더 오래 걸립니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 데이터를 Amazon S3로 내보낼 수 있습니다.

Lambda 함수를 사용하여 DB 클러스터를 내보내는 경우 Lambda 함수 정책에 `kms:DescribeKey` 작업을 추가합니다. 자세한 내용은 AWS Lambda 권한을 참조하세요.

콘솔

Amazon S3로 내보내기 콘솔 옵션은 Amazon S3로 내보낼 수 있는 DB 클러스터에만 나타납니다. 다음과 같은 이유로 DB 클러스터를 내보내기에 사용하지 못할 수 있습니다.

- DB 엔진이 S3 내보내기를 지원하지 않습니다.
- DB 클러스터 버전이 S3 내보내기에 지원되지 않습니다.
- DB 클러스터가 생성된 AWS 리전에서 S3 내보내기가 지원되지 않습니다.

DB 클러스터 데이터를 내보내는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 데이터를 내보낼 DB 클러스터를 선택합니다.
4. 작업에서 Export to Amazon S3(Amazon S3로 내보내기)를 선택합니다.

   Export to Amazon S3(Amazon S3로 내보내기) 창이 나타납니다.
5. Export identifier(내보내기 식별자)에 내보내기 작업을 식별할 이름을 입력합니다. 이 값은 S3 버킷에 생성된 파일의 이름에도 사용됩니다.
6. 내보낼 데이터를 선택합니다.

   - DB 클러스터의 모든 데이터를 내보내려면 All(모두)을 선택합니다.
   - DB 클러스터의 특정 부분을 내보내려면 Partial(부분)을 선택합니다. 클러스터에서 내보낼 부분을 식별하려면 Identifiers(식별자)에 공백으로 구분된 하나 이상의 데이터베이스, 스키마 또는 테이블을 입력합니다.

     다음 형식을 사용합니다.

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
     [.tablen]
     ```

예:

```
mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
 mydatabase2.myschema2.mytable2
```

7. S3 버킷에서 내보낼 버킷을 선택합니다.

   내보낸 데이터를 S3 버킷의 폴더 경로에 할당하려면 S3 prefix(S3 접두사)에 선택적 경로를 입력합니다.

8. IAM 역할에서 선택한 S3 버킷에 대한 쓰기 액세스 권한을 부여하는 역할을 선택하거나 새 역할을 생성합니다.

   - IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공의 단계에 따라 역할을 생성한 경우에는 해당 역할을 선택합니다.
   - 선택한 S3 버킷에 대한 쓰기 액세스 권한을 부여하는 역할을 생성하지 않은 경우 Create a new role(새 역할 생성)을 선택하여 역할을 자동으로 생성합니다. 그런 다음 IAM 역할 이름에 해당 역할의 이름을 입력합니다.

9. KMS key(KMS 키)에 내보낸 데이터를 암호화하는 데 사용할 키의 ARN을 입력합니다.
10. Amazon S3로 내보내기를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터 데이터를 Amazon S3로 내보내려면 start-export-task 명령을 다음과 같은 필수 옵션과 함께 사용합니다.

- `--export-task-identifier`
- `--source-arn` – DB 클러스터의 Amazon 리소스 이름(ARN)입니다.
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

다음 예에서 내보내기 작업의 이름은 *my_cluster_export*이고, 이 작업은 데이터를 *amzn-s3-demo-destination-bucket*이라는 S3 버킷으로 내보냅니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Windows의 경우:

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

샘플 출력은 다음과 같습니다.

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

DB 클러스터 내보내기를 위해 S3 버킷에 폴더 경로를 제공하려면 start-export-task 명령에 `--s3-prefix` 옵션을 포함합니다.

RDS API

Amazon RDS API를 사용하여 DB 클러스터 데이터를 Amazon S3로 내보내려면 다음 필수 파라미터와 함께 StartExportTask 작업을 사용합니다.

- `ExportTaskIdentifier`
- `SourceArn` – DB 클러스터의 ARN입니다.
- `S3BucketName`
- `IamRoleArn`

- KmsKeyId

DB 클러스터 내보내기 작업 모니터링

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 내보내기를 모니터링할 수 있습니다.

콘솔

DB 클러스터 내보내기를 모니터링하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Exports in Amazon S3(Amazon S3에서 내보내기)를 선택합니다.

   DB 클러스터 내보내기는 Source type(소스 유형) 열에 표시됩니다. 내보내기 상태는 Status(상태) 열에 표시됩니다.
3. 특정 DB 클러스터 내보내기에 대한 상세 정보를 보려면 해당 내보내기 작업을 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터 내보내기 작업을 모니터링하려면 describe-export-tasks 명령을 사용합니다.

다음 예제에서는 모든 DB 클러스터 내보내기에 대한 현재 정보를 표시하는 방법을 보여 줍니다.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
```

```
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket amzn-s3-demo-bucket2 isn't located in the
 current AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

특정 내보내기 작업에 대한 정보를 표시하려면 `--export-task-identifier` 명령과 함께 `describe-export-tasks` 옵션을 포함시킵니다. 출력을 필터링하려면 `--Filters` 옵션을 포함시킵니다. 자세한 옵션은 describe-export-tasks 명령을 참조하십시오.

RDS API

Amazon RDS API를 사용하여 DB 클러스터 내보내기에 대한 정보를 표시하려면 DescribeExportTasks 작업을 사용합니다.

내보내기 워크플로우의 완료를 추적하거나 다른 워크플로우를 시작하려면 Amazon Simple Notification Service 주제를 구독하십시오. Amazon SNS에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 단원을 참조하십시오.

DB 클러스터 내보내기 작업 취소

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 내보내기 작업을 취소할 수 있습니다.

> **Note**
> 내보내기 작업을 취소해도 Amazon S3로 내보낸 데이터는 제거되지 않습니다. 콘솔을 사용하여 데이터를 삭제하는 방법에 대한 자세한 내용은 S3 버킷에서 객체를 삭제하려면 어떻게 해야 합니까?를 참조하십시오. CLI를 사용하여 데이터를 삭제하려면 delete-object 명령을 사용합니다.

콘솔

DB 클러스터 내보내기 작업을 취소하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Exports in Amazon S3(Amazon S3에서 내보내기)를 선택합니다.

   DB 클러스터 내보내기는 Source type(소스 유형) 열에 표시됩니다. 내보내기 상태는 Status(상태) 열에 표시됩니다.
3. 취소할 내보내기 작업을 선택합니다.
4. 취소를 선택합니다.
5. 확인 페이지에서 Cancel export task(내보내기 작업 취소)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 내보내기 작업을 취소하려면 cancel-export-task 명령을 사용합니다. 이 명령에는 `--export-task-identifier` 옵션이 필요합니다.

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS API

Amazon RDS API를 사용하여 내보내기 작업을 취소하려면 `ExportTaskIdentifier` 파라미터와 함께 CancelExportTask 작업을 사용합니다.

DB 클러스터 내보내기 문제 해결

다음 섹션을 사용하면 Amazon S3로의 DB 클러스터 내보내기 작업에 대한 실패 메시지 및 PostgreSQL 권한 오류 문제를 해결하는 데 도움이 됩니다.

Amazon S3 내보내기 작업에 대한 실패 메시지

다음 표에서는 Amazon S3 내보내기 작업이 실패할 때 반환되는 메시지에 대해 설명합니다.

오류 메시지	설명
소스 DB 클러스터([클러스터 이름])를 찾거나 액세스하지 못했습니다.	소스 DB 클러스터는 복제할 수 없습니다.
알 수 없는 내부 오류가 발생했습니다.	알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.

오류 메시지	설명
내보내기 작업의 메타데이터를 S3 버킷 [버킷 이름]에 쓰는 중에 알 수 없는 내부 오류가 발생했습니다.	알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.
RDS 내보내기가 IAM 역할 [역할 ARN]을 수임할 수 없기 때문에 내보내기 작업의 메타데이터를 작성하지 못했습니다.	내보내기 작업은 S3 버킷에 메타데이터를 쓸 수 있는지 여부를 검증하는 IAM 역할을 가정합니다. 작업이 IAM 역할을 맡을 수 없으면 실패합니다.
RDS 내보내기가 KMS 키 [키 ID] 와 함께 IAM 역할[역할 ARN]을 사용하여 S3 버킷 [버킷 이름]에 내보내기 작업의 메타데이터를 쓰지 못했습니다. 오류 코드: [오류 코드]	하나 이상의 권한이 누락되어 내보내기 작업이 S3 버킷에 액세스할 수 없습니다. 이 실패 메시지는 다음 오류 코드 중 하나를 수신할 때 발생합니다. • `AWSSecurityTokenServiceException` , 오류 코드 `AccessDenied` • `AmazonS3Exception` , 오류 코드 `NoSuchBucket` , `AccessDenied` , `KMS.KMSInvalidStateException` , `403 Forbidden` 또는 `KMS.DisabledException` 이러한 오류 코드는 IAM 역할, S3 버킷 또는 KMS 키에 대한 설정이 잘못 구성되었음을 의미합니다.
IAM 역할 [역할 ARN]은 S3 버킷 [버킷 이름]에서 [S3 작업]을 호출할 권한이 없습니다. 권한을 검토하고 내보내기를 다시 시도해 주세요.	IAM 정책이 잘못 구성되었습니다. S3 버킷의 특정 S3 작업에 대한 권한이 없어 내보내기 작업이 실패하게 됩니다.
KMS 키 확인이 실패했습니다. KMS 키의 자격 증명을 확인하고 다시 시도해 주세요.	KMS 키 자격 증명 확인에 실패했습니다.
S3 자격 증명 확인이 실패했습니다. S3 버킷 및 IAM 정책에서 권한을 확인합니다.	S3 자격 증명 확인이 실패했습니다.

오류 메시지	설명
S3 버킷 [버킷 이름]이 유효하지 않습니다. 현재 AWS 리전에 위치하지 않거나 존재하지 않습니다. S3 버킷 이름을 검토하고 내보내기를 다시 시도해 주세요.	S3 버킷이 유효하지 않습니다.
S3 버킷 [버킷 이름]이 현재 AWS 리전에 위치하지 않습니다. S3 버킷 이름을 검토하고 내보내기를 다시 시도해 주세요.	S3 버킷이 잘못된 AWS 리전에 있습니다.

PostgreSQL 권한 오류 문제 해결

PostgreSQL 데이터베이스를 Amazon S3으로 내보낼 때 특정 테이블을 건너뛰었다는 `PERMISSIONS_DO_NOT_EXIST` 오류가 표시될 수 있습니다. 이 오류는 대부분 DB 클러스터를 생성할 때 지정한 수퍼유저에게 이러한 테이블에 액세스할 권한이 없을 때 발생합니다.

이 오류를 해결하려면 다음 명령을 실행합니다.

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

수퍼유저 권한에 대한 자세한 내용은 마스터 사용자 계정 권한 단원을 참조하십시오.

Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기

DB 클러스터 스냅샷 데이터를 Amazon S3 버킷으로 내보낼 수 있습니다. 내보내기 프로세스는 백그라운드에서 실행되며 활성 DB 클러스터의 성능에는 영향을 주지 않습니다.

DB 클러스터 스냅샷을 내보낼 때 Amazon Aurora는 해당 스냅샷에서 데이터를 추출하여 Amazon S3 버킷에 저장합니다. 수동 스냅샷 및 자동화된 시스템 스냅샷을 내보낼 수 있습니다. 기본적으로 스냅샷의 모든 데이터가 내보내집니다. 그러나 특정한 데이터베이스, 스키마 또는 테이블 집합을 내보내도록 선택할 수 있습니다.

> **Note**
>
> DB 클러스터 스냅샷에서 데이터를 내보내려면 스냅샷을 복원해야 합니다. 복원 시간은 사용 가능한 대역폭 대비 AWS 리전이 받는 네트워크 트래픽의 양 등 다양한 요인에 의해 영향을 받습니다. 트래픽이 갑자기 증가하면 완료 시간이 예상보다 길어질 수 있습니다.
> Aurora 데이터베이스의 S3 내보내기 시간을 줄일 수 있는 대안은 실시간 DB 클러스터를 S3로 내보내는 것입니다. DB 클러스터 내보내기는 스냅샷을 복원할 필요가 없기 때문에 DB 스냅샷 내보내기보다 시작 시간이 적게 듭니다. 자세한 내용은 Amazon S3로 DB 클러스터 데이터 내보내기 단원을 참조하십시오.

데이터는 압축되고 일관된 Apache Parquet 형식으로 저장됩니다. 개별 Parquet 파일은 일반적으로 크기가 1~10MB입니다.

데이터를 내보낸 후에는 Amazon Athena 또는 Amazon Redshift Spectrum 같은 도구를 통해 직접 내보낸 데이터를 분석할 수 있습니다. Parquet 데이터를 읽는 데 Athena를 사용하는 방법에 대한 자세한 내용은 Amazon Athena 사용 설명서의 Parquet SerDe를 참조하세요. Parquet 데이터를 읽는 데 Redshift Spectrum을 사용하는 방법에 대한 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서에서 열 기반 데이터 형식의 COPY를 참조하세요.

기능 가용성 및 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. DB 클러스터 스냅샷 데이터를 S3로 내보낼 때 사용할 수 있는 버전 및 리전에 대한 자세한 내용은 Amazon S3로 스냅샷 데이터 내보내기를 지원하는 리전 및 Aurora DB 엔진 단원을 참조하십시오.

다음 프로세스를 사용하여 DB 스냅샷 데이터를 Amazon S3 버킷으로 내보냅니다. 자세한 내용은 다음 섹션을 참조하십시오.

스냅샷 데이터 내보내기 개요

1. 내보낼 스냅샷을 식별합니다.

기존의 자동 또는 수동 스냅샷을 사용하거나 DB 인스턴스의 수동 스냅샷을 생성합니다.

2. Amazon S3 버킷에 대한 액세스를 설정합니다.

   버킷은 Amazon S3 객체 또는 파일에 대한 컨테이너입니다. 버킷에 액세스하기 위한 정보를 제공하려면 다음 단계를 수행하십시오.

   a. 스냅샷을 내보낼 S3 버킷을 식별합니다. S3 버킷은 스냅샷과 같은 AWS 리전에 있어야 합니다. 자세한 내용은 내보낼 Amazon S3 버킷 식별 단원을 참조하십시오.

   b. 스냅샷 내보내기 태스크에 S3 버킷에 대한 액세스 권한을 부여하는 AWS Identity and Access Management(IAM) 역할을 생성합니다. 자세한 내용은 IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공 단원을 참조하십시오.

3. 서버 측 암호화를 위한 대칭 암호화 AWS KMS key를 생성합니다. KMS 키는 스냅샷 내보내기 작업에서 S3에 내보내기 데이터를 기록할 때 AWS KMS 서버 측 암호화를 설정하는 데 사용됩니다.

   KMS 키 정책에는 `kms:CreateGrant` 및 `kms:DescribeKey` 권한이 모두 포함되어야 합니다. Amazon Aurora에서 KMS 키를 사용하는 자세한 방법은 AWS KMS key 관리 섹션을 참조하세요.

   KMS 키 정책에 거부 문이 있는 경우 AWS 서비스 보안 주체 `export.rds.amazonaws.com`을 명시적으로 제외해야 합니다.

   AWS 계정 안에서 KMS 키를 사용할 수 있으며, 또는 교차 계정 KMS 키를 사용할 수 있습니다. 자세한 내용은 교차 계정 AWS KMS key 사용 단원을 참조하십시오.

4. 콘솔 또는 `start-export-task` CLI 명령을 사용하여 Amazon S3로 스냅샷을 내보냅니다. 자세한 내용은 스냅샷 내보내기 작업 생성 단원을 참조하십시오.

5. Amazon S3 버킷에서 내보낸 데이터에 액세스하려면 Amazon Simple Storage Service 사용 설명서의 객체 업로드, 다운로드 및 관리를 참조하세요.

다음 섹션에서 DB 클러스터 스냅샷 내보내기 작업을 설정, 내보내기, 모니터링 및 취소하고 문제를 해결하는 방법을 알아봅니다.

주제

- DB 클러스터 스냅샷 내보내기 고려 사항
- Amazon S3 버킷에 대한 액세스 권한 설정
- 스냅샷 내보내기 작업 생성
- 스냅샷 내보내기 모니터링

- [스냅샷 내보내기 작업 취소]
- [Aurora MySQL에서의 내보내기 성능]
- [스냅샷 내보내기 문제 해결]

DB 클러스터 스냅샷 내보내기 고려 사항

제한 사항

DB 스냅샷 데이터를 Amazon S3로 내보내는 데는 다음과 같은 제한 사항이 적용됩니다.

- 동일한 DB 클러스터 스냅샷에 대해 여러 내보내기 작업을 동시에 실행할 수 없습니다. 이는 전체 및 부분 내보내기에 모두 적용됩니다.
- AWS 계정당 최대 5개의 DB 스냅샷 내보내기 작업을 동시에 수행할 수 있습니다.
- 스냅샷 데이터를 Aurora Serverless v1 DB 클러스터에서 S3으로 내보낼 수 없습니다.
- S3로 내보내는 경우 콜론(:)이 포함된 S3 접두사를 지원하지 않습니다.
- 내보내는 동안 S3 파일 경로의 다음 문자는 밑줄(_)로 변환됩니다.

```
\ ` " (space)
```

- 데이터베이스, 스키마 또는 테이블의 이름에 다음 문자가 아닌 문자가 있으면 부분 내보내기가 지원되지 않습니다. 그러나 전체 DB 스냅샷을 내보낼 수는 있습니다.
  - 라틴 문자(A–Z)
  - 숫자(0–9)
  - 달러 기호($)
  - 밑줄(_)
- 공백( )과 특정 문자는 데이터베이스 테이블 열 이름에서 지원되지 않습니다. 열 이름에 다음 문자가 포함되어 있는 테이블은 내보내기를 수행하는 동안 건너뛰기가 됩니다.

```
, ; { } ( ) \n \t = (space)
```

- 이름에 슬래시(/)가 포함되어 있는 테이블은 내보내기를 수행하는 동안 생략됩니다.
- Aurora PostgreSQL 임시 테이블과 로깅되지 않는 테이블은 내보내기 중에 건너뛰게 됩니다.
- 데이터에 500MB에 근접하거나 이보다 큰 객체(예: BLOB 또는 CLOB)가 포함되어 있으면 내보내기가 실패합니다.
- 테이블에 2GB에 가깝거나 그보다 큰 행이 있으면 내보내기 중 테이블을 건너뜁니다.

- 부분 내보내기의 경우 `ExportOnly` 목록의 최대 크기는 200KB입니다.
- 각 내보내기 작업에 고유한 이름을 사용하는 것이 좋습니다. 고유한 작업 이름을 사용하지 않으면 다음 오류 메시지가 표시될 수 있습니다.

  ExportTaskAlreadyExistsFault: StartExportTask 작업을 호출하는 동안 오류 (ExportTaskAlreadyExists)가 발생했습니다. ID가 *xxxxx*인 내보내기 작업이 이미 존재합니다.
- 데이터를 S3로 내보내는 동안 스냅샷을 삭제할 수 있지만, 내보내기 태스크가 완료될 때까지 해당 스냅샷에 대한 스토리지 비용은 계속 청구됩니다.
- 내보낸 스냅샷 데이터를 S3에서 새 DB 클러스터로 복원할 수 없습니다.

파일 명명 규칙

특정 테이블에 대해 내보낸 데이터는 *base_prefix*/*files* 형식으로 저장됩니다. 여기서 기본 접두사는 다음과 같습니다.

```
export_identifier/database_name/schema_name.table_name/
```

예:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

파일의 명명 규칙은 두 가지가 있습니다.

- 현행 규칙:

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  배치 인덱스는 테이블에서 읽은 데이터 배치를 나타내는 시퀀스 번호입니다. 테이블을 병렬로 내보낼 작은 청크로 파티셔닝할 수 없는 경우에는 배치 인덱스가 여러 개 생성됩니다. 테이블이 여러 테이블로 파티셔닝된 경우에도 마찬가지입니다. 기본 테이블의 각 테이블 파티션마다 하나씩, 배치 인덱스가 여러 개 생성됩니다.

  테이블을 병렬로 읽을 작은 청크로 파티셔닝할 수 있는 경우에는 배치 인덱스 1 폴더 하나만 생성됩니다.

  배치 인덱스 폴더에는 테이블 데이터를 포함하는 하나 이상의 Parquet 파일이 있습니다. Parquet 파일 이름의 접두사는 `part-`*partition_index*입니다. 테이블이 파티셔닝된 경우 파티션 인덱스 `00000`으로 시작되는 여러 개의 파일이 생성됩니다.

파티션 인덱스 시퀀스에 간격이 있을 수 있습니다. 이러한 간격은 테이블의 범위 지정 쿼리에서 각 파티션을 가져오기 때문에 발생합니다. 해당 파티션 범위에 데이터가 없는 경우 해당 시퀀스 번호는 건너뛰게 됩니다.

예를 들어 `id`열이 테이블의 프라이머리 키이고 최소값과 최대값이 `100` 및 `1000`이라고 가정해 보겠습니다. 9개의 파티션이 있는 이 테이블을 내보내려고 하면 다음과 같은 병렬 쿼리를 사용하여 테이블을 읽습니다.

```
SELECT * FROM table WHERE id <= 100 AND id < 200
 SELECT * FROM table WHERE id <= 200 AND id < 300
```

여기에서 `part-00000-random_uuid.gz.parquet`~`part-00008-random_uuid.gz.parquet`에 해당하는 9개의 파일이 생성됩니다. 그러나 `200`~`350` 사이의 ID가 있는 행이 없으면 완료된 파티션 중 하나가 비어 있게 되고 해당 파티션에 대한 파일이 생성되지 않습니다. 이전 예제에서는 `part-00001-random_uuid.gz.parquet`이 생성되지 않았습니다.

- 이전 규칙:

```
part-partition_index-random_uuid.format-based_extension
```

이는 현재 규칙과 동일하지만 *batch_index* 접두사는 없습니다. 예를 들면 다음과 같습니다.

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
 part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
 part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

파일 명명 규칙은 변경될 수 있습니다. 따라서 대상 테이블을 읽을 때 테이블의 기본 접두사 내에 있는 모든 내용을 읽는 것이 좋습니다.

Amazon S3 버킷으로 내보내기를 할 때 데이터 변환

DB 스냅샷을 Amazon S3 버킷으로 내보낼 때 Amazon Aurora는 데이터를 Parquet 형식으로 변환하고, 데이터를 Parquet 형식으로 내보내며, 데이터를 Parquet 형식으로 저장합니다. Parquet에 대한 자세한 내용은 Apache Parquet 웹 사이트를 참조하세요.

Apache Parquet은 다음과 같은 프리미티브 유형 중 하나로 모든 데이터를 저장합니다.

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – 가변 길이 바이트 배열(이진수라고도 함)
- FIXED_LEN_BYTE_ARRAY – 값이 일정한 크기를 가질 때 사용되는 고정 길이 바이트 배열

Parquet 데이터 유형은 형식을 읽고 쓸 때 복잡성을 줄이기 위한 것입니다. Parquet은 프리미티브 유형을 확장할 수 있도록 논리적 유형을 제공합니다. 논리적 유형은 `LogicalType` 메타 데이터 필드의 데이터를 이용으로 주석으로 구현됩니다. 논리적 유형 주석은 프리미티브 유형을 해석하는 방법을 설명합니다.

`STRING` 논리적 유형에 `BYTE_ARRAY` 유형이 주석으로 달려 있으면 바이트 배열이 UTF-8로 인코딩된 문자열로 해석되어야 한다는 뜻입니다. 내보내기 작업이 완료되면 Amazon Aurora에서 문자열 변환이 발생했는지를 알려줍니다. 내보낸 기초 데이터는 항상 소스의 데이터와 동일합니다. 그러나 UTF-8에서의 인코딩 차이로 인해 일부 문자는 Athena 같은 도구에서 읽을 때 소스와 다르게 나타날 수 있습니다.

자세한 내용은 Parquet 설명서의 [Parquet 논리적 유형 정의](#)를 참고하세요.

주제

- [Parquet로 MySQL 데이터 유형 매핑](#)
- [Parquet로 PostgreSQL 데이터 유형 매핑](#)

Parquet로 MySQL 데이터 유형 매핑

다음 테이블은 데이터를 변환하여 Amazon S3로 내보낼 때 MySQL 데이터 유형에서 Parquet 데이터 유형으로의 매핑을 보여줍니다.

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
숫자 데이터 유형			

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN _BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet는 서명된 유형만 지원하므로 매핑에는 BIGINT_UN SIGNED 유형을 저장하기 위해 추가 바이트 (8 + 1)가 필요합니다.
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL (p,s)	소스 값이 2^{31} 미만이면 INT32로 저장됩니다.
	INT64	DECIMAL (p,s)	소스 값이 2^{31} 이상이지만 2^{63} 미만인 경우 INT64로 저장됩니다.
	FIXED_LEN _BYTE_ARRAY(N)	DECIMAL (p,s)	소스 값이 2^{63} 이상이면 FIXED_LEN _BYTE_ARRAY(N)로 저장됩니다.
	BYTE_ARRAY	STRING	Parquet는 38보다 큰 십진수 정밀도를 지원하지 않습니다. 십진수 값은 BYTE_ARRA Y 유형의 문자열로 변환되고 UTF8로 인코딩됩니다.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL (p,s)	소스 값이 2^{31} 미만이면 INT32로 저장됩니다.
	INT64	DECIMAL (p,s)	소스 값이 2^{31} 이상이지만 2^{63} 미만인 경우 INT64로 저장됩니다.
	FIXED_LEN _ARRAY(N)	DECIMAL (p,s)	소스 값이 2^{63} 이상이면 FIXED_LEN _BYTE_ARRAY(N)로 저장됩니다.
	BYTE_ARRAY	STRING	Parquet는 38보다 큰 숫자 정밀도를 지원하지 않습니다. 이 숫자 값은 BYTE_ARRAY 유형의 문자열로 변환되고 UTF8로 인코딩됩니다.
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
TINYINT	INT32		
TINYINT UNSIGNED	INT32	INT(16, true)	
문자열 데이터 유형			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
날짜 및 시간 데이터 유형			

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
DATE	BYTE_ARRAY	STRING	날짜는 BYTE_ARRAY 유형의 문자열로 변환되고 UTF8로 인코딩됩니다.
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	TIME 유형은 BYTE_ARRAY의 문자열로 변환되고 UTF8로 인코딩됩니다.
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
기하학적 데이터 유형			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON 데이터 유형			
JSON	BYTE_ARRAY	STRING	

Parquet로 PostgreSQL 데이터 유형 매핑

다음 표는 데이터를 변환해 Amazon S3로 내보낼 때 PostgreSQL 데이터 유형에서 Parquet 데이터 유형으로의 매핑을 보여줍니다.

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
숫자 데이터 유형			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	DECIMAL 유형은 BYTE_ARRAY 유형의 문자열로 변환되고 UTF8로 인코딩됩니다. 이 변환은 데이터 정밀도와 숫자(NaN)가 아닌 데이터 값으로 인해 복잡해지는 것을 피하기 위한 것입니다.
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT(16, true)	
SMALLSERIAL	INT32	INT(16, true)	

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
문자열 및 관련 데이터 유형			
ARRAY	BYTE_ARRAY	STRING	배열은 문자열로 변환되고 BINARY(UTF8)로 인코딩됩니다. 이 변환은 데이터 정밀도와 숫자(NaN)가 아닌 데이터 값, 그리고 시간 데이터 값으로 인해 복잡해지는 것을 피하기 위한 것입니다.
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
날짜 및 시간 데이터 유형			

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP(시간대 사용)	BYTE_ARRAY	STRING	
기하학적 데이터 유형			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON 데이터 유형			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
기타 데이터 유형			
BOOLEAN	BOOLEAN		

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
CIDR	BYTE_ARRAY	STRING	네트워크 데이터 유형
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	네트워크 데이터 유형
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	해당 사항 없음		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

Amazon S3 버킷에 대한 액세스 권한 설정

Amazon S3 버킷을 식별한 다음 버킷에 액세스할 수 있는 권한을 스냅샷에 부여합니다.

주제

- 내보낼 Amazon S3 버킷 식별
- IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공
- 교차 계정 Amazon S3 버킷 사용하기
- 교차 계정 AWS KMS key 사용

내보낼 Amazon S3 버킷 식별

DB 스냅샷을 내보낼 Amazon S3 버킷을 식별합니다. 기존 S3 버킷을 사용하거나 새 S3 버킷을 생성합니다.

> **Note**
>
> 내보낼 S3 버킷은 스냅샷과 동일한 AWS 리전에 있어야 합니다.

Amazon S3 버킷 작업에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 다음을 참조하세요.

- S3 버킷에 대한 속성을 보려면 어떻게 해야 합니까?
- Amazon S3 버킷의 기본 암호화를 활성화하려면 어떻게 해야 합니까?
- S3 버킷을 생성하려면 어떻게 해야 합니까?

IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공

DB 스냅샷 데이터를 Amazon S3로 내보내기 전에 스냅샷 내보내기 작업에 Amazon S3 버킷에 대한 쓰기-액세스 권한을 부여하십시오.

이 권한을 부여하려면 버킷에 대한 액세스 권한을 부여하는 IAM 정책을 생성한 다음 IAM 역할을 생성하고 정책을 역할에 연결해야 합니다. 나중에 스냅샷 내보내기 작업에 IAM 역할을 할당할 수 있습니다.

> **Important**
>
> AWS Management Console을 사용하여 스냅샷을 내보내려는 경우 스냅샷을 내보낼 때 IAM 정책 및 역할을 자동으로 생성하도록 선택할 수 있습니다. 지침은 스냅샷 내보내기 작업 생성 섹션을 참조하세요.

DB 스냅샷 작업에 Amazon S3에 대한 액세스 권한을 부여하려면

1. IAM 정책을 생성합니다. 이 정책은 스냅샷 내보내기 작업에서 Amazon S3 액세스를 허용하는 버킷 및 객체 권한을 제공합니다.

   정책에 다음 필수 작업을 포함하여 Amazon Aurora에서 S3 버킷으로의 파일 전송을 허용합니다.

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`

- s3:GetBucketLocation

정책에 다음 리소스를 포함하여 버킷에 있는 S3 버킷과 객체를 식별합니다. 다음 리소스 목록은 Amazon S3에 액세스하기 위한 Amazon 리소스 이름(ARN) 형식을 보여 줍니다.

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

Amazon Aurora에 대한 IAM 정책을 생성하는 방법에 대한 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 섹션을 참조하세요. IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결도 참조하세요.

다음 AWS CLI 명령은 이 옵션으로 ExportPolicy라는 IAM 정책을 만듭니다. *amzn-s3-demo-bucket*이라는 버킷에 대한 액세스 권한을 부여합니다.

> **Note**
>
> 정책을 생성한 후 정책의 ARN을 기록해 둡니다. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
```

```
    ]
}'
```

2. Aurora가 사용자 대신 이 IAM 역할을 수임하여 Amazon S3 버킷에 액세스할 수 있도록 IAM 역할을 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자에게 권한을 위임하기 위한 역할 생성](#)을 참조하세요.

   다음 예제에서는 AWS CLI 명령을 사용해 `rds-s3-export-role`이라는 역할을 생성하는 방법을 보여줍니다.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "export.rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole"
       }
     ]
   }'
```

3. 생성한 IAM 역할에 생성한 IAM 정책을 연결합니다.

   다음 AWS CLI 명령은 앞서 생성한 정책을 `rds-s3-export-role`이라는 역할에 연결합니다. *`your-policy-arn`*을 이전 단계에서 기록해 둔 정책 ARN으로 바꿉니다.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

교차 계정 Amazon S3 버킷 사용하기

Amazon S3 버킷 교차 AWS 계정을 사용할 수 있습니다. 교차 계정 버킷을 사용하려면 S3 내보내기에 사용 중인 IAM 역할에 대한 액세스를 허용하는 버킷 정책을 추가합니다. 자세한 내용은 [예제 2: 버킷 소유자가 교차 계정 버킷 권한 부여](#)를 참조하세요.

- 다음 예에 표시된 대로 버킷에 버킷 정책을 연결합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

교차 계정 AWS KMS key 사용

Amazon S3 내보내기 암호화를 위해 교차 계정 AWS KMS key를 사용할 수 있습니다. 먼저 로컬 계정에 키 정책을 추가한 다음 외부 계정에 IAM 정책을 추가합니다. 자세한 내용은 다른 계정의 사용자가 KMS 키를 사용하도록 허용을 참조하세요.

교차 계정 KMS 키 사용

1. 로컬 계정에 키 정책을 추가합니다.

   다음 예는 로컬 계정 123456789012의 권한을 외부 계정 444455556666의 `ExampleRole`과 `ExampleUser`에 부여합니다.

```
{
```

```
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. 외부 계정에서 IAM 정책 추가

   다음 IAM 정책 예시는 보안 주체가 계정 123456789012의 KMS 키를 암호화 작업에 사용하도록 허용합니다. 이 권한을 계정 444455556666의 `ExampleRole` 및 `ExampleUser`에 부여하려면 계정에서 정책을 연결합니다.

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

스냅샷 내보내기 작업 생성

스냅샷 내보내기 작업을 생성하여 스냅샷의 데이터를 Amazon S3 버킷으로 내보냅니다. AWS 계정당 최대 5개의 DB 스냅샷 내보내기 작업을 동시에 수행할 수 있습니다.

> **Note**
>
> RDS 스냅샷 내보내기는 데이터베이스 유형 및 크기에 따라 다소 시간이 걸릴 수 있습니다. 내보내기 작업은 먼저 전체 데이터베이스를 복원하고 크기를 조정하여 Amazon S3로 데이터를 추출합니다. 이 단계 동안의 작업 진행 상황은 Starting으로 표시됩니다. 작업이 S3로 데이터 내보내기로 전환되면 진행 상황이 진행 중(In progress)으로 표시됩니다.
> 내보내기를 완료하는 데 걸리는 시간은 데이터베이스에 저장된 데이터에 따라 다릅니다. 예를 들어 숫자로 된 기본 키 또는 인덱스 열이 잘 분산되어 있는 테이블은 가장 빠르게 내보냅니다. 분할에 적합한 열을 포함하지 않는 테이블과 문자열 기반 열에 인덱스가 하나만 있는 테이블은 더 오래 걸립니다. 내보내기가 느린 단일 스레드 프로세스를 사용하기 때문에 이렇게 더 긴 내보내기 시간이 발생합니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 스냅샷을 Amazon S3로 내보낼 수 있습니다.

Lambda 함수를 사용하여 스냅샷을 내보내는 경우 Lambda 함수 정책에 `kms:DescribeKey` 작업을 추가합니다. 자세한 내용은 AWS Lambda 권한을 참조하세요.

콘솔

Amazon S3로 내보내기 옵션은 Amazon S3로 내보낼 수 있는 스냅샷에 대해서만 나타납니다. 다음과 같은 이유로 스냅샷을 내보내기에 사용하지 못할 수 있습니다.

- DB 엔진이 S3 내보내기를 지원하지 않습니다.
- DB 인스턴스 버전이 S3 내보내기에 지원되지 않습니다.
- 스냅샷이 생성된 AWS 리전에서 S3 내보내기가 지원되지 않습니다.

DB 스냅샷을 내보내려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.

3. 이 탭에서 내보낼 스냅샷 유형을 선택합니다.
4. 스냅샷 목록에서 내보낼 스냅샷을 선택합니다.
5. 작업에서 Export to Amazon S3(Amazon S3로 내보내기)를 선택합니다.

   Export to Amazon S3(Amazon S3로 내보내기) 창이 나타납니다.
6. Export identifier(내보내기 식별자)에 내보내기 작업을 식별할 이름을 입력합니다. 이 값은 S3 버킷에 생성된 파일의 이름에도 사용됩니다.
7. 내보낼 데이터를 선택합니다.
   - 스냅샷의 모든 데이터를 내보내려면 모두를 선택합니다.
   - 스냅샷의 특정 부분을 내보내려면 Partial(부분)을 선택합니다. 스냅샷에서 내보낼 부분을 식별하려면 [식별자(Identifiers)]에 공백으로 구분된 하나 이상의 데이터베이스, 스키마 또는 테이블을 입력합니다.

     다음 형식을 사용합니다.

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
     [.tablen]
     ```

     예:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
8. S3 버킷에서 내보낼 버킷을 선택합니다.

   내보낸 데이터를 S3 버킷의 폴더 경로에 할당하려면 S3 prefix(S3 접두사)에 선택적 경로를 입력합니다.
9. IAM 역할에서 선택한 S3 버킷에 대한 쓰기 액세스 권한을 부여하는 역할을 선택하거나 새 역할을 생성합니다.
   - IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공의 단계에 따라 역할을 생성한 경우에는 해당 역할을 선택합니다.
   - 선택한 S3 버킷에 대한 쓰기 액세스 권한을 부여하는 역할을 생성하지 않은 경우 Create a new role(새 역할 생성)을 선택하여 역할을 자동으로 생성합니다. 그런 다음 IAM 역할 이름에 해당 역할의 이름을 입력합니다.
10. AWS KMS key에 내보낸 데이터를 암호화하는 데 사용할 키의 ARN을 입력합니다.

11. Amazon S3로 내보내기를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 스냅샷을 Amazon S3로 내보내려면 [start-export-task](https://docs.aws.amazon.com/cli/latest/reference/rds/start-export-task.html) 명령을 다음과 같은 필수 옵션과 함께 사용합니다.

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

다음 예에서 스냅샷 내보내기 작업의 이름은 *my_snapshot_export*이고, 이 작업은 스냅샷을 *amzn-s3-demo-destination-bucket*이라는 S3 버킷으로 내보냅니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Windows의 경우:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

샘플 출력은 다음과 같습니다.

```
{
```

```
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

스냅샷 내보내기를 위해 S3 버킷에 폴더 경로를 제공하려면 start-export-task 명령에 `--s3-prefix` 옵션을 포함시킵니다.

RDS API

Amazon RDS API를 사용하여 DB 스냅샷을 Amazon S3로 내보내려면 다음 필수 파라미터와 함께 StartExportTask 작업을 사용합니다.

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

스냅샷 내보내기 모니터링

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 스냅샷 내보내기를 모니터링할 수 있습니다.

콘솔

DB 스냅샷 내보내기를 모니터링하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Amazon S3에서 내보내기를 선택합니다.

DB 스냅샷 내보내기는 소스 유형 열에 표시됩니다. 내보내기 상태는 상태 열에 표시됩니다.

3. 특정 스냅샷 내보내기에 대한 상세 정보를 보려면 해당 내보내기 작업을 선택합니다.

AWS CLI

AWS CLI을 사용하여 DB 스냅샷 내보내기를 모니터링하려면 [describe-export-tasks](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-export-tasks.html) 명령을 사용합니다.

다음 예제에서는 모든 스냅샷 내보내기에 대한 현재 정보를 표시하는 방법을 보여 줍니다.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
```

```
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

특정 스냅샷 내보내기에 대한 정보를 표시하려면 `--export-task-identifier` 명령과 함께 `describe-export-tasks` 옵션을 포함시킵니다. 출력을 필터링하려면 `--Filters` 옵션을 포함시킵니다. 자세한 옵션은 describe-export-tasks 명령을 참조하십시오.

RDS API

Amazon RDS API를 사용하여 DB 스냅샷 내보내기에 대한 정보를 표시하려면 DescribeExportTasks 작업을 사용합니다.

내보내기 워크플로우의 완료를 추적하거나 다른 워크플로우를 시작하려면 Amazon Simple Notification Service 주제를 구독하십시오. Amazon SNS에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 단원을 참조하십시오.

스냅샷 내보내기 작업 취소

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 스냅샷 내보내기 작업을 취소할 수 있습니다.

> **Note**
>
> 스냅샷 내보내기 작업을 취소해도 Amazon S3로 내보낸 데이터는 제거되지 않습니다. 콘솔을 사용하여 데이터를 삭제하는 방법에 대한 자세한 내용은 S3 버킷에서 객체를 삭제하려면 어떻게 해야 합니까?를 참조하십시오. CLI를 사용하여 데이터를 삭제하려면 delete-object 명령을 사용합니다.

콘솔

스냅샷 내보내기 작업을 취소하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Amazon S3에서 내보내기를 선택합니다.

   DB 스냅샷 내보내기는 소스 유형 열에 표시됩니다. 내보내기 상태는 상태 열에 표시됩니다.
3. 취소할 스냅샷 내보내기 작업을 선택합니다.
4. 취소를 선택합니다.
5. 확인 페이지에서 Cancel export task(내보내기 작업 취소)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 스냅샷 내보내기 작업을 취소하려면 cancel-export-task 명령을 사용합니다. 이 명령에는 `--export-task-identifier` 옵션이 필요합니다.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
```

```
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

Amazon RDS API를 사용하여 스냅샷 내보내기 작업을 취소하려면 `ExportTaskIdentifier` 파라미터와 함께 CancelExportTask 작업을 사용합니다.

Aurora MySQL에서의 내보내기 성능

Aurora MySQL 버전 2 및 버전 3 DB 클러스터 스냅샷은 고급 내보내기 메커니즘을 사용하여 성능을 개선하고 내보내기 시간을 단축합니다. 이 메커니즘에는 Aurora 공유 스토리지 아키텍처를 활용하기 위한 최적화(예: 다중 내보내기 스레드 및 Aurora MySQL 병렬 쿼리)가 포함됩니다. 최적화는 데이터 세트 크기 및 구조에 맞게 적용됩니다.

병렬 쿼리를 켜지 않아도 더 빠른 내보내기 프로세스를 사용할 수 있지만, 프로세스에는 병렬 쿼리와 동일한 제한이 적용됩니다. 또한 일이 `0`이거나 연도가 `0000`인 날짜 같은 일부 데이터 값이 지원되지 않습니다. 자세한 내용은 Amazon Aurora MySQL용 병렬 쿼리 단원을 참조하십시오.

성능 최적화를 적용하면 훨씬 큰 Aurora MySQL 버전 2 및 3 내보내기용 Parquet 파일(최대 200GB)도 표시됩니다.

데이터 유형이나 값이 호환되지 않는 등의 이유로 더 빠른 내보내기 프로세스를 사용할 수 없다면, Aurora는 병렬 쿼리를 사용하지 않는 단일 스레드 내보내기 모드로 자동 전환됩니다. 사용하는 프로세스와 내보낼 데이터 양에 따라 내보내기 성능이 달라질 수 있습니다.

스냅샷 내보내기 문제 해결

다음 섹션을 사용하면 Amazon S3로의 DB 클러스터 내보내기 작업에 대한 실패 메시지 및 PostgreSQL 권한 오류 문제를 해결하는 데 도움이 됩니다.

Amazon S3 내보내기 작업에 대한 실패 메시지

다음 표에서는 Amazon S3 내보내기 작업이 실패할 때 반환되는 메시지에 대해 설명합니다.

오류 메시지	설명
알 수 없는 내부 오류가 발생했습니다.	알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.
내보내기 작업의 메타데이터를 S3 버킷 [버킷 이름]에 쓰는 중에 알 수 없는 내부 오류가 발생했습니다.	알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.
RDS 내보내기가 IAM 역할 [역할 ARN]을 수임할 수 없기 때문에 내보내기 작업의 메타데이터를 작성하지 못했습니다.	내보내기 작업은 S3 버킷에 메타데이터를 쓸 수 있는지 여부를 검증하는 IAM 역할을 가정합니다. 작업이 IAM 역할을 맡을 수 없으면 실패합니다.
RDS 내보내기가 KMS 키 [키 ID] 와 함께 IAM 역할[역할 ARN]을 사용하여 S3 버킷 [버킷 이름]에 내보내기 작업의 메타데이터를 쓰지 못했습니다. 오류 코드: [오류 코드]	하나 이상의 권한이 누락되어 내보내기 작업이 S3 버킷에 액세스할 수 없습니다. 이 실패 메시지는 다음 오류 코드 중 하나를 수신할 때 발생합니다. • `AWSSecurityTokenServiceException` , 오류 코드 `AccessDenied` • `AmazonS3Exception` , 오류 코드 `NoSuchBucket` , `AccessDenied` , `KMS.KMSInvalidStateException` , `403 Forbidden` 또는 `KMS.DisabledException` 이러한 오류 코드는 IAM 역할, S3 버킷 또는 KMS 키에 대한 설정이 잘못 구성되었음을 의미합니다.
IAM 역할 [역할 ARN]은 S3 버킷 [버킷 이름]에서 [S3 작업]을 호출할 권한이 없습니다. 권한을 검토하고 내보내기를 다시 시도해 주세요.	IAM 정책이 잘못 구성되었습니다. S3 버킷의 특정 S3 작업에 대한 권한이 없어 내보내기 작업이 실패하게 됩니다.
KMS 키 확인이 실패했습니다. KMS 키의 자격 증명을 확인하고 다시 시도해 주세요.	KMS 키 자격 증명 확인에 실패했습니다.

오류 메시지	설명
S3 자격 증명 확인이 실패했습니다. S3 버킷 및 IAM 정책에서 권한을 확인합니다.	S3 자격 증명 확인이 실패했습니다.
S3 버킷 [버킷 이름]이 유효하지 않습니다. 현재 AWS 리전에 위치하지 않거나 존재하지 않습니다. S3 버킷 이름을 검토하고 내보내기를 다시 시도해 주세요.	S3 버킷이 유효하지 않습니다.
S3 버킷 [버킷 이름]이 현재 AWS 리전에 위치하지 않습니다. S3 버킷 이름을 검토하고 내보내기를 다시 시도해 주세요.	S3 버킷이 잘못된 AWS 리전에 있습니다.

PostgreSQL 권한 오류 문제 해결

PostgreSQL 데이터베이스를 Amazon S3으로 내보낼 때 특정 테이블을 건너뛰었다는 `PERMISSIONS_DO_NOT_EXIST` 오류가 표시될 수 있습니다. 이 오류는 대부분 DB 인스턴스를 생성할 때 지정한 수퍼유저에게 이러한 테이블에 액세스할 권한이 없을 때 발생합니다.

이 오류를 해결하려면 다음 명령을 실행합니다.

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

수퍼유저 권한에 대한 자세한 내용은 마스터 사용자 계정 권한 단원을 참조하십시오.

지정된 시간으로 DB 클러스터 복원

DB 클러스터를 특정 시점으로 복원하여 새 DB 클러스터를 생성할 수 있습니다.

특정 시점으로 DB 클러스터를 복원할 때 기본 Virtual Private Cloud(VPC) 보안 그룹을 선택할 수 있습니다. 또는 DB 클러스터에 사용자 정의 VPC 보안 그룹을 적용할 수 있습니다.

복원된 DB 인스턴스는 기본 DB 클러스터 및 DB 파라미터 그룹과 자동으로 연결됩니다. 하지만 복원 중에 사용자 지정 파라미터 그룹을 지정하여 적용할 수 있습니다.

Amazon Aurora는 DB 클러스터에 대한 로그 레코드를 Amazon S3에 지속적으로 업로드합니다. DB 클러스터의 최근 복원 가능 시간을 확인하려면 AWS CLIdescribe-db-clusters 명령을 사용한 후 DB 클러스터의 `LatestRestorableTime` 필드에 반환되는 값을 살펴봅니다.

백업 보존 기간 중 어느 특정 시점으로든 복원할 수 있습니다. DB 클러스터의 복원 가능한 가장 빠른 시간을 확인하려면 AWS CLIdescribe-db-clusters 명령을 사용한 후 DB 클러스터의 `EarliestRestorableTime` 필드에 반환되는 값을 살펴봅니다.

복원된 DB 클러스터의 백업 보존 기간은 소스 DB 클러스터의 백업 보존 기간과 동일합니다.

> **Note**
>
> 이 주제의 정보는 Amazon Aurora에 적용됩니다. Amazon RDS DB 인스턴스 복원에 대한 자세한 내용은 지정된 시간으로 DB 인스턴스 복원을 참조하세요.
> Aurora DB 클러스터 백업 및 복원에 대한 자세한 내용은 Aurora DB 클러스터 백업 및 복원에 대한 개요 단원을 참조하십시오.
> Aurora MySQL의 경우 프로비저닝된 DB 클러스터를 Aurora Serverless DB 클러스터에 복원할 수 있습니다. 자세한 내용은 Aurora Serverless v1 DB 클러스터 복원 섹션을 참조하세요.
> 또한 AWS Backup을 사용해 Amazon Aurora DB 클러스터의 백업 데이터를 관리할 수도 있습니다. DB 클러스터가 AWS Backup의 백업 계획에 연결되어 있는 경우, 해당 백업 계획이 시점 복구에 사용됩니다. 자세한 내용은 AWS Backup을 사용해 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요.

RDS 추가 지원 버전을 사용하는 Aurora DB 클러스터 또는 글로벌 클러스터를 복원하는 방법에 대한 자세한 내용은 Amazon RDS 추가 지원이 포함된 Aurora DB 클러스터 또는 글로벌 클러스터 복원 섹션을 참조하세요.

자동 백업, 보존된 자동 백업 또는 AWS Backup 사용을 통해 지정된 시간으로 DB 클러스터를 복원합니다.

주제

- [특정 시점으로 DB 클러스터 복원]
- [보존된 자동 백업에서 지정된 시간으로 DB 클러스터 복원]
- [AWS Backup을 사용해 지정된 시간으로 DB 클러스터 복원]

특정 시점으로 DB 클러스터 복원

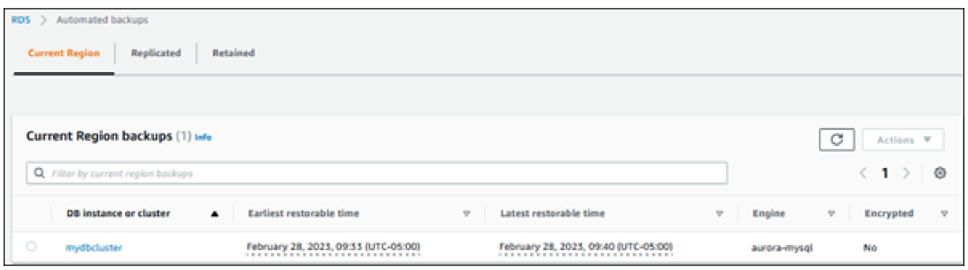
AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 특정 시점으로 복원할 수 있습니다.

콘솔

지정된 시간으로 DB 클러스터 복원

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 자동 백업(Automated backups)을 선택합니다.

   자동 백업은 현재 리전(Current Region) 탭에 표시됩니다.



3. 복원할 DB 클러스터를 선택합니다.
4. 작업에서 특정 시점으로 복구를 선택합니다.

   특정 시점으로 복구 창이 나타납니다.
5. 최근 복원 가능 시간을 선택하여 가능한 최근 시간으로 복원하거나, 사용자 지정을 선택하여 시간을 선택합니다.

   사용자 지정(Custom)을 선택한 경우 클러스터를 복원할 날짜 및 시간을 입력합니다.

> **Note**
>
> 시간은 현지 시간대로 표시됩니다. 즉, 협정 세계시(UTC)에서 오프셋으로 표시됩니다. 예를 들어 UTC-5는 동부 표준시/하절기 중부 표준시입니다.

6. DB 클러스터 식별자에 대상 복원된 DB 클러스터의 이름을 입력합니다. 이름은 고유해야 합니다.
7. 필요에 따라 DB 인스턴스 클래스 및 DB 클러스터 스토리지 구성과 같은 기타 옵션을 선택합니다.

   각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.
8. 특정 시점으로 복구를 선택합니다.

AWS CLI

DB 클러스터를 지정된 시간으로 복원하려면 AWS CLI 명령인 estore-db-cluster-to-point-in-time을 사용하여 DB 클러스터를 새로 생성합니다.

다른 설정을 지정할 수 있습니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

이 작업에는 리소스 태깅이 지원됩니다. `--tags` 옵션을 사용하면 소스 DB 클러스터 태그가 무시되고 제공된 태그가 사용됩니다. 이 옵션을 사용하지 않으면 소스 클러스터의 최신 태그가 사용됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
       --source-db-cluster-identifier mysourcedbcluster \
       --db-cluster-identifier mytargetdbcluster \
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
       --source-db-cluster-identifier mysourcedbcluster ^
       --db-cluster-identifier mytargetdbcluster ^
       --restore-to-time 2017-10-14T23:45:00.000Z
```

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터에 대한 기본 인스턴스(라이터)를 생성합니다. AWS CLI를 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.
> DB 클러스터에 대해 프라이머리 인스턴스를 생성하려면 create-db-instanceAWS CLI 명령을 호출합니다. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하십시오.

RDS API

DB 클러스터를 특정 시간으로 복원하려면, Amazon RDS API `RestoreDBClusterToPointInTime` 작업을 다음 파라미터와 함께 호출합니다.

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터에 대한 기본 인스턴스(라이터)를 생성합니다. RDS API를 사용하여 DB 클러스터를 지정된 시간으로 복원할 경우 반드시 DB 클러스터에 대한 프라이머리 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.
> DB 클러스터에 대한 프라이머리 인스턴스를 생성하려면 RDS API 작업 CreateDBInstance를 호출합니다. DB 클러스터의 이름을 `DBClusterIdentifier` 파라미터 값으로 포함합니다.

보존된 자동 백업에서 지정된 시간으로 DB 클러스터 복원

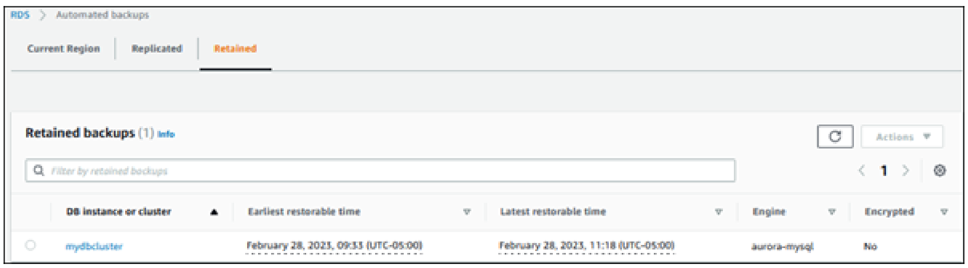
백업이 원본 클러스터의 보존 기간 내에 있는 경우 원본 DB 클러스터를 삭제한 후 보존된 자동 백업에서 DB 클러스터를 복원할 수 있습니다. 프로세스는 자동 백업에서 DB 클러스터를 복원하는 것과 유사합니다.

> **Note**
>
> Aurora Serverless v1 클러스터의 자동 백업은 유지되지 않으므로 이 절차를 사용하여 Aurora Serverless v1 DB 클러스터를 복원할 수 없습니다.

콘솔

지정된 시간으로 DB 클러스터 복원

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 자동 백업(Automated backups)을 선택합니다.
3. 보존됨 탭을 선택합니다.

4. 복원할 DB 클러스터를 선택합니다.
5. 작업에서 특정 시점으로 복구를 선택합니다.

   특정 시점으로 복구 창이 나타납니다.
6. 최근 복원 가능 시간을 선택하여 가능한 최근 시간으로 복원하거나, 사용자 지정을 선택하여 시간을 선택합니다.

   사용자 지정(Custom)을 선택한 경우 클러스터를 복원할 날짜 및 시간을 입력합니다.

   > **Note**
   >
   > 시간은 현지 시간대로 표시됩니다. 즉, 협정 세계시(UTC)에서 오프셋으로 표시됩니다. 예를 들어 UTC-5는 동부 표준시/하절기 중부 표준시입니다.
7. DB 클러스터 식별자에 대상 복원된 DB 클러스터의 이름을 입력합니다. 이름은 고유해야 합니다.
8. 필요에 따라 DB 인스턴스 클래스와 같은 기타 옵션을 선택합니다.

   각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

9. 특정 시점으로 복구를 선택합니다.

AWS CLI

DB 클러스터를 지정된 시간으로 복원하려면 AWS CLI 명령인 estore-db-cluster-to-point-in-time을 사용하여 DB 클러스터를 새로 생성합니다.

다른 설정을 지정할 수 있습니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

이 작업에는 리소스 태깅이 지원됩니다. `--tags` 옵션을 사용하면 소스 DB 클러스터 태그가 무시되고 제공된 태그가 사용됩니다. 이 옵션을 사용하지 않으면 소스 클러스터의 최신 태그가 사용됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Windows의 경우:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터에 대한 기본 인스턴스(라이터)를 생성합니다. AWS CLI를 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 반드시 DB 클러스터를 위한 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.
> DB 클러스터에 대해 프라이머리 인스턴스를 생성하려면 create-db-instanceAWS CLI 명령을 호출합니다. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하십시오.

RDS API

DB 클러스터를 특정 시간으로 복원하려면, Amazon RDS API `RestoreDBClusterToPointInTime` 작업을 다음 파라미터와 함께 호출합니다.

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> 콘솔을 사용하여 DB 클러스터를 특정 시간으로 복원할 경우 Amazon RDS에서 자동으로 DB 클러스터에 대한 기본 인스턴스(라이터)를 생성합니다. RDS API를 사용하여 DB 클러스터를 지정된 시간으로 복원할 경우 반드시 DB 클러스터에 대한 프라이머리 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.
> DB 클러스터에 대한 프라이머리 인스턴스를 생성하려면 RDS API 작업 CreateDBInstance를 호출합니다. DB 클러스터의 이름을 `DBClusterIdentifier` 파라미터 값으로 포함합니다.

AWS Backup을 사용해 지정된 시간으로 DB 클러스터 복원

AWS Backup을 사용하여 자동 백업을 관리한 다음, 지정된 시간으로 이를 복원할 수 있습니다. 이렇게 하려면 AWS Backup에서 백업 계획을 생성하고 DB 클러스터를 리소스로 할당합니다. 그런 다음 백업 규칙에서 PITR에 대해 연속 백업을 활성화합니다. 백업 계획 및 백업 규칙에 대한 자세한 내용은 AWS 백업 개발자 가이드를 참조하세요.

AWS Backup에서 연속 백업 활성화

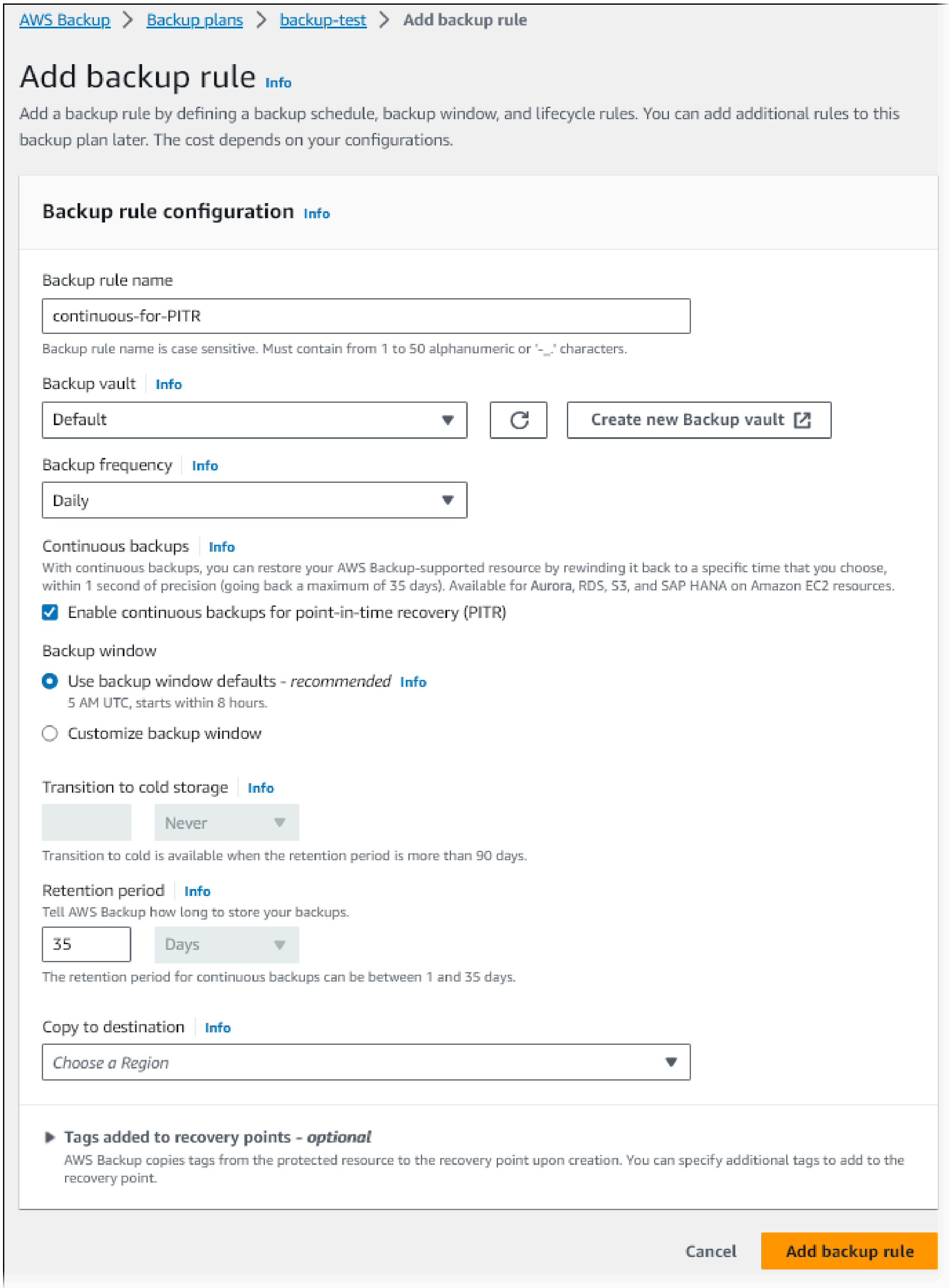
백업 규칙에서 연속 백업을 활성화합니다.

PITR에 대해 연속 백업을 활성화하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/backup에서 AWS Backup 콘솔을 엽니다.
2. 탐색 창에서 백업 계획을 선택합니다.
3. 백업 계획 이름에서 DB 클러스터를 백업하는 데 사용할 백업 계획을 선택합니다.
4. 백업 규칙 섹션에서 백업 규칙 추가를 선택합니다.

그러면 백업 규칙 추가 페이지가 표시됩니다.

5. PITR(특정 시점으로 복구)용 지속적 백업 활성화 확인란을 선택합니다.



6. 필요에 따라 다른 설정을 선택한 다음 백업 규칙 추가를 선택합니다.

AWS Backup의 연속 백업에서 복원

백업 볼트에서 지정된 시간으로 복원합니다.

콘솔

AWS Management Console을 사용해 DB 클러스터를 지정된 시간으로 복원합니다.

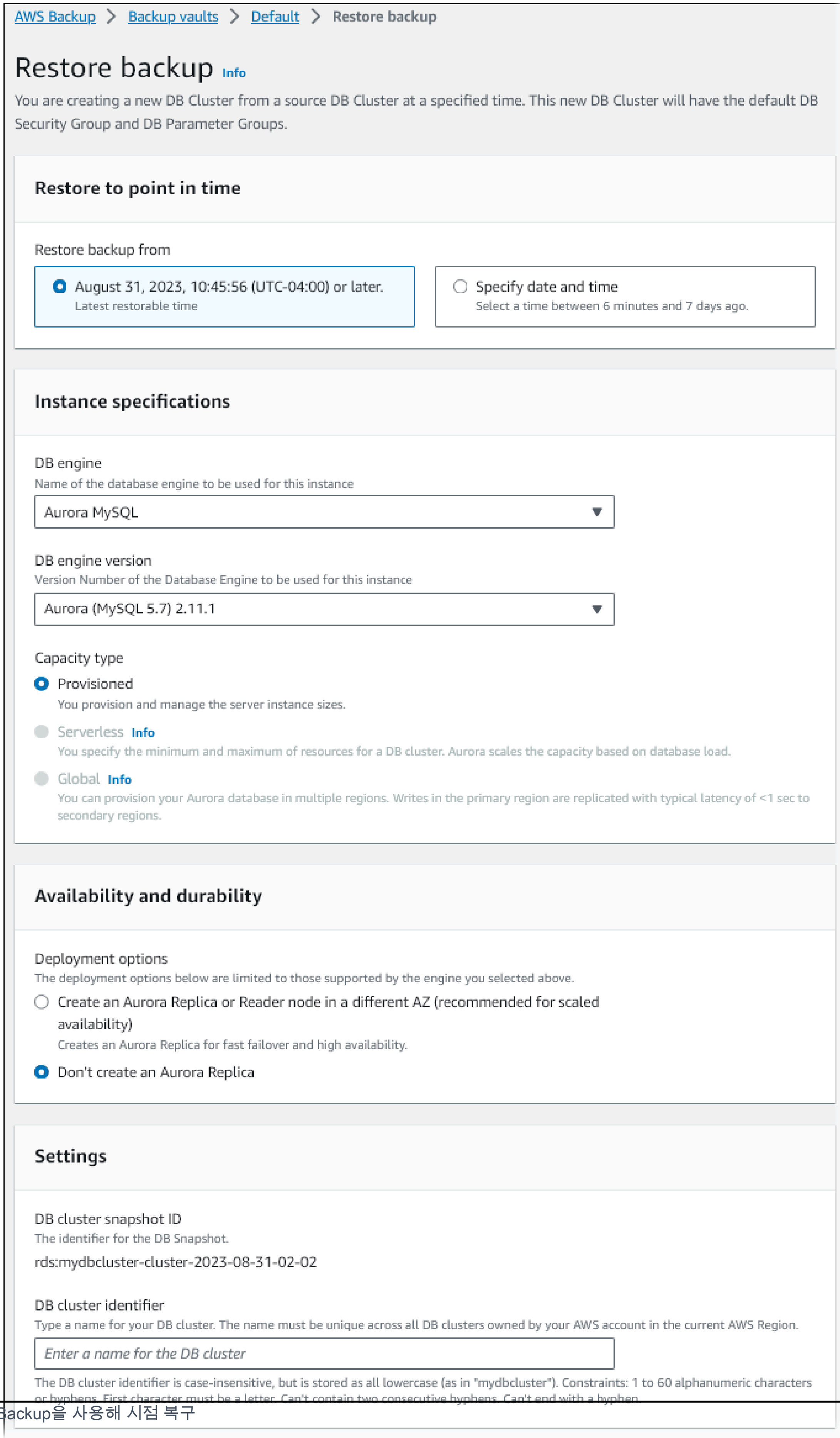
AWS Backup의 연속 백업에서 복원하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/backup에서 AWS Backup 콘솔을 엽니다.
2. 탐색 창에서 백업 저장소를 선택합니다.
3. 연속 백업이 포함된 백업 볼트를 선택합니다(예: 기본값).

   백업 볼트 세부 정보 페이지가 표시됩니다.
4. 복구 시점에서 자동 백업용 복구 시점을 선택합니다.

   백업 유형은 연속이고 이름은 `continuous:cluster-`*`AWS-Backup-job-number`*입니다.
5. 작업에서 복원을 선택합니다.

   백업 복원 페이지가 표시됩니다.


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. 특정 시점으로 복원의 경우, 날짜 및 시간 지정을 선택하여 특정 시점으로 복원할 수 있습니다.
7. DB 클러스터를 복원하는 데 필요한 다른 설정을 선택한 다음, 백업 복원을 선택합니다.

   작업 페이지가 표시되며 복원 작업 창이 나타납니다. 페이지 상단에 복원 작업에 대한 정보를 제공하는 메시지가 나타납니다.

DB 클러스터를 복원한 후에는 기본(라이터) DB 인스턴스를 추가해야 합니다. DB 클러스터에 대해 프라이머리 인스턴스를 생성하려면 create-db-instanceAWS CLI 명령을 호출합니다. DB 클러스터의 이름을 `--db-cluster-identifier` 파라미터 값으로 포함합니다.

CLI

start-restore-job AWS CLI 명령을 사용하여 DB 클러스터를 지정된 시간으로 복원합니다. 다음 파라미터는 필수 파라미터입니다.

- `--recovery-point-arn` – 복원할 복구 시점의 Amazon 리소스 이름(ARN)입니다.
- `--resource-type` – `Aurora`를 사용합니다.
- `--iam-role-arn` – AWS Backup 작업에 사용하는 IAM 역할의 ARN입니다.
- `--metadata` – DB 클러스터를 복원하는 데 사용하는 메타데이터입니다. 다음 파라미터는 필수 파라미터입니다.
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` 또는 `UseLatestRestorableTime`

다음 예제는 DB 클러스터를 지정된 시간으로 복원하는 방법을 보여줍니다.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

다음 예제는 DB 클러스터를 복원 가능한 최신 시간으로 복원하는 방법을 보여줍니다.

```
aws backup start-restore-job \
```

```
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

DB 클러스터를 복원한 후에는 기본(라이터) DB 인스턴스를 추가해야 합니다. DB 클러스터에 대해 프라이머리 인스턴스를 생성하려면 create-db-instanceAWS CLI 명령을 호출합니다. DB 클러스터의 이름을 `--db-cluster-identifier` 파라미터 값으로 포함합니다.

DB 클러스터 스냅샷 삭제

Amazon RDS에서 관리하는 DB 클러스터 스냅샷이 더 이상 필요하지 않으면 삭제할 수 있습니다.

> **Note**
>
> AWS Backup에서 관리하는 백업을 삭제하려면 AWS Backup 콘솔을 사용하십시오. AWS Backup에 대한 자세한 내용은 AWS Backup 개발자 안내서를 참조하세요.

DB 클러스터 스냅샷 삭제

콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

공유 또는 퍼블릭 스냅샷을 삭제하려면 해당 스냅샷을 소유하는 AWS 계정에 로그인해야 합니다.

콘솔

DB 클러스터 스냅샷을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 삭제하고 싶은 DB 클러스터 스냅샷을 선택합니다.
4. 작업(Actions)에서 스냅샷 삭제(Delete snapshot)를 선택합니다.
5. 확인 페이지에서 삭제를 선택합니다.

AWS CLI

AWS CLI 명령 delete-db-cluster-snapshot을 사용하여 DB 크러스터 스냅샷을 삭제할 수 있습니다.

다음 옵션을 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

- `--db-cluster-snapshot-identifier` – DB 클러스터 스냅샷의 식별자입니다.

Example

다음 코드는 `mydbclustersnapshot` DB 클러스터 스냅샷을 삭제합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Windows의 경우:

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

Amazon RDS API 연산 DeleteDBClusterSnapshot을 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

다음 파라미터를 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

- `DBClusterSnapshotIdentifier` – DB 클러스터 스냅샷의 식별자입니다.

자습서: DB 클러스터 스냅샷에서 Amazon Aurora DB 클러스터 복원

Amazon Aurora로 작업할 때 일반적인 시나리오는 가끔 작업하지만 정규직이 필요하지 않은 DB 인스턴스를 갖는 것입니다. 예를 들어 분기별로만 실행하는 보고서의 데이터를 보관하기 위해 DB 클러스터를 사용할 수 있습니다 이러한 시나리오에서 비용을 절약하는 한 가지 방법은 보고서 작성 후 DB 클러스터의 DB 클러스터 DB 클러스터 DB 클러스터 스냅샷을 생성하는 것입니다. 그런 다음 DB 클러스터를 삭제하고 다음 분기 동안 새 데이터를 업로드하고 보고서를 실행해야 할 때 복원합니다.

DB 클러스터를 복원할 때 복원할 DB 클러스터 스냅샷의 이름을 제공합니다. 그런 다음 복원 작업에서 생성되는 새 DB 클러스터의 이름을 입력하면 됩니다. 스냅샷에서 DB 클러스터를 복원하는 방법에 대한 자세한 내용은 [DB 클러스터 스냅샷에서 복원] 섹션을 참조하세요.

이 자습서에서는 복원된 DB 클러스터를 Aurora MySQL 버전 2(MySQL 5.7과 호환)에서 Aurora MySQL 버전 3(MySQL 8.0과 호환)으로 업그레이드합니다.

Amazon RDS 콘솔 또는 AWS CLI를 사용하여 DB 클러스터 스냅샷에서 DB 클러스터를 지정된 시간으로 복원합니다.

Amazon RDS의 AWS KMS 키 관리에 대한 내용은 [AWS KMS key 관리] 섹션을 참조하세요.

주제

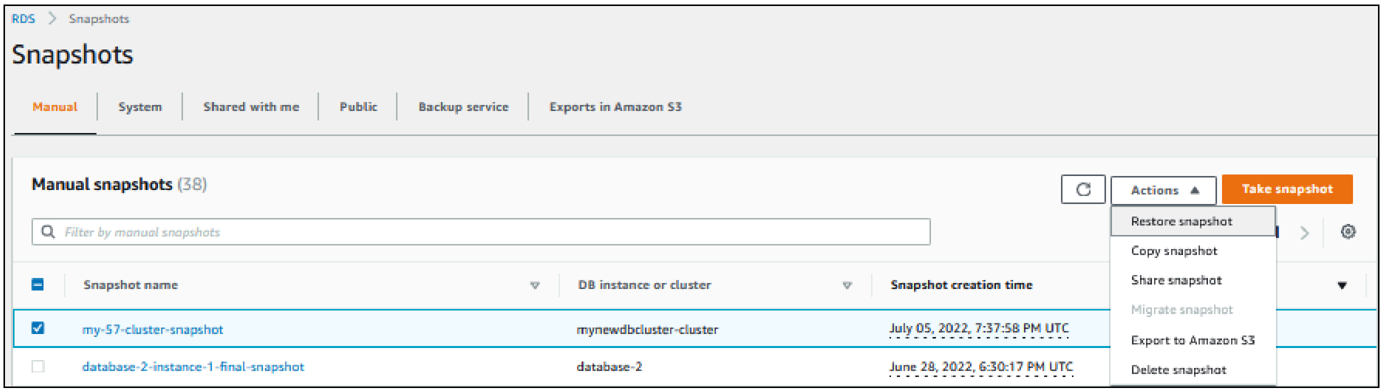
자습서: Amazon RDS 콘솔을 사용하여 DB 클러스터 스냅샷에서 DB 클러스터 복원

이 자습서에서는 Amazon RDS 콘솔을 사용하여 DB 클러스터 스냅샷에서 DB 클러스터를 복원합니다. AWS Management Console을 사용하여 스냅샷에서 DB 클러스터를 복원하는 경우 기본(라이터) DB 인스턴스도 생성됩니다.

> **Note**
>
> 기본 DB 인스턴스가 생성되는 동안에는 리더 인스턴스로 표시되지만 생성 후에는 라이터 인스턴스로 표시됩니다.

DB 클러스터 스냅샷에서 DB 클러스터를 복원하는 방법

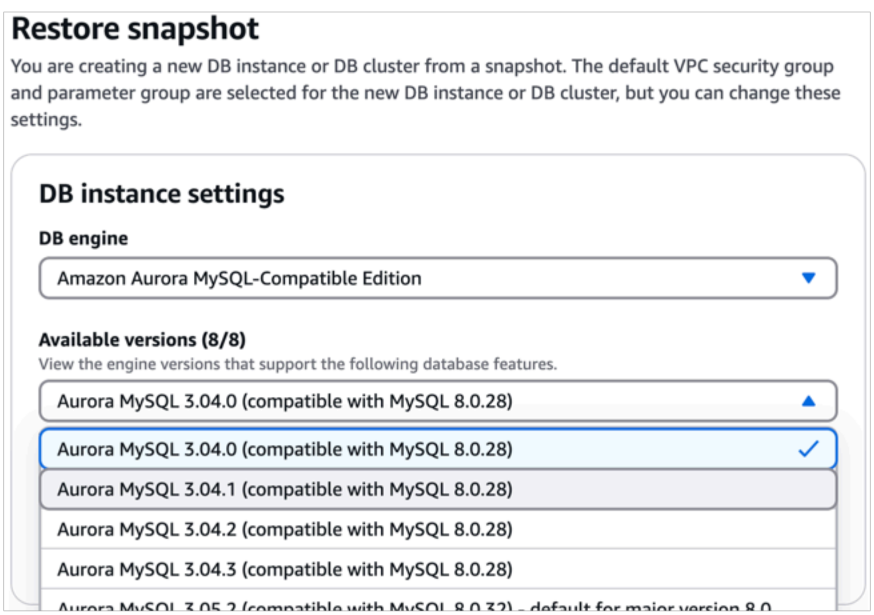
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 복원 원본으로 사용할 DB 클러스터 스냅샷을 선택합니다.
4. 작업에서 스냅샷 복원을 선택합니다.



스냅샷 복원 페이지가 표시됩니다.

5. DB 인스턴스 설정에서 다음을 수행합니다.

   a. DB 엔진의 기본 설정을 사용합니다.
   b. 사용 가능한 버전에서 Aurora MySQL 3.04.0(MySQL 8.0.28과 호환)과 같은 MySQL–8.0 호환 버전을 선택합니다.



6. 설정(Settings) 아래의 DB 인스턴스 식별자(DB instance identifier)에서 복원된 DB 인스턴스에 사용할 고유 이름(예: **`my-80`**)을 입력합니다.

> **Note**
>
> DB 클러스터 식별자를 생성하기 위해 Amazon RDS가 지정한 DB 인스턴스 식별자에 `-cluster`를 추가합니다.

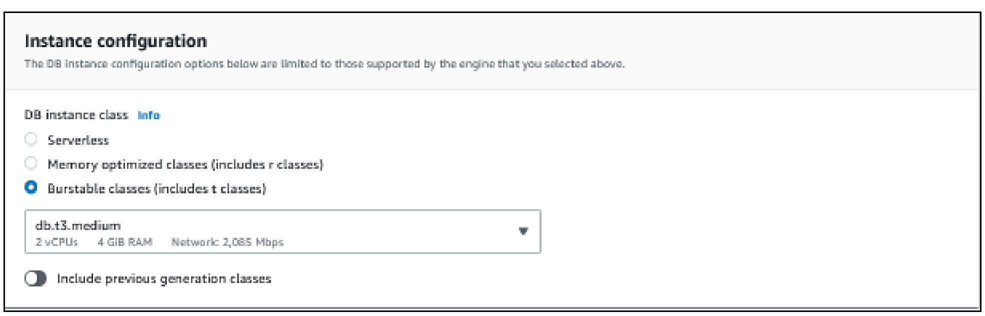
7. 연결 아래에서 다음에 대해 기본 설정을 사용합니다.

   - Virtual private cloud(VPC)
   - DB 서브넷 그룹
   - 공개 액세스(Public access)
   - VPC 보안 그룹(방화벽)

8. DB 인스턴스 클래스(DB instance class)를 선택합니다.

   이 자습서에서는 버스트 가능한 클래스(t 클래스 포함)를 선택한 다음 db.t3.small을 선택합니다.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

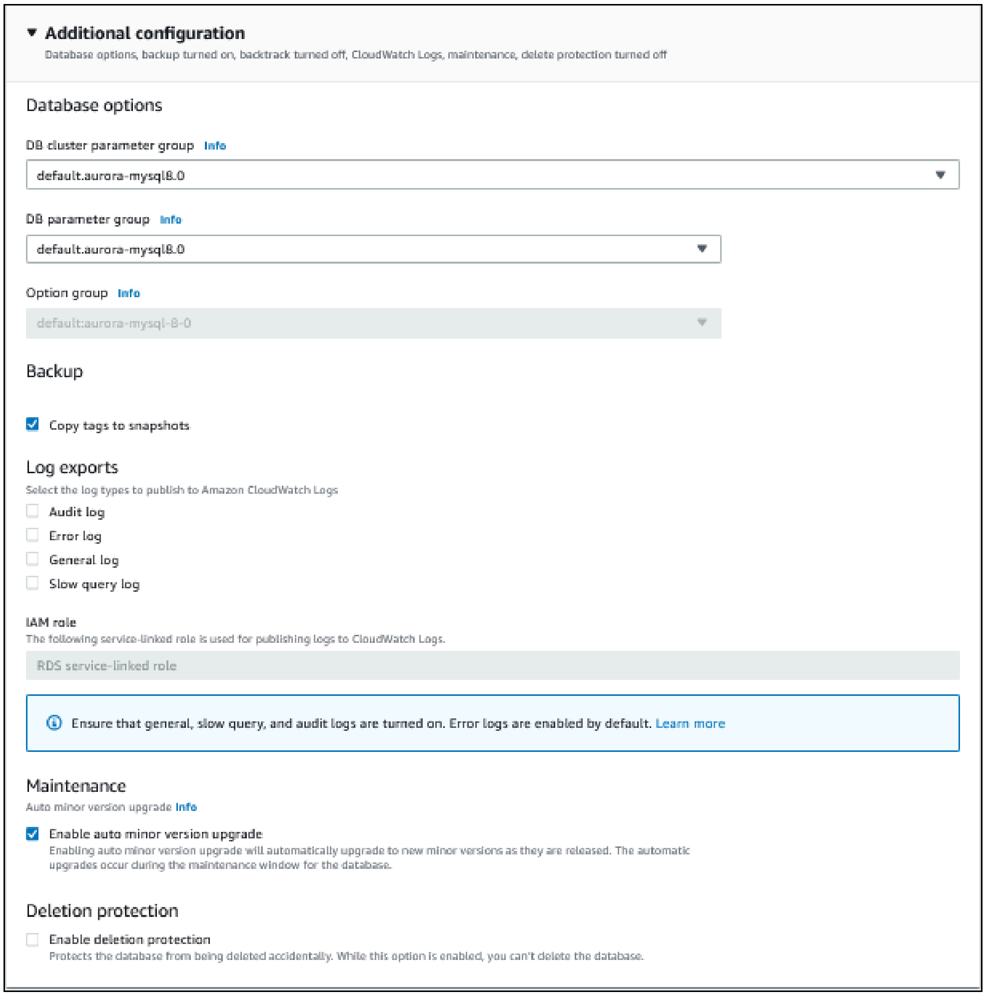


9. 에 대한데이터베이스 인증, 기본 설정을 사용합니다.

10. 암호화에 기본 설정을 사용합니다.

    스냅샷의 원본 DB 클러스터가 암호화된 경우 복원된 DB 클러스터도 암호화됩니다. 암호화되지 않은 상태로 만들 수는 없습니다.

11. 페이지 하단에서 추가 구성(Additional configuration)을 확장합니다.



12. 다음과 같이 변경합니다.

    a. 이 자습서에서는 DB 클러스터 파라미터 그룹을 사용합니다.

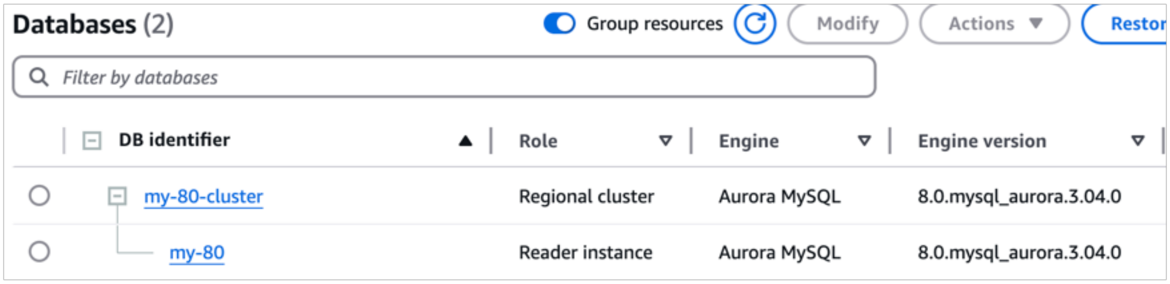
    b. 이 자습서에서는 DB 파라미터 그룹을 사용합니다.

    c. 로그 내보내기의 경우 모든 확인란을 선택합니다.

    d. 삭제 방지에서 삭제 방지 활성화 확인란을 선택합니다.

13. DB 인스턴스 복원을 선택합니다.

데이터베이스 페이지에 복원된 DB 클러스터가 `Creating` 상태로 표시됩니다.



기본 DB 인스턴스가 생성되는 동안에는 리더 인스턴스로 표시되지만 생성 후에는 라이터 인스턴스로 표시됩니다.

자습서: AWS CLI을 사용하여 DB 클러스터 스냅샷에서 DB 클러스터 복원

이 자습서에서는 AWS CLI를 사용하여 DB 클러스터 스냅샷에서 DB 클러스터를 복원합니다. AWS CLI를 사용하여 스냅샷에서 DB 클러스터를 복원하는 데는 두 단계가 있습니다.

1. restore-db-cluster-from-snapshot 명령을 사용한 DB 클러스터 복원
2. create-db-instance 명령을 사용한 기본(라이터) DB 인스턴스 생성

DB 클러스터 복원

`restore-db-cluster-from-snapshot` 명령을 사용합니다. 다음 옵션이 필요합니다.

- `--db-cluster-identifier` - 복원된 DB 클러스터의 이름입니다.
- `--snapshot-identifier` - 복원하는 데 사용되는 스냅샷의 이름입니다.
- `--engine` - 복원된 DB 클러스터의 데이터베이스 엔진입니다. 소스 DB 클러스터의 데이터베이스 엔진과 호환되어야 합니다.

  선택할 수 있는 항목은 다음과 같습니다.

  - `aurora-mysql` – Aurora MySQL 5.7 and 8.0 호환.
  - `aurora-postgresql` – Aurora PostgreSQL과 호환됩니다.

  이 예제에서는 `aurora-mysql`을 사용합니다.
- `--engine-version` - 복원된 DB 클러스터의 이름. 이 예에서는 MySQL-8.0 호환 버전을 사용합니다.

다음 예에서는 이라는 DB 클러스터 스냅샷에서 명명된 `my-new-80-cluster` Aurora MySQL 8.0 호환 DB 클러스터를 복원합니다.

DB 클러스터를 복원하려면

• 다음 명령 중 하나를 사용합니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

다음과 유사하게 출력됩니다.

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
```

```
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```


기본(라이터) DB 인스턴스 생성

기본(라이터) DB 인스턴스를 만들려면 `create-db-instance` 명령을 사용합니다. 다음 옵션이 필요합니다.

- `--db-cluster-identifier` - 복원된 DB 클러스터의 이름.
- `--db-instance-identifier` - 기본 인스턴스의 이름.

- `--db-instance-class` - 기본 인스턴스에 사용할 DB 인스턴스 클래스의 이름입니다. 이 예제에서는 `db.t3.medium`을 사용합니다.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

- `--engine` - 프라이머리 DB 인스턴스의 데이터베이스 엔진입니다. 복원된 DB 클러스터에서 사용하는 것과 동일한 데이터베이스 엔진이어야 합니다.

  선택할 수 있는 항목은 다음과 같습니다.

  - `aurora-mysql` – Aurora MySQL 5.7 and 8.0 호환.
  - `aurora-postgresql` – Aurora PostgreSQL과 호환됩니다.

  이 예제에서는 `aurora-mysql`을 사용합니다.

다음 예제에서는 `my-new-80-cluster-instance`라는 이름의 복원된 Aurora MySQL 8.0 호환 DB 클러스터에 이름이 `my-new-80-cluster`라고 지정된 기본(라이터) DB 인스턴스를 생성합니다

기본(라이터) DB 인스턴스를 생성하려면

- 다음 명령 중 하나를 사용합니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  Windows의 경우:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
```

```
    --engine aurora-mysql
```

다음과 유사하게 출력됩니다.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
```

```
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
```

```
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

Amazon Aurora 클러스터에서 지표 모니터링

Amazon Aurora은 복제된 데이터베이스 서버의 클러스터를 사용합니다. 일반적으로 Aurora 클러스터를 모니터링하려면 여러 DB 인스턴스의 상태를 확인해야 합니다. 인스턴스는 대부분의 쓰기 작업을 처리하거나, 읽기 작업만 처리하거나, 두 작업을 조합하여 수행하는 등 전문적인 역할을 맡을 수 있습니다. 또한 복제 지연을 측정하여 클러스터의 전반적인 상태를 모니터링합니다. 한 DB 인스턴스에서 변경한 내용을 다른 인스턴스에서 사용할 수 있는 시간의 양입니다.

주제

모니터링 계획

Amazon Aurora 모니터링을 시작하기 전에 모니터링 계획을 생성합니다. 이 계획에서는 다음과 같은 의문 사항을 해결합니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구

- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

성능 기준

모니터링 목표를 달성하려면 기준을 설정해야 합니다. 이렇게 하려면 Amazon Aurora 환경에서 다양한 시간과 다양한 부하 조건으로 성능을 측정해야 합니다. 다음과 같은 지표를 모니터링할 수 있습니다.

- 네트워크 처리량
- 클라이언트 연결
- 읽기, 쓰기 또는 메타데이터 작업의 I/O
- DB 인스턴스의 버스트 크레딧 밸런스

Amazon Aurora에 대한 성능 이력 데이터를 저장하는 것이 좋습니다. 저장된 데이터를 사용하여 현재 성능을 과거 추세와 비교할 수 있습니다. 또한 정상적인 성능 패턴과 이상 현상을 구별하고 문제를 해결하는 기술을 고안할 수 있습니다.

성능 지침

일반적으로 성능 지표에 허용되는 값은 기준이 무엇인지 그리고 애플리케이션 무엇을 수행하는지에 따라 다릅니다. 기준과의 일관된 차이 또는 추세를 조사하십시오. 다음과 같은 지표가 성능 문제의 원인인 경우가 많습니다.

- CPU 또는 RAM 사용량이 많음 – CPU 또는 RAM 사용량이 많을 경우 해당 애플리케이션의 목표(처리량 또는 동시성)와 일치하고 예상되는 결과라면 문제가 되지 않을 수 있습니다.
- 디스크 공간 사용량 – 총 디스크 용량의 85퍼센트 이상이 계속 사용될 경우 디스크 공간 사용량을 검사합니다. 인스턴스에서 데이터를 삭제할 수 있는지 또는 다른 시스템에 데이터를 아카이브하여 공간을 확보할 수 있는지 확인합니다.
- 네트워크 트래픽 – 네트워크 트래픽의 경우 시스템 관리자에게 문의하여 해당 도메인 네트워크 및 인터넷 연결의 기대 처리량을 확인합니다. 처리량이 기대값보다 항상 낮으면 네트워크 트래픽을 검사합니다.
- 데이터베이스 연결 – 사용자 연결 수가 많고 인스턴스 성능 및 응답 시간이 저하되는 경우 데이터베이스 연결 제한을 고려합니다. DB 인스턴스에 대한 최적의 사용자 연결 수는 해당 인스턴스 클래스와, 수행하는 작업의 복잡성에 따라 다릅니다. 데이터베이스 연결 수를 지정하려면 DB 인스턴스를 `User Connections` 파라미터가 0(무제한)이 아닌 다른 값으로 설정된 파라미터 그룹과 연결합니

다. 기존 파라미터 그룹을 사용하거나 새로 하나 만들 수 있습니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

- IOPS 지표 – IOPS 지표의 기대값은 디스크 사양 및 서버 구성에 따라 다르므로 해당 기준에 일반적인 값을 파악합니다. 값이 기준과 계속 차이가 나는지 검사합니다. 최적의 IOPS 성능을 위해, 일반적인 작업 세트가 메모리에 적합하고 읽기 및 쓰기 작업을 최소화하는지 확인합니다.

성능이 설정된 기준을 벗어나면 워크로드에 맞게 데이터베이스 가용성을 최적화하기 위해 변경해야 할 수 있습니다. 예를 들어 DB 인스턴스의 인스턴스 클래스를 변경해야 할 수 있습니다. 또는 클라이언트에 사용할 수 있는 DB 인스턴스 및 읽기 전용 복제본의 수를 변경해야 할 수도 있습니다.

Amazon Aurora의 모니터링 도구

모니터링은 Amazon Aurora 및 사용자의 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 있어서 중요한 부분입니다. AWS는 Amazon Aurora을 모니터링하고, 이상이 있을 때 이를 보고하고, 적절할 경우 자동 조치를 취할 수 있도록 모니터링 도구를 제공합니다.

주제

- 자동 모니터링 도구
- 수동 모니터링 도구

자동 모니터링 도구

모니터링 작업은 최대한 자동화하는 것이 좋습니다.

주제

- Amazon Aurora 클러스터상태 및 권장 사항
- Amazon Aurora에 대한 Amazon CloudWatch 지표
- Amazon RDS 성능 개선 도우미 및 운영 체제 모니터링
- 통합 서비스

Amazon Aurora 클러스터상태 및 권장 사항

다음과 같은 자동화된 도구를 사용하여 Amazon Aurora을 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon Aurora 클러스터 상태 - Amazon RDS 콘솔, AWS CLI 또는 RDS API를 사용하여 클러스터의 현재 상태에 대한 세부 정보를 봅니다.
- Amazon Aurora 권장 사항 - DB 인스턴스, DB 클러스터, DB 클러스터 파라미터 그룹 등의 데이터베이스 리소스에 대한 자동화된 권장 사항에 응답합니다. 자세한 내용은 Amazon Aurora의 권장 사항 단원을 참조하십시오.

Amazon Aurora에 대한 Amazon CloudWatch 지표

Amazon Aurora는 추가 모니터링 기능을 위해 Amazon CloudWatch, Amazon EventBridge 및 과 통합됩니다.

- Amazon CloudWatch – 이 서비스는 AWS 리소스와 AWS에서 실시간으로 실행 중인 애플리케이션을 모니터링합니다. 다음 Amazon CloudWatch 기능을 Amazon Aurora에 사용할 수 있습니다.
  - Amazon CloudWatch 지표 – Amazon Aurora는 각각의 활성 데이터베이스에 대해 자동으로 1분마다 CloudWatch로 지표를 전송합니다. CloudWatch에서 Amazon RDS 지표에 대한 추가 요금은 표시되지 않습니다. 자세한 내용은 Amazon Aurora에 대한 Amazon CloudWatch 지표 섹션을 참조하세요.
  - Amazon CloudWatch 경보 – 특정 기간 동안 단일 Amazon Aurora 지표를 볼 수 있습니다. 그런 다음 설정한 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행할 수 있습니다.

Amazon RDS 성능 개선 도우미 및 운영 체제 모니터링

다음과 같은 자동화된 도구를 사용하여 Amazon Aurora 성능을 모니터링할 수 있습니다.

- Amazon RDS 성능 개선 도우미 - 데이터베이스의 로드를 평가하고 조치를 취할 시점과 위치를 결정합니다. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하세요.
- Amazon RDS 확장된 모니터링 - 운영 체제에 대한 지표를 실시간으로 확인합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 단원을 참조하세요.

통합 서비스

다음의 AWS 서비스는 Amazon Aurora와 통합됩니다.

- Amazon EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. 자세한 내용은 Amazon Aurora 이벤트 모니터링 단원을 참조하세요.
- Amazon CloudWatch Logs로 Amazon Aurora 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. 자세한 내용은 Amazon Aurora 로그 파일 모니터링 단원을 참조하세요.
- AWS CloudTrail은 직접 수행하거나 AWS 계정을 대신하여 수행한 API 직접 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 자세한 내용은 AWS CloudTrail에서 Amazon Aurora API 호출 모니터링 단원을 참조하세요.
- 데이터베이스 활동 스트림은 DB 클러스터에서 거의 실시간에 가까운 활동 스트림을 제공하는 Amazon Aurora 기능입니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 단원을 참조하세요.
- DevOps Guru for RDS는 Amazon Aurora 데이터베이스의 성능 개선 도우미 지표에 기계 학습을 적용하는 Amazon DevOps Guru 기능입니다. 자세한 내용은 Amazon DevOps Guru for Amazon RDS로 Aurora 성능 이상 분석 단원을 참조하세요.

수동 모니터링 도구

CloudWatch 경보가 다루지 않는 항목은 수동으로 모니터링해야 합니다. Amazon RDS, CloudWatch, AWS Trusted Advisor 및 기타 AWS 콘솔 대시보드에서는 AWS 환경의 상태를 한 눈에 볼 수 있습니다. 또한 DB 인스턴스에서 로그 파일을 확인하는 것이 좋습니다.

- Amazon RDS 콘솔에서 리소스에 대해 다음과 같은 항목을 모니터링할 수 있습니다.
  - DB 인스턴스에 대한 연결 수
  - DB 인스턴스에 대한 읽기 및 쓰기 작업량
  - DB 인스턴스에서 현재 사용 중인 스토리지 양
  - DB 인스턴스에 대해 사용 중인 메모리 및 CPU 양
  - DB 인스턴스에서 주고 받는 네트워크 트래픽 양
- Trusted Advisor 대시보드에서는 다음과 같은 비용 최적화, 보안, 내결함성과 성능 개선 확인을 살펴볼 수 있습니다.
  - Amazon RDS 유휴 DB 인스턴스
  - Amazon RDS 보안 그룹 액세스 위험
  - Amazon RDS 백업
  - Amazon RDS 다중 AZ
  - Aurora DB 인스턴스 액세스

  이러한 사항에 대한 자세한 정보를 알고 싶다면 Trusted Advisor Best Practices (Checks) 단원을 참조하십시오.
- CloudWatch 홈 페이지에 표시되는 항목은 다음과 같습니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch를 사용하여 다음을 수행할 수 있습니다.
  - 사용자 정의 대시보드를 생성하여 관심 있는 서비스를 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악.
  - 모든 AWS 리소스 지표 검색 및 찾아보기
  - 문제에 대해 알려주는 경보 생성 및 편집.

클러스터 상태 조회

Amazon RDS 콘솔을 사용하여 빠르게 DB 클러스터 상태에 액세스할 수 있습니다.

주제

Amazon Aurora DB 클러스터 보기

Amazon Aurora DB 클러스터 및 DB 클러스터에 있는 DB 인스턴스에 대한 정보를 보는 몇 가지 옵션은 다음과 같습니다.

- Amazon RDS 콘솔의 탐색 창에서 Databases(데이터베이스)를 선택하면 DB 클러스터와 DB 인스턴스를 볼 수 있습니다.
- AWS Command Line Interface(AWS CLI)를 사용하여 DB 클러스터와 DB 인스턴스에 대한 정보를 얻을 수 있습니다.
- Amazon RDS API를 사용하여 DB 클러스터와 DB 인스턴스에 대한 정보를 얻을 수 있습니다.

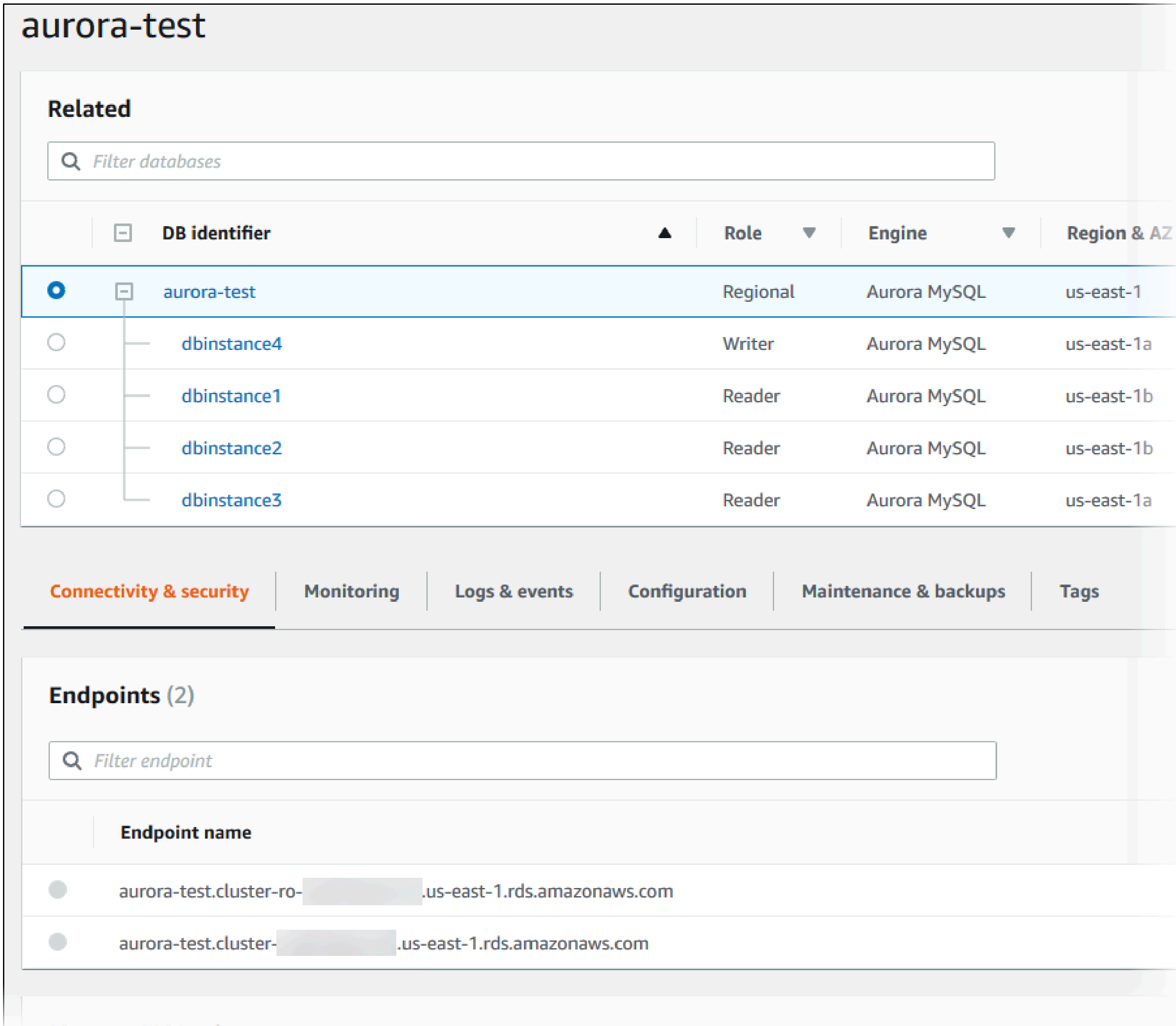
콘솔

Amazon RDS 콘솔에서 콘솔의 탐색 창에 있는 Databases(데이터베이스)를 선택하면 DB 클러스터에 관한 세부 정보를 볼 수 있습니다. Amazon Aurora DB 클러스터의 구성원인 DB 인스턴스에 대한 세부 정보도 볼 수 있습니다.

Amazon RDS 콘솔에서 DB 클러스터를 보거나 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 목록에서 확인하려는 Aurora DB 클러스터의 이름을 선택합니다.

   예를 들어 다음 이미지는 `aurora-test`라는 DB 클러스터의 세부 정보 페이지를 보여줍니다. DB 클러스터에는 DB 식별자 목록에 표시된 4개의 DB 인스턴스가 있습니다. 라이터 DB 인스턴스인 `dbinstance4`는 DB 클러스터의 기본 인스턴스입니다.



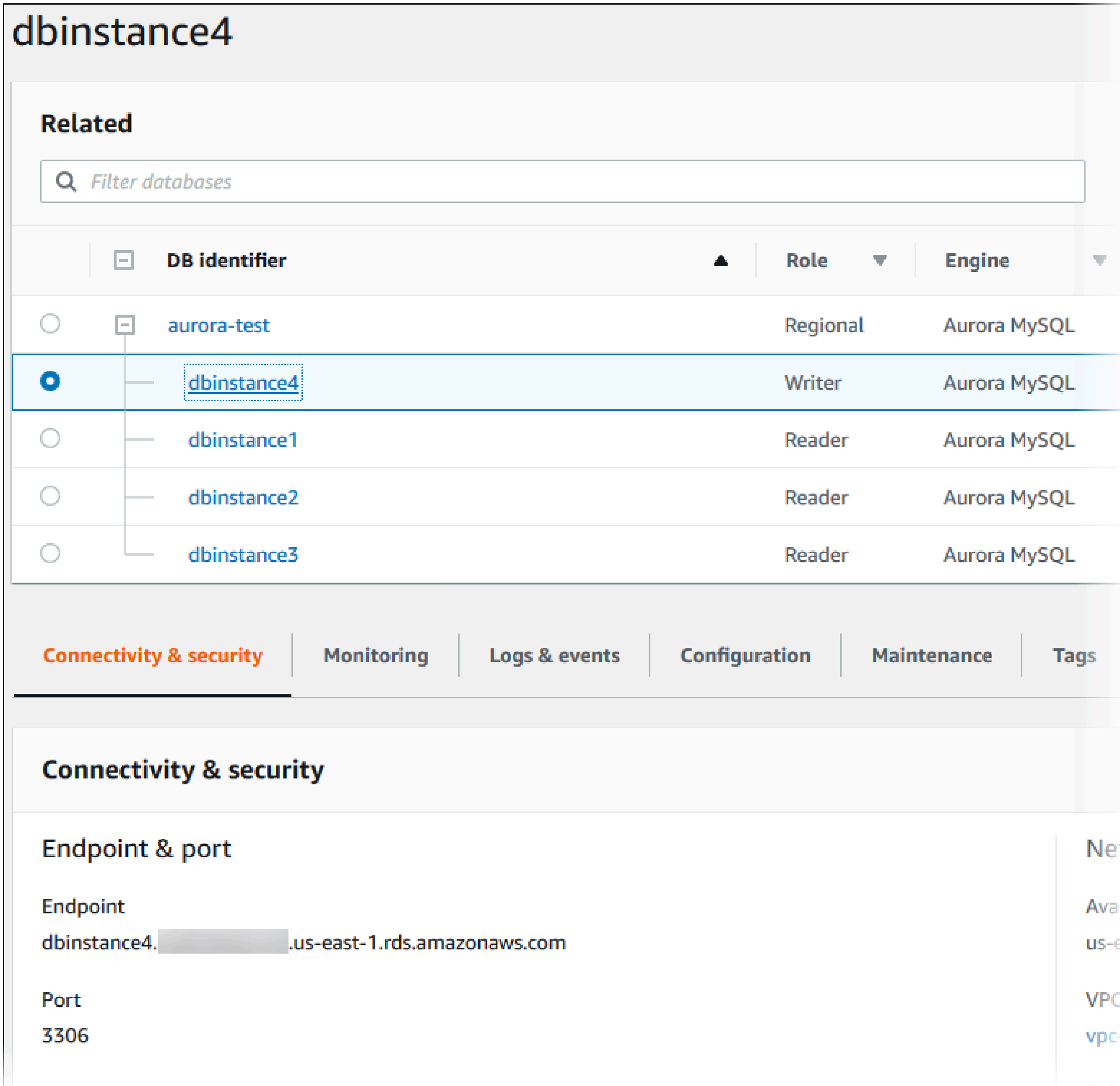
4. DB 클러스터를 수정하려면 목록에서 DB 클러스터를 선택하고 Modify(수정)를 선택합니다.

Amazon RDS 콘솔에서 DB 클러스터의 DB 인스턴스를 보거나 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 다음 중 하나를 수행합니다.

   - DB 인스턴스를 보려면 목록에서 Aurora DB 클러스터의 구성원인 인스턴스를 하나 선택합니다.

예를 들어 `dbinstance4` DB 인스턴스 식별자를 선택하면 콘솔에 다음 이미지와 같이 `dbinstance4` DB 인스턴스에 대한 세부 정보 페이지가 표시됩니다.



- DB 인스턴스를 수정하려면 목록에서 DB 인스턴스를 선택하고 Modify(수정)를 선택합니다. DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정](#) 단원을 참조하세요.

AWS CLI

AWS CLI를 사용하여 DB 클러스터 정보를 보려면 [describe-db-clusters](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-db-clusters.html) 명령을 사용하세요. 예를 들어 다음 AWS CLI 명령은 구성된 `us-east-1` 계정에 대해 수정 AWS 리전에 있는 모든 DB 클러스터의 정보를 표시합니다.

```
aws rds describe-db-clusters --region us-east-1
```

AWS CLI가 JSON 출력으로 구성되어 있을 경우 이 명령은 다음과 같은 출력을 반환합니다.

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
```

```
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터 정보를 보려면 DescribeDBClusters 작업을 사용하십시오.

DB 클러스터 상태 보기

DB 클러스터의 상태는 해당 상태를 나타냅니다. Amazon RDS 콘솔, AWS CLI 또는 API를 사용하여 DB 클러스터와 클러스터 인스턴스 상태를 확인할 수 있습니다.

> **Note**
>
> 또한 Aurora는 유지 관리 상태라는 상태를 한 가지 더 사용하며, 이는 Amazon RDS 콘솔의 유지 관리 열에 표시됩니다. 이 값은 DB 클러스터에 적용해야 하는 모든 유지 관리 패치 상태를 나타냅니다. 유지 관리 상태는 DB 클러스터 상태와 무관합니다. 유지 관리 상태에 대한 자세한 내용은 DB 클러스터에 업데이트 적용 단원을 참조하세요.

다음 표에서 DB 클러스터에 대한 가능한 상태 값을 확인하세요.

DB 클러스터 상태	청구	설명
사용 가능	청구	DB 클러스터에 문제가 없으며 사용할 수 있습니다. Aurora Serverless 클러스터를 사용할 수 있고 일시 중지한 경우에는 스토리지에 대한 요금만 청구됩니다.
Backing-up	청구	현재 DB 클러스터를 백업 중입니다.
Backtracking	청구	현재 DB 클러스터를 백트랙 중입니다. 이 상태는 Aurora MySQL에만 적용됩니다.
Cloning-failed	미청구	DB 클러스터 복제에 실패했습니다.
[생성 중]	미청구	DB 클러스터를 생성 중입니다. 생성 중인 DB 클러스터에는 액세스할 수 없습니다.
[삭제 중]	미청구	DB 클러스터를 삭제 중입니다.
Failing-over	청구	기본 인스턴스에서 Aurora Replica에 대한 장애 조치가 수행 중입니다.

DB 클러스터 상태	청구	설명
Inaccessible-encry ption-credentials	미청구	DB 클러스터를 암호화 또는 복호화하는 데 사용되는 AWS KMS key에 액세스하거나 복구할 수 없습니다.
Inaccessible-encry ption-credentials-recoverable	요금 부과 스토리지	DB 클러스터를 암호화 또는 해독하는 데 사용되는 KMS 키는 액세스할 수 없습니다. 그러나 KMS 키가 활성 상태이면 DB 클러스터를 다시 시작하면 복구할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 암호화을 참조하세요.
Maintenance	청구	Amazon RDS는 DB 클러스터에 유지 관리 업데이트를 적용합니다. 이 상태는 RDS가 한참 전에 미리 예약하는 DB 클러스터 수준의 유지 관리에 사용됩니다.
Migrating	청구	DB 클러스터 스냅샷이 DB 클러스터로 복원 중입니다.
Migration-failed	미청구	마이그레이션이 실패했습니다.
Modifying	청구	고객의 DB 클러스터 수정 요청으로 인해 DB 클러스터를 수정 중입니다.
Promoting	청구	읽기 전용 복제본이 독립형 DB 클러스터로 승격 중입니다.
Preparing-data-mig ration	청구	Amazon RDS가 데이터를 Aurora로 마이그레이션할 준비를 하고 있습니다.
이름 바꾸기	청구	고객의 이름 바꾸기 요청으로 인해 DB 클러스터 이름을 바꾸는 중입니다.
Resetting-master-c redentials	청구	고객의 재설정 요청으로 인해 DB 클러스터 사용자 자격 증명을 재설정하는 중입니다.

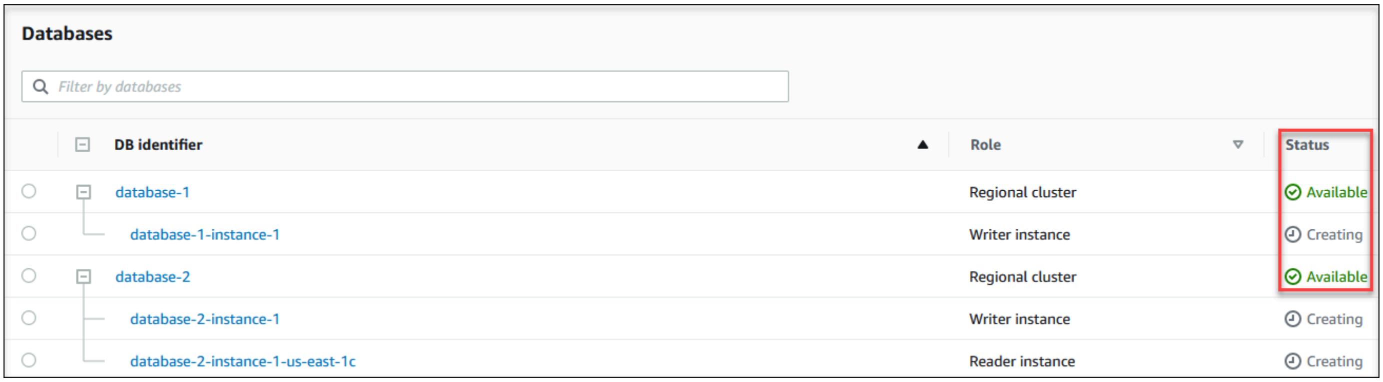
DB 클러스터 상태	청구	설명
[시작됨]	요금 부과 스토리지	DB 클러스터를 시작 중입니다.
중지됨	요금 부과 스토리지	DB 클러스터가 중단됩니다.
[중지 중]	요금 부과 스토리지	DB 클러스터를 중지 중입니다.
스토리지 최적화	청구	DB 인스턴스가 스토리지 크기 또는 유형을 변경하도록 수정 중입니다. DB 인스턴스가 완전히 작동 중입니다. 그러나 DB 인스턴스가 스토리지 최적화 상태에 있는 동안 DB 인스턴스의 스토리지에 대한 변경을 요청할 수 없습니다. 스토리지 최적화 프로세스는 보통 짧지만, 가끔 최대 24시간까지 걸릴 수도 있습니다.
Update-iam-db-auth	청구	DB 클러스터에 대한 IAM 권한 부여가 업데이트 중입니다.
Upgrading	청구	DB 클러스터 엔진 또는 운영 체제 버전은 현재 업그레이드 중입니다.

콘솔

DB 클러스터의 상태를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

   DB 클러스터 목록과 함께 데이터베이스 페이지가 표시됩니다. 각 DB 클러스터에 대해 상태 값이 표시됩니다.



CLI

DB 클러스터의 상태만 보려면 AWS CLI에서 다음 쿼리를 사용하십시오.

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

Aurora 클러스터에서 DB 인스턴스 상태 보기

Aurora 클러스터의 DB 인스턴스 상태는 DB 인스턴스의 상태를 나타냅니다. 다음 절차를 사용하여 Amazon RDS 콘솔, AWS CLI 명령 또는 API 작업에서 클러스터의 DB 인스턴스 상태를 볼 수 있습니다.

> **Note**
>
> 또한 Amazon RDS는 유지 관리 상태라는 상태를 한 가지 더 사용하며, 이는 Amazon RDS 콘솔의 유지 관리 열에 표시됩니다. 이 값은 DB 인스턴스에 적용해야 하는 모든 유지 관리 패치 상태를 나타냅니다. 유지 관리 상태는 DB 인스턴스 상태와 무관합니다. 유지 관리 상태에 대한 자세한 내용은 DB 클러스터에 업데이트 적용 단원을 참조하세요.

다음 표에서 DB 인스턴스에 사용할 수 있는 상태 값을 찾습니다. 이 표에는 DB 인스턴스와 스토리지에만 요금이 청구되는지, 스토리지에만 청구되는지, 혹은 청구되지 않는지 표시됩니다. 모든 DB 인스턴스 상태의 백업 사용에는 항상 청구됩니다.

DB 인스턴스 상태	청구	설명
available	청구	DB 인스턴스에 문제가 없으며 사용할 수 있습니다.
backing-up	청구	현재 DB 인스턴스를 백업 중입니다.
backtracking	청구	현재 DB 인스턴스를 역추적 중입니다. 이 상태는 Aurora MySQL에만 적용됩니다.
configuring-enhanced-monitoring	청구	이 DB 인스턴스에 대해 확장 모니터링이 활성화 또는 비활성화 상태입니다.
configuring-iam-database-auth	청구	이 DB 인스턴스에 대해 AWS Identity and Access Management(IAM) 데이터베이스 인증이 활성화 또는 비활성화 상태입니다.
configuring-log-exports	청구	Amazon CloudWatch Logs에 로그 파일을 게시하는 기능이 이 DB 인스턴스에 대해 활성화 또는 비활성화 상태입니다.

DB 인스턴스 상태	청구	설명
converting-to-vpc	청구	DB 인스턴스를 Amazon Virtual Private Cloud(Amazon VPC)에 없는 DB 인스턴스에서 Amazon VPC에 있는 DB 인스턴스로 변환 중입니다.
creating	미청구(비 PITR) 청구(PITR만 해당)	DB 인스턴스를 생성 중입니다. 생성 중인 DB 인스턴스에는 액세스할 수 없습니다. 시점 복구(PITR) 시 데이터베이스를 복원하는 경우 데이터베이스가 생성 중 상태인 동안 요금이 청구됩니다. 이는 생성 중 상태에서 요금이 발생하는 유일한 시나리오입니다.
delete-precheck	미청구	Amazon RDS가 읽기 전용 복제본이 정상이며 삭제해도 안전한지 검증하고 있습니다.
deleting	미청구	DB 인스턴스를 삭제 중입니다.
failed	미청구	DB 인스턴스 오류가 발생하여 Amazon RDS로 복구할 수 없습니다. DB 인스턴스의 복원 가능한 가장 최근 시간을 기준으로 특정 시점으로 복원을 수행하여 데이터를 복구합니다.
inaccessible-encryption-credentials	미청구	DB 인스턴스를 암호화 또는 복호화하는 데 사용되는 AWS KMS key에 액세스하거나 복구할 수 없습니다.
inaccessible-encryption-credentials-recoverable	요금 부과 스토리지	DB 인스턴스를 암호화 또는 해독하는 데 사용되는 KMS 키는 액세스할 수 없습니다. 그러나 KMS 키가 활성 상태이면 DB 인스턴스를 다시 시작하면 복구할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 암호화 섹션을 참조하세요.

DB 인스턴스 상태	청구	설명
incompatible-create	미청구	Amazon RDS가 DB 인스턴스를 만들려고 하지만 리소스가 DB 인스턴스와 호환되지 않기 때문에 만들 수 없습니다. 예를 들어 DB 인스턴스의 인스턴스 프로파일에 올바른 권한이 없는 경우 이 상태가 발생할 수 있습니다.
incompatible-network	미청구	Amazon RDS가 DB 인스턴스에 대한 복구 작업을 시도했으나 VPC의 상태로 인하여 작업을 완료할 수 없어 복구하지 못했습니다. 예를 들어 서브넷에 사용 가능한 IP 주소가 모두 사용 중이어서 Amazon RDS가 DB 인스턴스에 대한 IP 주소를 받을 수 없는 경우 이러한 상태가 발생할 수 있습니다.
incompatible-option-group	청구	Amazon RDS 옵션 그룹 변경을 적용하려고 시도했으나 적용하지 못했습니다. 또한 Amazon RDS는 이전 옵션 그룹 상태를 롤백하지 못했습니다. 자세한 내용은 DB 인스턴스에 대한 최근 이벤트 목록을 참조하십시오. 예를 들어, 옵션 그룹은 TDE 등의 옵션을 포함하는데 DB 인스턴스는 암호화된 정보를 포함하지 않는 경우 이러한 상태가 발생할 수 있습니다.
incompatible-parameters	청구	Amazon RDS가 DB 인스턴스의 DB 파라미터 그룹에서 지정된 파라미터가 DB 인스턴스와 호환되지 않아 DB 인스턴스를 시작할 수 없습니다. DB 인스턴스에 대한 액세스 권한을 다시 얻으려면 변경된 파라미터를 되돌리거나 DB 인스턴스와 호환되는 파라미터를 정의해야 합니다. 호환되지 않는 파라미터에 대한 자세한 내용은 DB 인스턴스에 대한 최근 이벤트 목록을 참조하십시오.
incompatible-restore	미청구	Amazon RDS가 특정 시점으로 복원을 수행할 수 없습니다. 임시 테이블 사용 또는 MySQL에서 MyISAM 테이블 사용이 일반적인 원인입니다.

DB 인스턴스 상태	청구	설명
insufficient-capacity	미청구	Amazon RDS가 현재 충분한 용량을 사용할 수 없기 때문에 인스턴스를 생성할 수 없습니다. 동일한 인스턴스 유형으로 동일한 AZ에 DB 인스턴스를 생성하려면 DB 인스턴스를 삭제하고 몇 시간 기다린 후 다시 생성해 봅니다. 또는 다른 인스턴스 클래스 또는 AZ를 사용하여 새 인스턴스를 생성합니다.
유지 관리	청구	Amazon RDS는 DB 인스턴스에 유지 관리 업데이트를 적용합니다. 이 상태는 RDS가 한참 전에 미리 예약하는 인스턴스 수준의 유지 관리에 사용됩니다.
modifying	청구	고객의 DB 인스턴스 수정 요청으로 인해 DB 인스턴스를 수정 중입니다.
moving-to-vpc	청구	DB 인스턴스가 새 Amazon Virtual Private Cloud(Amazon VPC)로 이동하는 중입니다.
rebooting	청구	고객의 요청 또는 DB 인스턴스 재부팅이 필요한 Amazon RDS 프로세스로 인해 DB 인스턴스를 재부팅 중입니다.
resetting-master-credentials	청구	고객의 재설정 요청으로 인해 DB 인스턴스 사용자 자격 증명을 재설정하는 중입니다.
renaming	청구	고객의 이름 바꾸기 요청으로 인해 DB 인스턴스 이름을 바꾸는 중입니다.
restore-error	청구	특정 시점으로 복원을 시도하거나 스냅샷에서 복원을 시도할 때 DB 인스턴스에서 오류가 발생했습니다.
starting	요금 부과 스토리지	DB 인스턴스를 시작하는 중입니다.
stopped	요금 부과 스토리지	DB 인스턴스가 중지되었습니다.

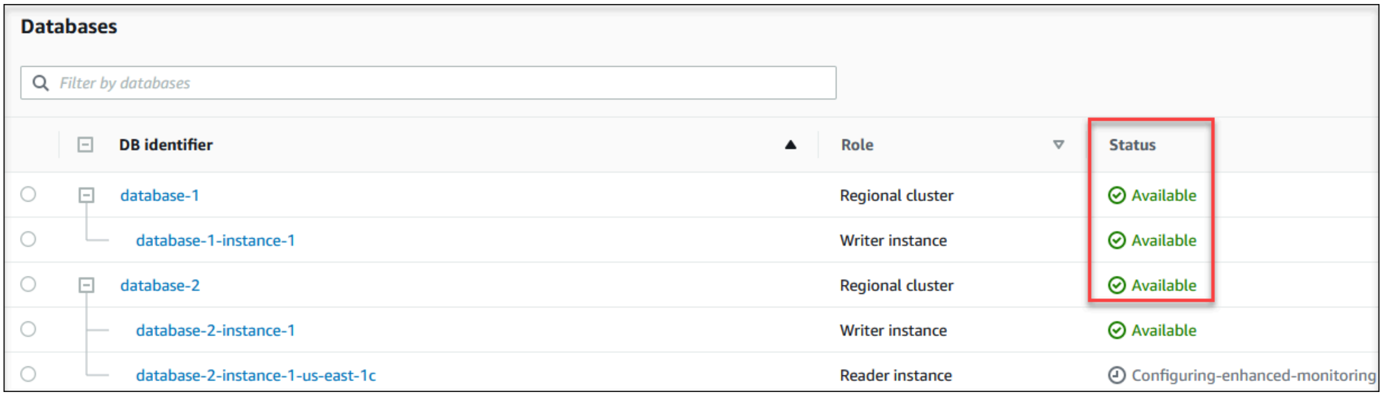
DB 인스턴스 상태	청구	설명
stopping	요금 부과 스토리지	DB 인스턴스를 중지 중입니다.
storage-config-upgrade	청구	DB 인스턴스의 스토리지 파일 시스템 구성이 업그레이드되고 있습니다. 이 상태는 블루/그린 배포 내의 그린 데이터베이스 또는 DB 인스턴스 읽기 전용 복제본에만 적용됩니다.
storage-full	청구	DB 인스턴스에 할당된 스토리지 용량이 거의 다 찼습니다. 이것은 중요한 상태이므로 즉각 이 문제를 수정하는 것이 좋습니다. 그러려면 DB 인스턴스를 수정하여 스토리지를 확장하십시오. 이러한 상황을 피하려면 스토리지 공간이 부족해지면 경고하도록 Amazon CloudWatch 경보를 설정하십시오.
storage-initialization	청구	DB 인스턴스는 스냅샷에서 복원된 후 볼륨 성능을 최적화하기 위해 Amazon S3에서 데이터 블록을 로드하고 있습니다. 작업에 사용할 수 있지만 초기화가 완료될 때까지 성능이 최대가 아닐 수 있습니다.
storage-optimization	청구	Amazon RDS가 DB 인스턴스의 스토리지를 최적화하고 있습니다. 스토리지 최적화 프로세스는 보통 짧지만, 가끔 최대 24시간까지 걸릴 수도 있습니다. 스토리지 최적화 중에는 DB 인스턴스를 계속 사용할 수 있습니다. 스토리지 최적화는 인스턴스의 가용성에 영향을 주지 않는 백그라운드 프로세스입니다.
upgrading	청구	데이터베이스 엔진 또는 운영 체제 버전은 현재 업그레이드 중입니다.

콘솔

DB 인스턴스의 상태를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

   DB 인스턴스 목록과 함께 데이터베이스 페이지가 표시됩니다. 클러스터의 각 DB 인스턴스에 대해 상태 값이 표시됩니다.



CLI

AWS CLI를 사용하여 DB 인스턴스 및 DB 인스턴스 상태 정보를 보려면 [describe-db-instances](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-db-instances.html) 명령을 사용하십시오. 예를 들어 다음 AWS CLI 명령은 모든 DB 인스턴스의 정보를 나열합니다.

```
aws rds describe-db-instances
```

특정 DB 인스턴스 및 DB 인스턴스 상태를 보려면 다음 옵션을 지정한 describe-db-instances 명령을 호출하십시오.

- `DBInstanceIdentifier` – DB 인스턴스의 이름입니다.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

모든 DB 인스턴스의 상태만 확인하려면 AWS CLI에서 다음 쿼리를 사용하십시오.

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

Amazon RDS API를 사용하여 DB 인스턴스 상태를 보려면 DescribeDBInstances 작업을 호출하십시오.

Amazon Aurora의 권장 사항

Amazon Aurora에서는 DB 인스턴스, DB 클러스터, DB 파라미터 그룹과 같은 데이터베이스 리소스에 대한 자동화된 권장 사항을 제공합니다. 이러한 권장 사항은 DB 클러스터 구성, DB 인스턴스 구성, 사용량 및 성능 데이터에 대한 분석을 통해 모범 사례 지침을 제공합니다.

Amazon RDS 성능 개선 도우미는 특정 지표를 모니터링하고 특정 리소스에 대해 문제가 될 수 있다고 간주되는 수준을 분석하여 임계값을 자동으로 생성합니다. 지정된 기간 동안 새 지표 값이 사전 정의된 임계값을 초과하면 성능 개선 도우미는 사전 예방적 권장 사항을 생성합니다. 이 권장 사항은 향후 데이터베이스 성능에 미치는 영향을 방지하는 데 도움이 됩니다. 예를 들어, 데이터베이스에 연결된 세션이 활성 작업을 수행하지 않지만 데이터베이스 리소스가 차단된 상태로 유지될 수 있는 경우 Aurora PostgreSQL 인스턴스에 대해 'Idle In Transaction' 권장 사항이 생성됩니다. 사전 예방적 권장 사항을 받으려면 유료 등급 보존 기간과 함께 성능 개선 도우미를 활성화해야 합니다. 성능 개선 도우미를 활성화하는 방법에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 섹션을 참조하세요. 성능 개선 도우미의 요금 및 데이터 보존에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.

DevOps Guru for RDS는 특정 지표를 모니터링하여 지표의 행동이 매우 비정상적이거나 이례적으로 전환되는 시점을 탐지합니다. 이러한 이상 현상은 권장 사항이 포함된 사후 대응적 인사이트로 보고됩니다. 예를 들어, DevOps Guru for RDS는 CPU 용량을 늘리거나 DB 로드에 기여하는 대기 이벤트를 조사할 것을 권장할 수 있습니다. DevOps Guru for RDS는 임계값 기반 사전 예방적 권장 사항도 제공합니다. 이러한 권장 사항을 적용하려면 DevOps Guru for RDS를 켜야 합니다. DevOps Guru for RDS를 켜는 방법에 대한 자세한 내용은 DevOps Guru 활성화 및 리소스 적용 범위 지정 섹션을 참조하세요.

권장 사항은 활성, 무시, 보류 또는 해결됨 상태 중 하나로 나타납니다. 해결된 권장 사항은 365일 동안 제공됩니다.

권장 사항을 보거나 무시할 수 있습니다. 즉시 구성 기반 활성 권장 사항을 적용하거나, 다음 유지 관리 기간으로 예약하거나, 무시할 수 있습니다. 임계값 기반 사전 예방 및 기계 학습 기반 사후 대응 권장 사항의 경우 문제의 제안된 원인을 검토한 다음 권장 조치를 수행하여 문제를 해결해야 합니다.

권장 사항은 다음 AWS 리전에서 지원됩니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오레곤)

- 아시아 태평양(뭄바이)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스톡홀름)
- 남아메리카(상파울루)

다음 섹션에서 Amazon Aurora의 권장 사항을 보고, 적용하고, 무시하고, 수정하는 방법을 알아보세요.

주제

- Amazon Aurora 권장 사항 보기
- Amazon Aurora 권장 사항 적용
- Amazon Aurora 권장 사항 무시
- 무시된 Amazon Aurora 권장 사항을 활성 권장 사항으로 수정
- Amazon Aurora 권장 사항 참조

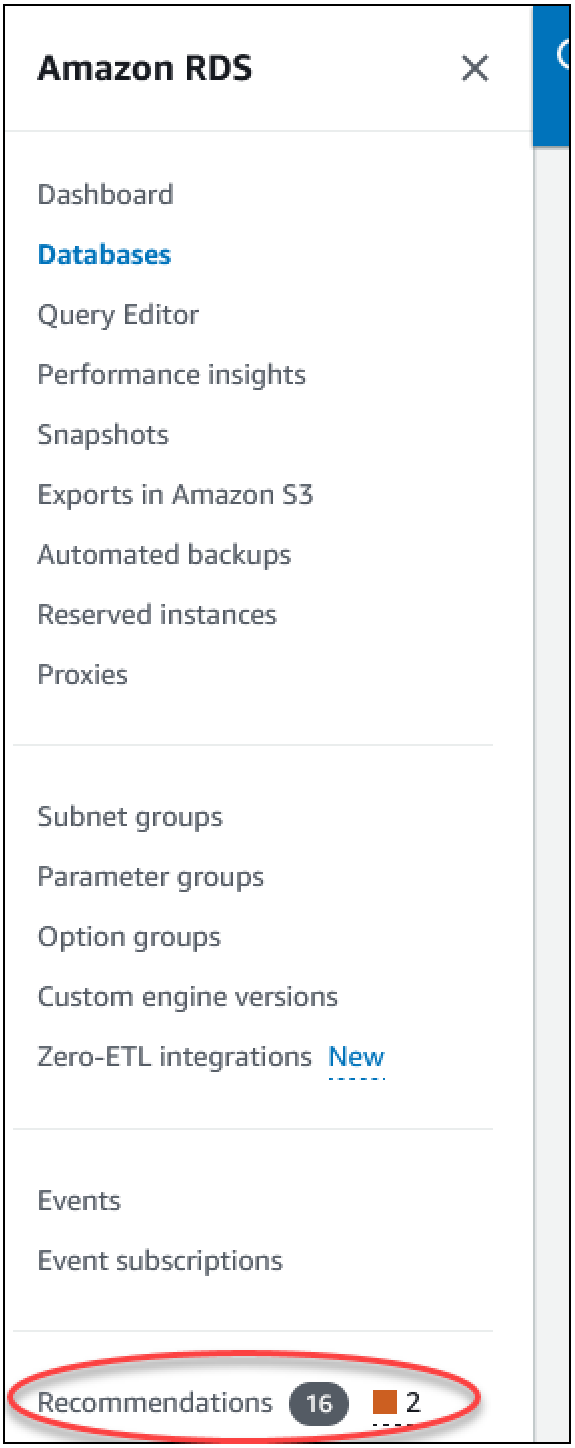
Amazon Aurora 권장 사항 보기

Amazon RDS 콘솔을 사용하면 데이터베이스 리소스에 대한 Amazon Aurora 권장 사항을 볼 수 있습니다. DB 클러스터의 경우 DB 클러스터 및 해당 인스턴스에 대한 권장 사항이 표시됩니다.

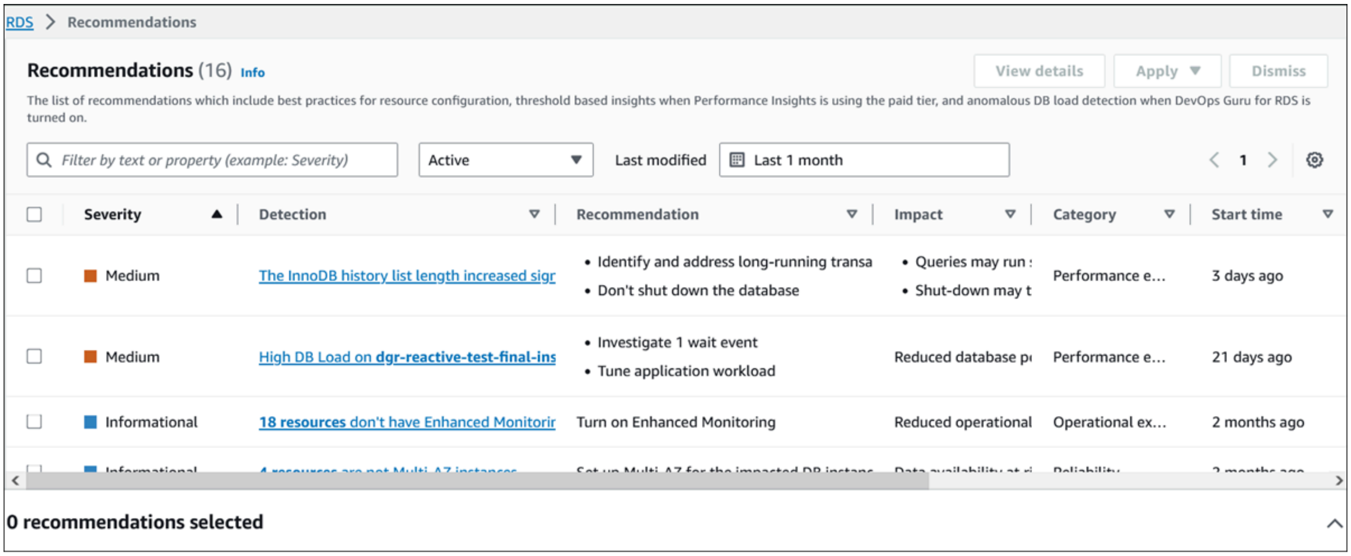
콘솔

Amazon Aurora 권장 사항을 확인하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 다음 중 하나를 수행합니다.
   - 권장 사항을 선택합니다. 리소스에 대한 활성 권장 사항 수와 지난달 생성된 심각도가 가장 높은 권장 사항 수는 권장 사항 옆에서 확인할 수 있습니다. 각 심각도에 대한 활성 권장 사항 수를 찾으려면 가장 높은 심각도를 나타내는 숫자를 선택하면 됩니다.

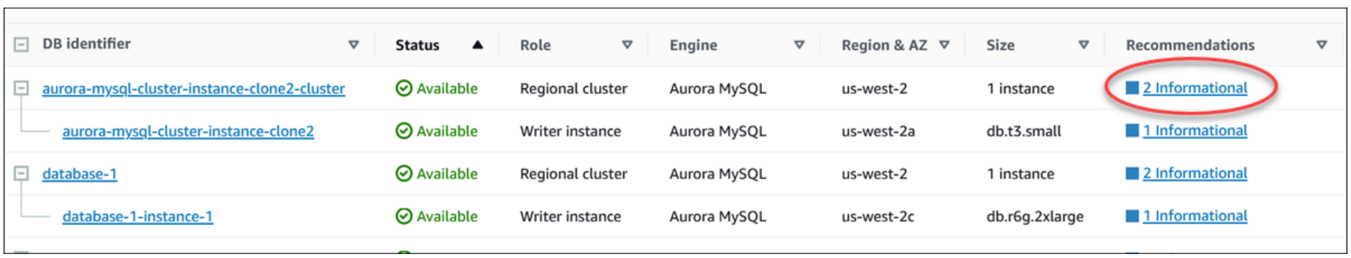


기본적으로 권장 사항 페이지에는 지난달의 새 권장 사항 목록이 표시됩니다. Amazon Aurora는 계정의 모든 리소스에 대해 권장 사항을 제공하고 심각도별로 권장 사항을 정렬합니다.

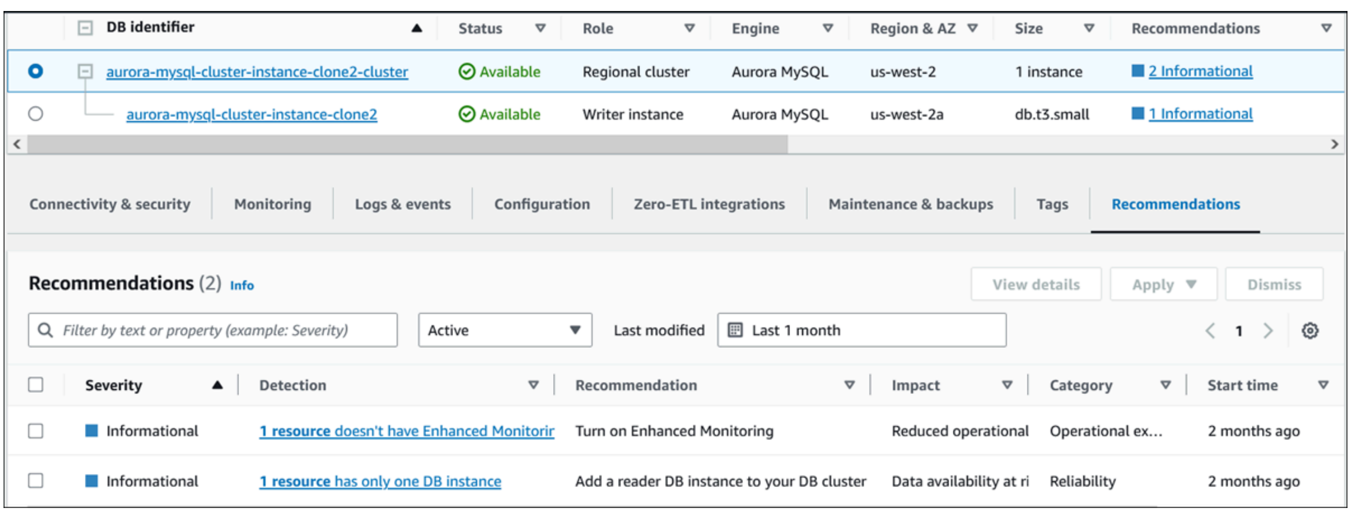


권장 사항을 선택하여 페이지 하단에서 영향을 받는 리소스와 권장 사항 적용 방법에 대한 세부 정보가 포함된 섹션을 볼 수 있습니다.

- 데이터베이스 페이지에서 리소스에 대한 권장 사항을 선택합니다.



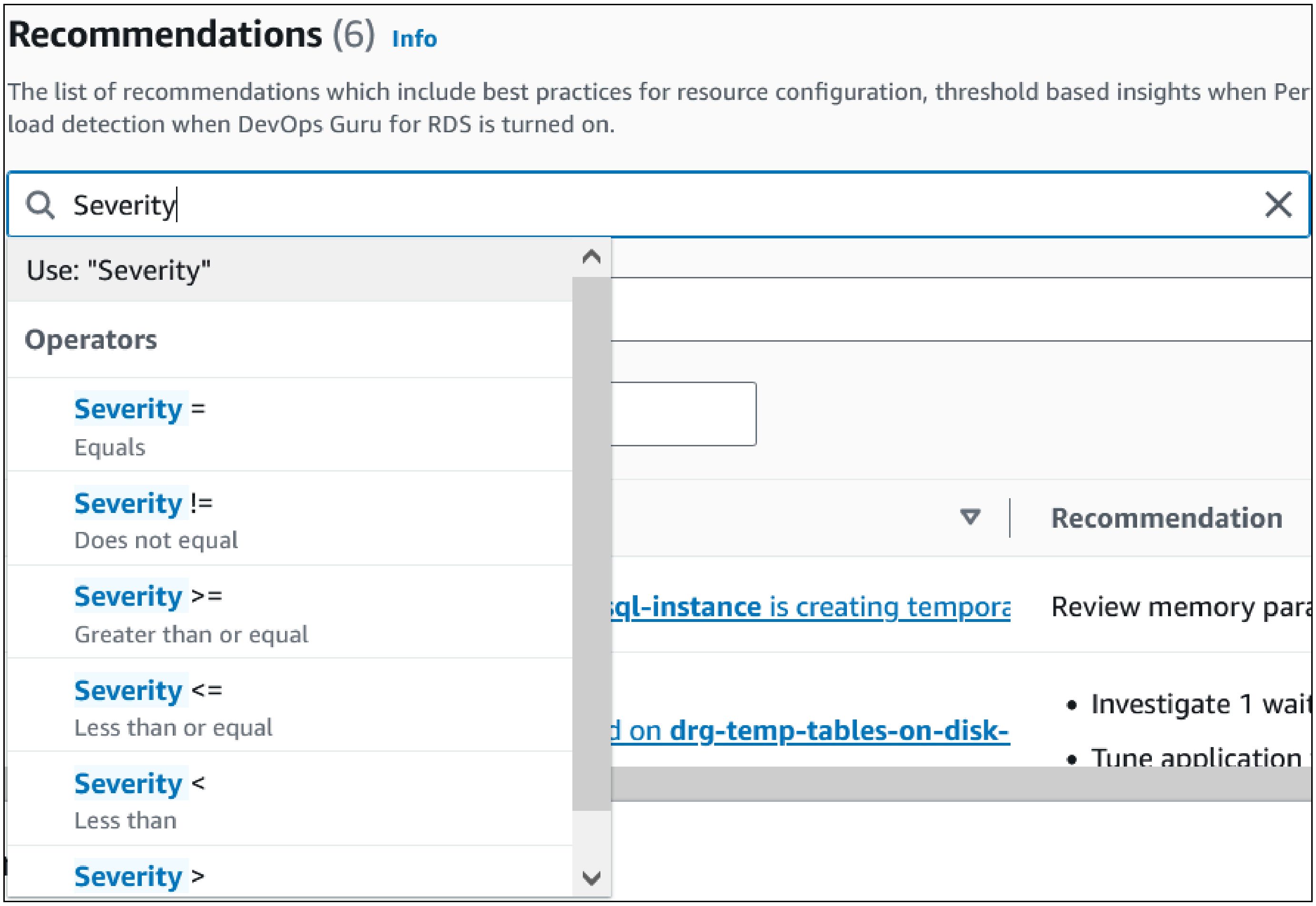
권장 사항 탭에는 선택한 리소스에 대한 권장 사항 및 세부 정보가 표시됩니다.



권장 사항에 대한 세부 정보는 다음과 같습니다.

- 심각도 - 문제의 영향 수준입니다. 심각도 수준은 높음, 보통, 낮음, 정보용으로 나뉩니다.
- 탐지 - 영향을 받는 리소스의 수와 문제에 대한 간략한 설명입니다. 권장 사항 및 분석 세부 정보를 보려면 이 링크를 선택하세요.
- 권장 사항 - 적용할 권장 조치에 대한 간략한 설명입니다.
- 영향 - 권장 사항이 적용되지 않을 때 발생할 수 있는 영향에 대한 간략한 설명입니다.
- 카테고리 - 권장 사항 유형입니다. 카테고리는 성능 효율성, 보안, 신뢰성, 비용 최적화, 운영 우수성, 지속 가능성입니다.
- 상태 - 현재 권장 사항의 상태입니다. 가능한 상태로는 모두, 활성, 무시, 해결됨, 보류 중이 있습니다.
- 시작 시간 - 문제가 시작된 시간입니다. 예를 들어, 18시간 전일 수 있습니다.
- 최종 수정 시간 - 심각도 변경으로 인해 시스템에서 권장 사항을 마지막으로 업데이트한 시간 또는 권장 사항에 응답한 시간입니다. 예를 들어, 10시간 전일 수 있습니다.
- 종료 시간 - 문제가 종료된 시간입니다. 계속되는 문제의 경우 시간이 표시되지 않습니다.
- 리소스 식별자 - 하나 이상의 리소스 이름입니다.

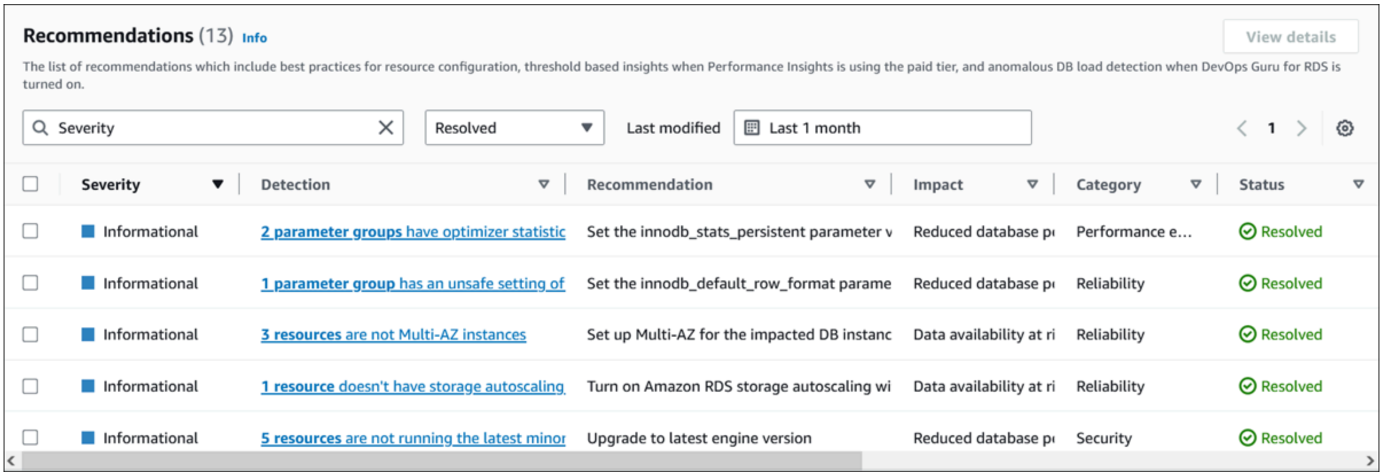
3. (선택 사항) 필드에서 심각도 또는 카테고리 연산자를 선택하여 권장 사항 목록을 필터링합니다.



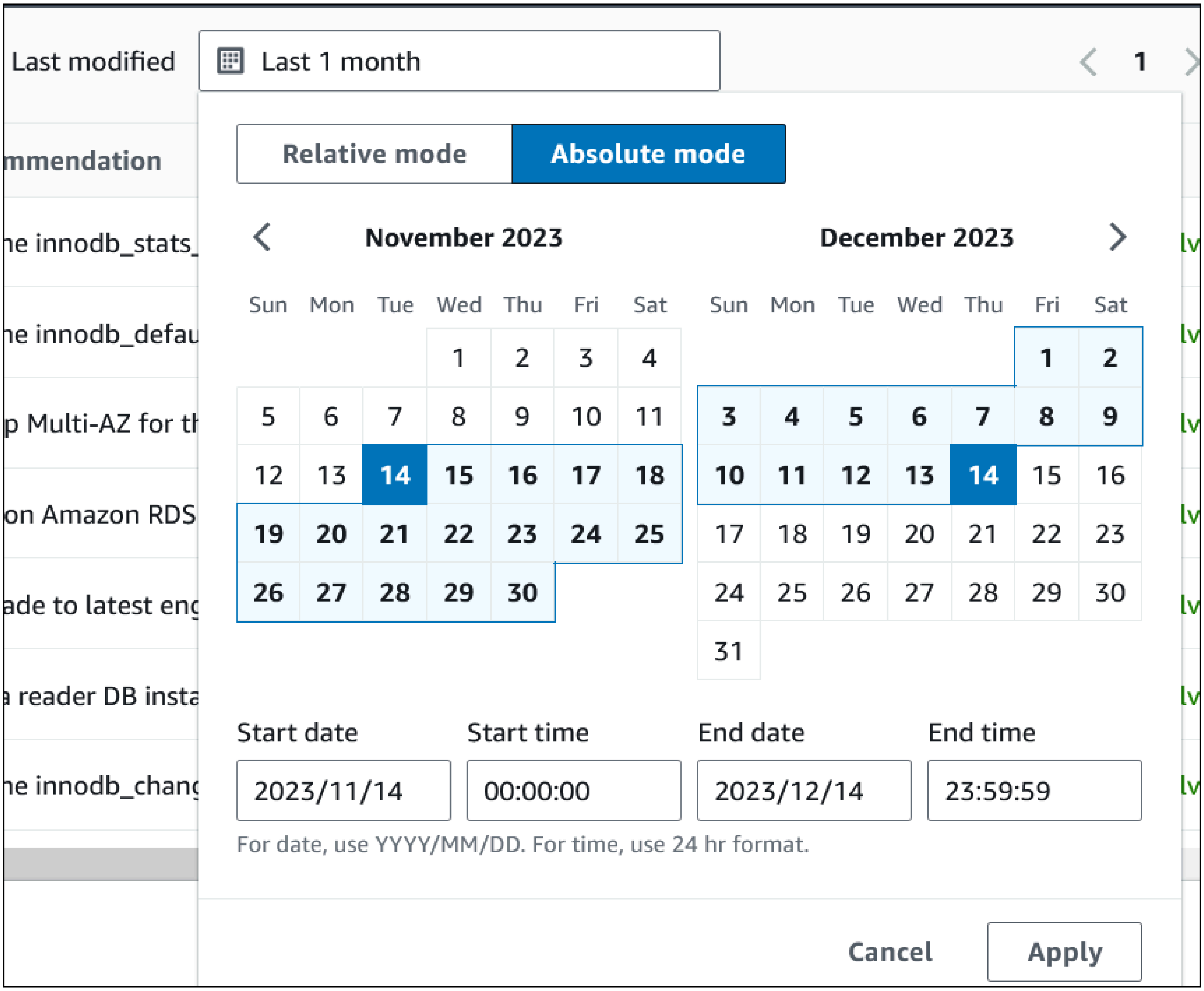
선택한 작업에 대한 권장 사항이 표시됩니다.

4. (선택 사항) 다음 권장 사항 상태 중 하나를 선택합니다.

- 활성(기본값) – 적용하거나, 다음 유지 관리 기간에 예약하거나, 무시할 수 있는 현재 권장 사항을 표시합니다.
- 모두 - 현재 상태의 모든 권장 사항을 표시합니다.
- 무시 - 무시된 권장 사항을 표시합니다.
- 해결됨 - 해결된 권장 사항을 표시합니다.
- 보류 중 - 권장 조치가 진행 중이거나 다음 유지 관리 기간에 예정되어 있는 권장 사항을 표시합니다.



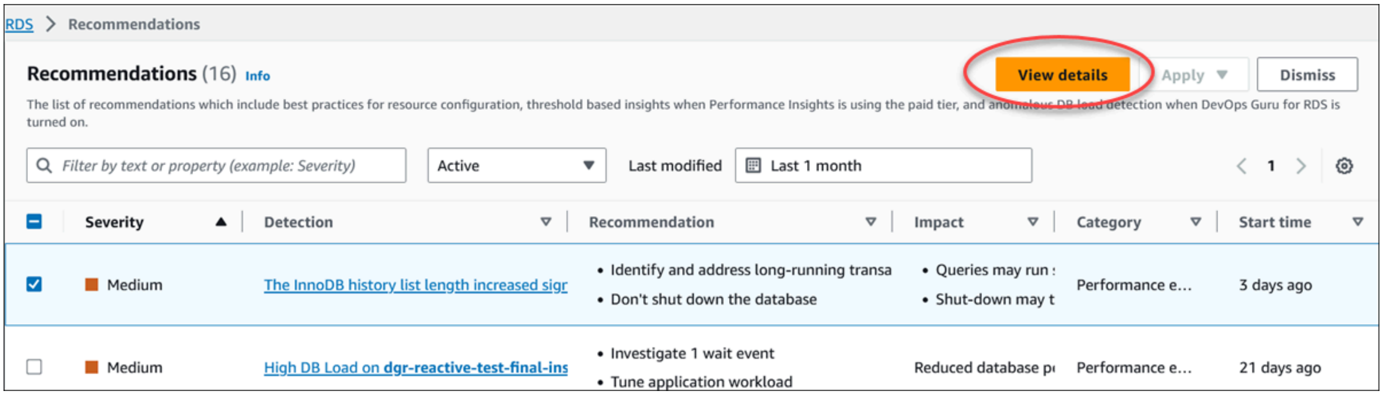
5. (선택 사항) 기간을 수정하려면 최종 수정 시간에서 상대 모드 또는 절대 모드를 선택합니다. 권장 사항 페이지에는 해당 기간에 생성된 권장 사항이 표시됩니다. 기본 기간은 지난달입니다. 절대 모드에서는 기간을 선택하거나 시작 날짜 및 종료 날짜 필드에 시간을 입력할 수 있습니다.



설정된 기간에 대한 권장 사항이 표시됩니다.

범위를 전체로 설정하면 계정 내 리소스에 대한 모든 권장 사항을 볼 수 있다는 점을 참고하세요.

6. (선택 사항) 오른쪽에서 기본 설정을 선택하여 표시할 세부 정보를 사용자 지정합니다. 페이지 크기를 선택하고, 텍스트 줄을 줄 바꿈하고, 열을 허용하거나 숨길 수 있습니다.
7. (선택 사항) 권장 사항을 선택한 다음 세부 정보 보기를 선택합니다.



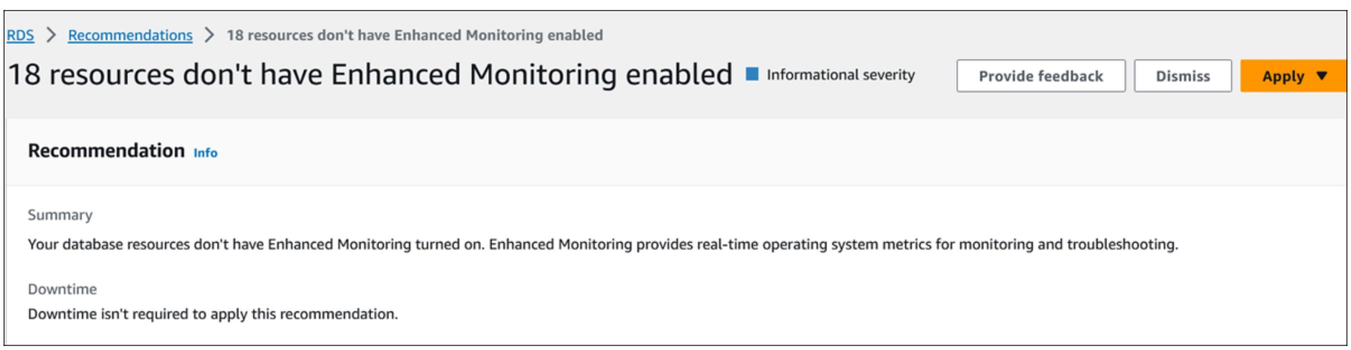
권장 사항 세부 정보 페이지가 표시됩니다. 제목은 발견된 문제가 있는 리소스의 총 개수와 심각도를 제공합니다.

이상 항목 기반 사후 대응 권장 사항의 세부 정보 페이지에 있는 구성 요소에 대한 자세한 내용은 Amazon DevOps Guru 사용 설명서의 Viewing reactive anomalies를 참조하세요.

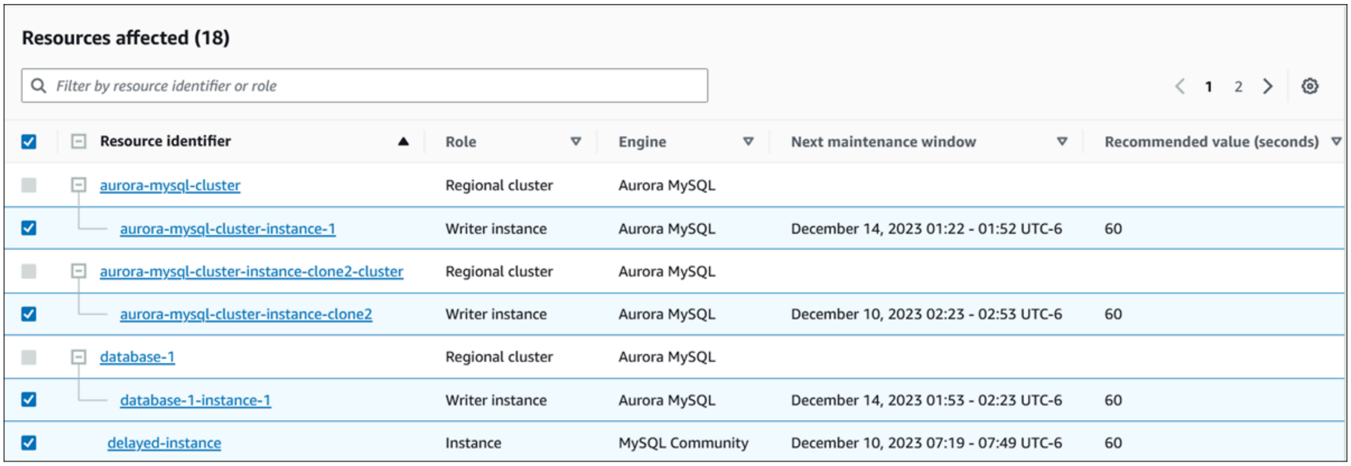
임계값 기반 사전 권장 사항의 세부 정보 페이지에 있는 구성 요소에 대한 자세한 내용은 성능 개선 도우미 사전 권장 사항 보기 섹션을 참조하세요.

다른 자동 권장 사항은 권장 사항 세부 정보 페이지에 다음 구성 요소를 표시합니다.

- 권장 사항 - 권장 사항에 대한 요약과 권장 사항을 적용하는 데 가동 중지가 필요한지 여부를 표시합니다.



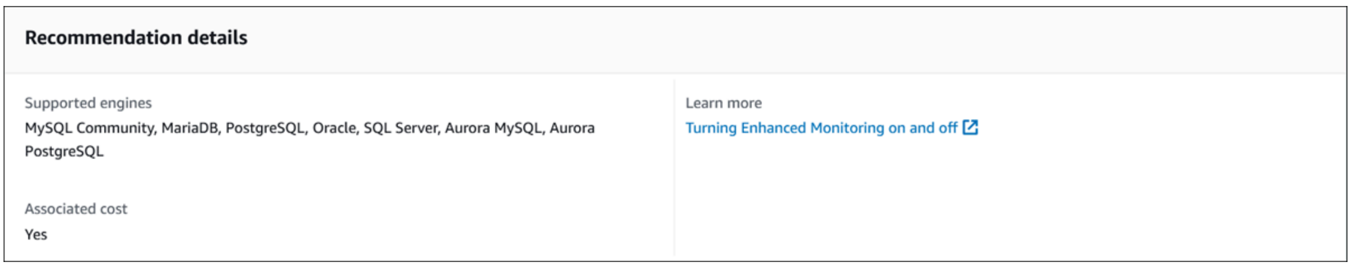
- 영향을 받는 리소스 - 영향을 받는 리소스의 세부 정보입니다.



Resources affected (18)

	Resource identifier	Role	Engine	Next maintenance window	Recommended value (seconds)
	aurora-mysql-cluster	Regional cluster	Aurora MySQL		
☑	aurora-mysql-cluster-instance-1	Writer instance	Aurora MySQL	December 14, 2023 01:22 - 01:52 UTC-6	60
	aurora-mysql-cluster-instance-clone2-cluster	Regional cluster	Aurora MySQL		
☑	aurora-mysql-cluster-instance-clone2	Writer instance	Aurora MySQL	December 10, 2023 02:23 - 02:53 UTC-6	60
	database-1	Regional cluster	Aurora MySQL		
☑	database-1-instance-1	Writer instance	Aurora MySQL	December 14, 2023 01:53 - 02:23 UTC-6	60
☑	delayed-instance	Instance	MySQL Community	December 10, 2023 07:19 - 07:49 UTC-6	60

- 권장 사항 세부 정보 - 지원되는 엔진 정보, 권장 사항을 적용하는 데 필요한 모든 관련 비용, 자세히 알아볼 수 있는 설명서 링크가 나와 있습니다.



CLI

DB 인스턴스 또는 DB 클러스터의 Amazon RDS 권장 사항을 보려면 AWS CLI에서 다음 명령을 사용하세요.

```
aws rds describe-db-recommendations
```

RDS API

Amazon RDS API를 사용하여 Amazon RDS 권장 사항을 보려면 DescribeDBRecommendations 작업을 사용하세요.

Amazon Aurora 권장 사항 적용

Amazon RDS 콘솔을 사용하여 Amazon Aurora 권장 사항을 적용하려면 세부 정보 페이지에서 구성 기반 권장 사항 또는 영향을 받는 리소스를 선택하세요. 그런 다음 권장 사항을 즉시 적용하거나 다음 유지 관리 기간으로 미루도록 선택하세요. 변경 사항을 적용하려면 리소스를 다시 시작해야 할 수 있습니다. 일부 DB 파라미터 그룹 권장 사항의 경우 리소스를 다시 시작해야 할 수 있습니다.

임계값 기반 사전 예방적 권장 사항이나 이상 항목 기반 사후 권장 사항에는 적용 옵션이 없으므로, 추가 검토가 필요할 수 있습니다.

콘솔

구성 기반 권장 사항을 적용하려면

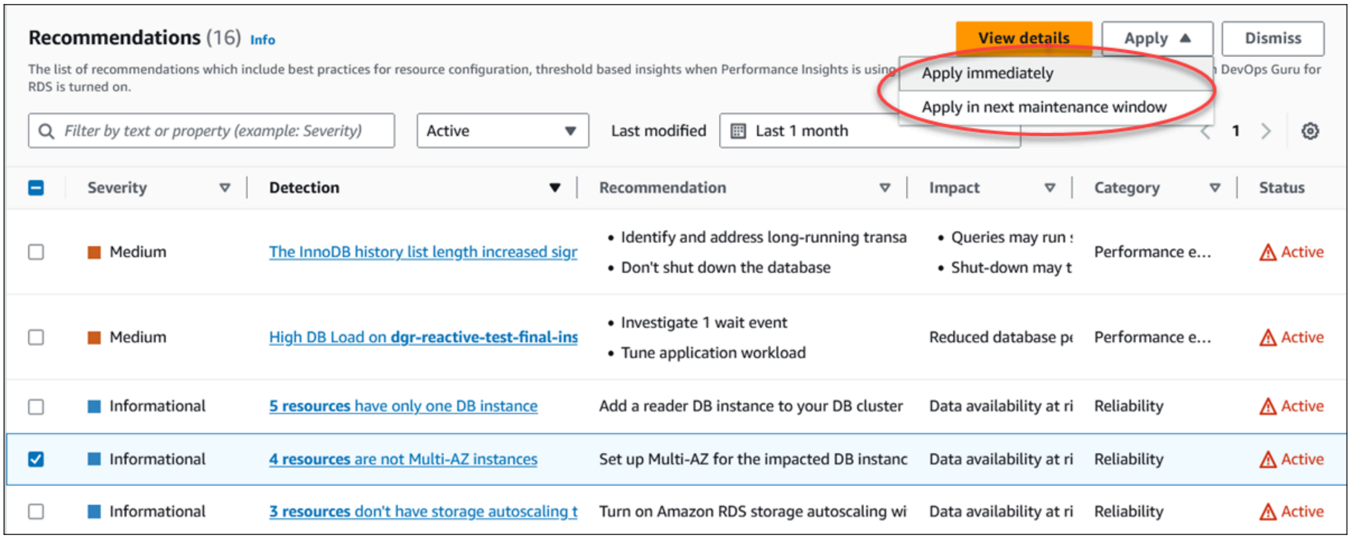
1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 다음 중 하나를 수행합니다.
   - 권장 사항을 선택합니다.

     모든 권장 사항 목록과 함께 권장 사항 페이지가 나타납니다.
   - 데이터베이스를 선택한 다음 데이터베이스 페이지에서 리소스에 대한 권장 사항을 선택합니다.

     선택한 권장 사항의 권장 사항 탭에 세부 정보가 표시됩니다.
   - 권장 사항 페이지 또는 데이터베이스 페이지의 권장 사항 탭에서 활성 권장 사항에 대한 탐지를 선택합니다.

     권장 사항 세부 정보 페이지가 표시됩니다.
3. 권장 사항 세부 정보 페이지에서 권장 사항 또는 영향을 받는 리소스를 하나 이상 선택하고 다음 중 하나를 수행합니다.
   - 적용을 선택한 다음 즉시 적용을 선택하여 권장 사항을 즉시 적용합니다.
   - 적용을 선택하고 다음 유지 관리 기간에 적용을 선택하여 다음 유지 관리 기간에 일정을 예약합니다.

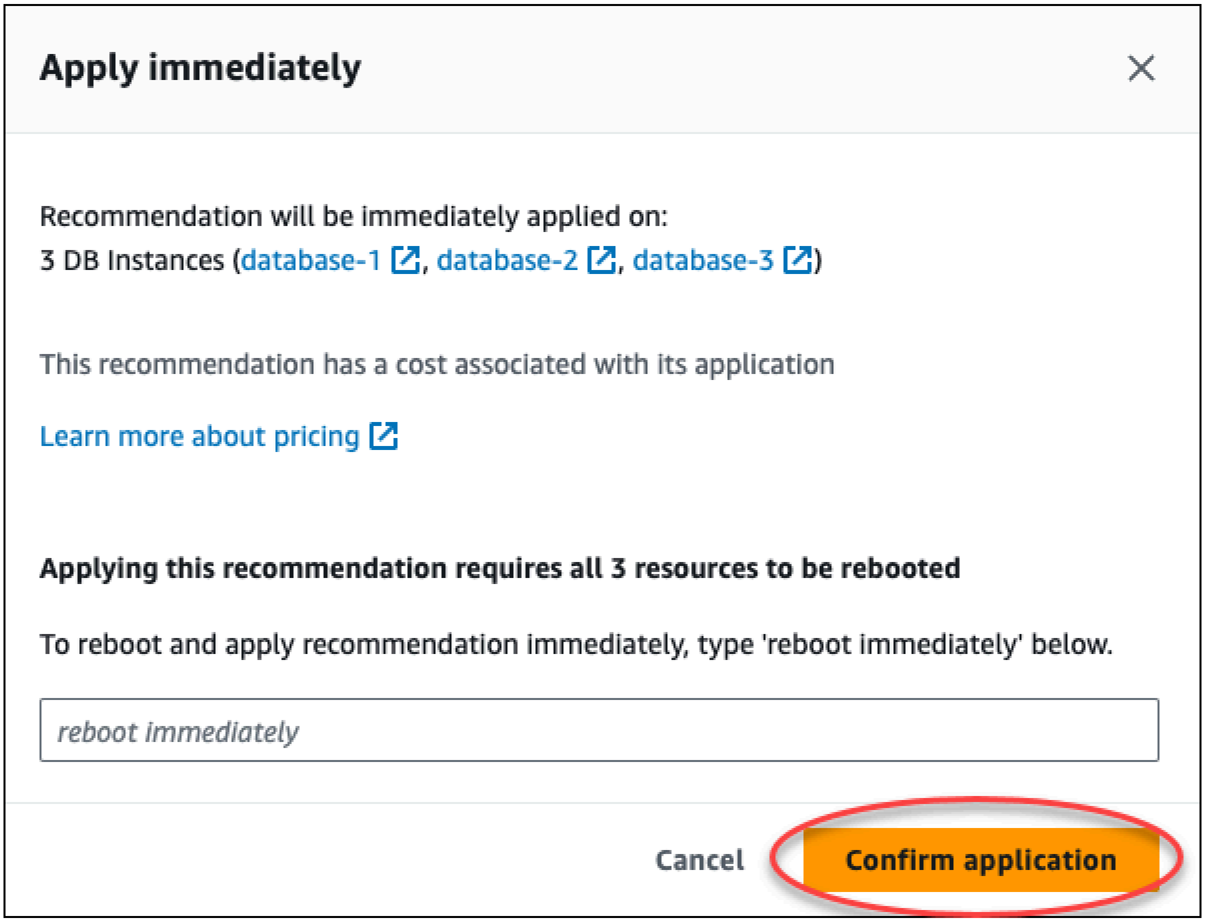
     선택한 권장 사항 상태는 다음 유지 관리 기간까지 보류 중으로 업데이트됩니다.



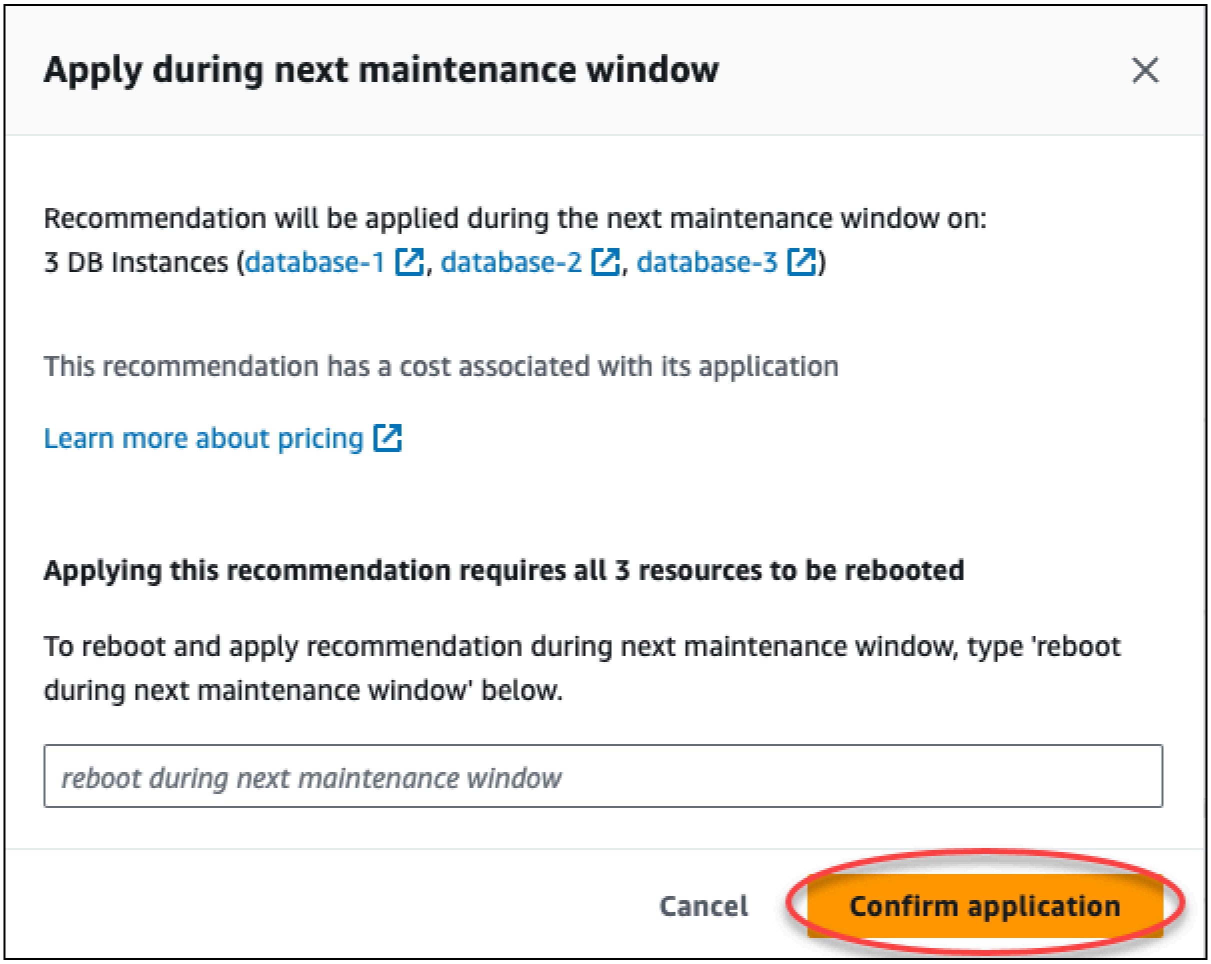
확인 창이 나타납니다.

4. 애플리케이션 확인을 선택하여 권장 사항을 적용합니다. 이 창은 변경 사항을 적용하기 위해 리소스를 자동으로 다시 시작해야 할지, 수동으로 다시 시작해야 할지 확인합니다.

   다음 예제에서는 권장 사항을 즉시 적용하기 위한 확인 창을 보여줍니다.



다음 예제는 다음 유지 관리 기간에 권장 사항을 적용하기로 예약하는 확인 창을 보여줍니다.



배너에 권장 사항이 적용되거나 적용에 실패하면 메시지가 표시됩니다.

다음 예제에서는 성공 메시지가 포함된 배너를 보여줍니다.

다음 예제에서는 실패 메시지가 포함된 배너를 보여줍니다.

RDS API

Amazon RDS API를 사용하여 구성 기반 Aurora 권장 사항을 적용하려면

1. DescribeDBRecommendations 작업을 사용합니다. 출력의 `RecommendedActions`에 하나 이상의 권장 조치가 포함될 수 있습니다.
2. 1단계의 각 권장 조치에 대해 RecommendedAction 객체를 사용합니다. 출력에는 `Operation` 및 `Parameters`가 포함됩니다.

   다음 예제는 하나 이상의 권장 조치가 포함된 출력을 보여줍니다.

```
    "RecommendedActions": [
        {
            "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
            "Title": "Turn on auto backup", // localized
            "Description": "Turn on auto backup for my-mysql-instance-1", //
 localized
            "Operation": "ModifyDbInstance",
            "Parameters": [
                {
                    "Key": "DbInstanceIdentifier",
                    "Value": "my-mysql-instance-1"
                },
                {
                    "Key": "BackupRetentionPeriod",
                    "Value": "7"
                }
            ],
            "ApplyModes": ["immediately", "next-maintenance-window"],
            "Status": "applied"
        },
        ... // several others
    ],
```

3. 2단계의 출력에서 각 권장 조치에 대해 `operation`을 사용하고 `Parameters` 값을 입력합니다.
4. 2단계의 작업이 성공하면 ModifyDBRecommendation 작업을 사용하여 권장 사항 상태를 수정합니다.

Amazon Aurora 권장 사항 무시

Amazon RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 하나 이상의 Amazon Aurora 권장 사항을 무시할 수 있습니다.

콘솔

하나 이상의 권장 사항을 무시하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 다음 중 하나를 수행합니다.

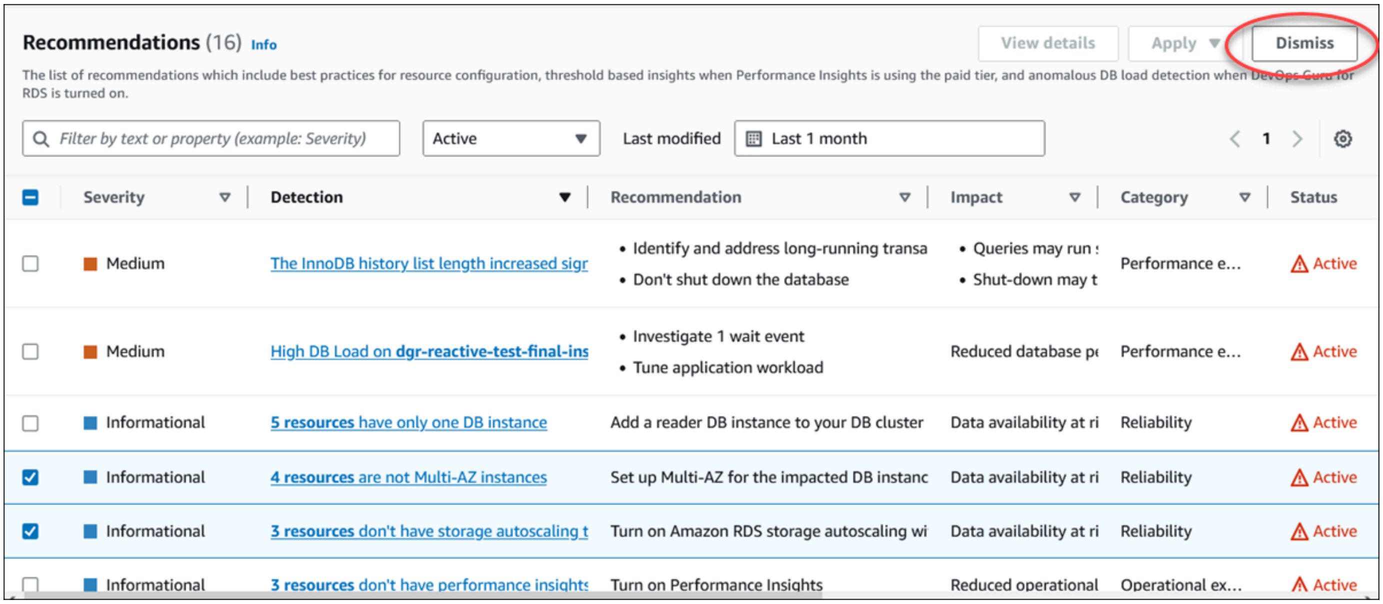
   - 권장 사항을 선택합니다.

     모든 권장 사항 목록과 함께 권장 사항 페이지가 나타납니다.
   - 데이터베이스를 선택한 다음 데이터베이스 페이지에서 리소스에 대한 권장 사항을 선택합니다.

     선택한 권장 사항의 권장 사항 탭에 세부 정보가 표시됩니다.
   - 권장 사항 페이지 또는 데이터베이스 페이지의 권장 사항 탭에서 활성 권장 사항에 대한 탐지를 선택합니다.

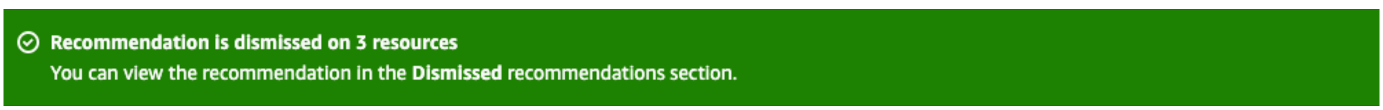
     권장 사항 세부 정보 페이지에는 영향을 받는 리소스 목록이 표시됩니다.
3. 권장 사항 세부 정보 페이지에서 하나 이상의 권장 사항 또는 영향을 받는 리소스를 선택한 다음 무시를 선택합니다.

   다음 예제는 무시하기로 선택한 활성 권장 사항이 여러 개 있는 권장 사항 페이지를 보여줍니다.



선택한 하나 이상의 권장 사항이 무시되면 배너에 메시지가 표시됩니다.

다음 예제에서는 성공 메시지가 포함된 배너를 보여줍니다.



다음 예제에서는 실패 메시지가 포함된 배너를 보여줍니다.

CLI

AWS CLI를 사용하여 Aurora 권장 사항을 무시하려면

1. `aws rds describe-db-recommendations --filters "Name=status,Values=active"` 명령을 실행합니다.

   출력에서 `active` 상태에 있는 권장 사항의 목록을 제공합니다.

2. 1단계에서 무시하려는 권장 사항에 대한 `recommendationId`를 찾습니다.

3. 2단계에서 `recommendationId`와 함께 `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` 명령을 실행하여 권장 사항을 무시합니다.

RDS API

Amazon RDS API를 사용하여 Aurora 권장 사항을 무시하려면 ModifyDBRecommendation 작업을 사용합니다.

무시된 Amazon Aurora 권장 사항을 활성 권장 사항으로 수정

Amazon RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 무시된 Amazon Aurora 권장 사항 하나 이상을 활성 권장 사항으로 변경할 수 있습니다.

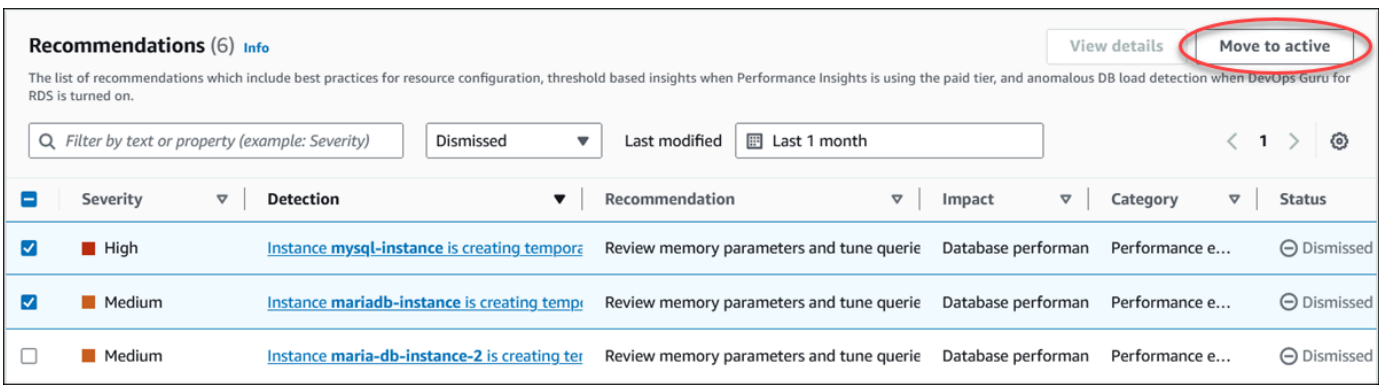
콘솔

하나 이상의 무시된 권장 사항을 활성 권장 사항으로 이동하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 다음 중 하나를 수행합니다.
   - 권장 사항을 선택합니다.

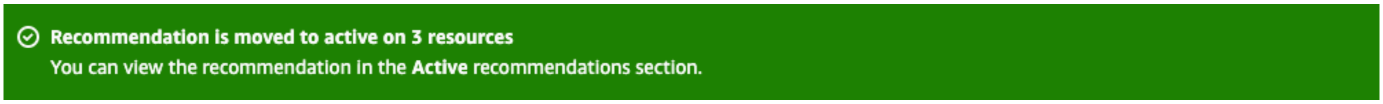
     권장 사항 페이지에는 계정 내 모든 리소스의 심각도별로 정렬된 권장 사항 목록이 표시됩니다.
   - 데이터베이스를 선택한 다음 데이터베이스 페이지에서 리소스에 대한 권장 사항을 선택합니다.

     권장 사항 탭에는 선택한 리소스에 대한 권장 사항 및 세부 정보가 표시됩니다.
3. 목록에서 무시된 권장 사항을 하나 이상 선택한 다음 활성으로 이동을 선택합니다.



선택한 권장 사항을 무시 상태에서 활성 상태로 이동하면 배너에 성공 또는 실패 메시지가 표시됩니다.

다음 예제에서는 성공 메시지가 포함된 배너를 보여줍니다.



다음 예제에서는 실패 메시지가 포함된 배너를 보여줍니다.

CLI

AWS CLI를 사용하여 무시된 Aurora 권장 사항을 활성 권장 사항으로 변경하려면

1. `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"` 명령을 실행합니다.

   출력에서 `dismissed` 상태에 있는 권장 사항의 목록을 제공합니다.
2. 1단계에서 상태를 변경하려는 권장 사항에 대한 `recommendationId`를 찾습니다.
3. 2단계에서 `recommendationId`와 함께 `>aws rds modify-db-recommendation --status active --recommendationId <ID>` 명령을 실행하여 활성 권장 사항으로 변경합니다.

RDS API

Amazon RDS API를 사용하여 무시된 Aurora 권장 사항을 활성 권장 사항으로 변경하려면 ModifyDBRecommendation 작업을 사용합니다.

Amazon Aurora 권장 사항 참조

Amazon Aurora는 리소스가 생성되거나 수정될 때 리소스에 대해 권장 사항을 생성합니다. 다음 표에서 Amazon Aurora의 권장 사항 예제를 찾을 수 있습니다.

유형	설명	권장 사항	가동 중지 필요	추가 정보
리소스 자동 백업이 비활	DB 인스턴스에 대해 자동 백업이 활성화되어 있지 않습니다. 자	보존 기간이 최대 14일인 자동 백업을 활성화하세요.	예	Aurora DB 클러스터 백업 및 복원에 대한 개요

유형	설명	권장 사항	가동 중지 필요	추가 정보
성화되어 있습니다.	동 백업은 DB 인스턴스의 특정 시점 복구를 가능하게 하므로, 권장됩니다.			AWS 데이터베이스 블로그의 Demystifying Amazon RDS backup storage costs
엔진 마이너 버전 업그레이드가 필요합니다.	데이터베이스 리소스가 최신 마이너 DB 엔진 버전을 실행하지 않습니다. 최신 마이너 버전에는 최신 보안 수정 및 기타 개선 사항이 포함되어 있습니다.	최신 엔진 버전으로 업그레이드하세요.	예	Amazon Aurora DB 클러스터 유지 관리
향상된 모니터링이 꺼졌습니다.	데이터베이스 리소스에 향상된 모니터링이 켜져 있지 않습니다. 확장된 모니터링은 모니터링 및 문제 해결을 위해 실시간 운영 체제 지표를 제공합니다.	향상된 모니터링을 켜세요.	No	Enhanced Monitoring을 사용하여 OS 지표 모니터링

유형	설명	권장 사항	가동 중지 필요	추가 정보
스토리지 암호화가 비활성화되어 있습니다.	Amazon RDS는 AWS Key Management Service(AWS KMS)에서 관리하는 키를 사용하여 모든 데이터베이스 엔진에 대한 저장 중 암호화를 지원합니다. Amazon RDS 암호화를 사용하는 활성 DB 인스턴스에서는 스토리지에 저장된 데이터가 자동 백업, 읽기 전용 복제본 및 스냅샷과 마찬가지로 암호화됩니다. Aurora DB 클러스터를 생성할 때 암호화를 켜지 않은 경우 암호 해독된 스냅샷을 암호화된 DB 클러스터로 복원해야 합니다.	DB 클러스터에 저장된 데이터의 암호화를 활성화하세요.	예	[Amazon Aurora의 보안]()
동일한 가용 영역에 있는 DB 클러스터의 모든 인스턴스	DB 클러스터는 현재 단일 가용 영역에 있습니다. 여러 가용 영역을 사용하여 가용성을 개선하세요.	DB 클러스터의 여러 가용 영역에 DB 인스턴스를 추가합니다.	No	[Amazon Aurora의 고가용성]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
이기종 인스턴스 크기를 가진 클러스터의 DB 인스턴스	DB 클러스터의 모든 DB 인스턴스에 대해 동일한 DB 인스턴스 클래스와 크기를 사용하는 것이 좋습니다.	DB 클러스터의 모든 DB 인스턴스에 대해 동일한 인스턴스 클래스와 크기를 사용하세요.	예	[Amazon Aurora를 사용한 복제]
이기종 인스턴스 클래스를 가진 클러스터의 DB 인스턴스	DB 클러스터의 모든 DB 인스턴스에 대해 동일한 DB 인스턴스 클래스와 크기를 사용하는 것이 좋습니다.	DB 클러스터의 모든 DB 인스턴스에 대해 동일한 인스턴스 클래스와 크기를 사용하세요.	예	[Amazon Aurora를 사용한 복제]
이기종 파라미터 그룹이 있는 클러스터의 DB 인스턴스	DB 클러스터의 모든 DB 인스턴스가 동일한 DB 파라미터 그룹을 사용하는 것이 좋습니다.	DB 인스턴스를 DB 클러스터의 라이터 인스턴스와 연결된 DB 파라미터 그룹과 연결하세요.	No	[Amazon Aurora의 파라미터 그룹]
Amazon RDS DB 클러스터에 DB 인스턴스가 하나 있습니다.	DB 클러스터에 DB 인스턴스를 하나 이상 추가하여 가용성과 성능을 개선하세요.	DB 클러스터에 리더 DB 인스턴스를 추가합니다.	No	[Amazon Aurora의 고가용성]
성능 개선 도우미가 비활성화되어 있습니다.	성능 개선 도우미는 데이터베이스 성능 문제를 분석하고 해결하는 데 도움이 되는 DB 인스턴스 로드를 모니터링합니다. 성능 개선 도우미를 켜는 것이 좋습니다.	성능 개선 도우미를 활성화합니다.	No	[성능 개선 도우미를 통한 Amazon Aurora 모니터링]

유형	설명	권장 사항	가동 중지 필요	추가 정보
RDS 리소스 메이저 버전 업데이트가 필요합니다.	현재 DB 엔진용 메이저 버전이 설치된 데이터베이스는 지원되지 않습니다. 새 기능과 개선 사항이 포함된 최신 메이저 버전으로 업그레이드하는 것이 좋습니다.	DB 엔진의 최신 메이저 버전으로 업그레이드하세요.	예	Amazon Aurora 업데이트 Amazon Aurora에서 블루/그린 배포 생성
DB 클러스터가 최대 64TiB 볼륨만 지원합니다.	DB 클러스터가 최대 64TiB의 볼륨을 지원합니다. 최신 엔진 버전은 DB 클러스터에 최대 128TiB의 볼륨을 지원합니다. DB 클러스터의 엔진 버전을 최신 버전으로 업그레이드하여 최대 128TiB의 볼륨을 지원하는 것이 좋습니다.	DB 클러스터의 엔진 버전을 업그레이드하여 최대 128TiB의 볼륨을 지원하세요.	예	Amazon Aurora 크기 제한

유형	설명	권장 사항	가동 중지 필요	추가 정보
동일한 가용 영역에 있는 DB 클러스터의 모든 리더 인스턴스	가용 영역은 각 AWS 리전 내에서 중단이 발생할 경우 격리를 제공하기 위해 서로 구별되는 위치를 나타냅니다. DB 클러스터의 가용성을 개선하려면 여러 AZ에 걸쳐 있는 DB 클러스터에 기본 인스턴스와 리더 인스턴스를 배포하는 것이 좋습니다. 클러스터를 생성할 때 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 다중 AZ 클러스터를 생성할 수 있습니다. 새 리더 인스턴스를 추가하고 다른 AZ를 지정하여 기존 Aurora 클러스터를 다중 AZ 클러스터로 수정할 수 있습니다.	DB 클러스터의 모든 읽기 인스턴스가 동일 가용 영역에 있습니다. 리더 인스턴스를 여러 가용 영역에 걸쳐 배포하는 것이 좋습니다. 여러 가용 영역에 배포하면 가용성이 높아지고 클라이언트와 데이터베이스 간의 네트워크 지연 시간을 줄여 응답 시간이 개선됩니다.	No	[Amazon Aurora의 고가용성]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
DB 메모리 파라미터가 기본값과 다릅니다.	DB 인스턴스의 메모리 파라미터가 기본값과 크게 다릅니다. 이러한 설정은 성능에 영향을 미치고 오류를 일으킬 수 있습니다. DB 인스턴스에 대한 사용자 지정 메모리 파라미터를 DB 파라미터 그룹의 기본값으로 재설정하는 것이 좋습니다.	메모리 파라미터를 기본값으로 재설정하세요.	No	[Amazon Aurora의 파라미터 그룹]()
쿼리 캐시 파라미터가 켜져 있습니다.	변경으로 인해 쿼리 캐시를 제거해야 하는 경우 DB 인스턴스가 정지된 것처럼 보입니다. 쿼리 캐시는 대부분의 워크로드에 이점이 되지 못합니다. 쿼리 캐시는 MySQL 8.0 이상 버전에서 제거되었습니다. query_cache_type 파라미터를 0으로 설정하는 것이 좋습니다.	DB 파라미터 그룹에서 `query_cache_type` 파라미터 값을 `0`으로 설정하세요.	예	[Amazon Aurora의 파라미터 그룹]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
log_output 파라미터가 표로 설정되었습니다.	log_output 을 TABLE로 설정하면 log_output 을 FILE로 설정한 경우보다 더 많은 스토리지가 사용됩니다. 스토리지 크기 제한에 도달하지 않도록 파라미터를 FILE로 설정하는 것이 좋습니다. MySQL 8.4 이상 버전에서 기본적으로 FILE로 설정되어 있습니다.	DB 파라미터 그룹에서 log_output 파라미터 값을 FILE으로 설정하세요.	No	[Aurora MySQL 데이터베이스 로그 파일](#)
autovacuum 파라미터가 비활성화되어 있습니다.	DB 클러스터의 autovacuum 파라미터가 비활성화되어 있습니다. autovacuum 기능을 비활성화하면 표 및 인덱스 팽창이 증가하고 성능에 영향을 미칩니다. DB 파라미터 그룹에서 autovacuum을 켜는 것이 좋습니다.	DB 클러스터 파라미터 그룹에서 autovacuum 파라미터를 활성화하세요.	No	AWS 데이터베이스 블로그의 [Understanding autovacuum in Amazon RDS for PostgreSQL environments](#)

유형	설명	권장 사항	가동 중지 필요	추가 정보
`synchronous_commit` 파라미터가 비활성화되어 있습니다.	`synchronous_commit` 파라미터를 끄면 데이터베이스 충돌로 인해 데이터가 손실될 수 있습니다. 데이터베이스의 내구성에 악영향을 미칠 수 있습니다. `synchronous_commit` 파라미터를 활성화하는 것이 좋습니다.	DB 파라미터 그룹에서 `synchronous_commit` 파라미터를 활성화하세요.	예	AWS 데이터베이스 블로그의 Amazon Aurora PostgreSQL parameters: Replication, security, and logging
`track_counts` 파라미터가 비활성화되어 있습니다.	`track_counts` 파라미터를 비활성화하면 데이터베이스에서 데이터베이스 활동 통계를 수집하지 않습니다. Autovacuum을 사용하려면 이러한 통계가 제대로 작동해야 합니다. `track_counts` 파라미터를 1으로 설정하는 것이 좋습니다.	`track_counts` 파라미터를 1로 설정하세요.	No	PostgreSQL의 런타임 통계

유형	설명	권장 사항	가동 중지 필요	추가 정보
enable_indexonlyscan 파라미터가 비활성화되어 있습니다.	인덱스 전용 스캔 계획 유형이 비활성화되어 있으면 쿼리 플래너 또는 옵티마이저가 인덱스 전용 스캔 계획 유형을 사용할 수 없습니다. enable_indexonlyscan 파라미터 값을 1으로 설정하는 것이 좋습니다.	enable_indexonlyscan 파라미터 값을 1로 설정하세요.	No	[PostgreSQL용 플래너 메서드 구성]()
enable_indexscan 파라미터가 비활성화되어 있습니다.	인덱스 스캔 계획 유형이 비활성화되어 있으면 쿼리 플래너 또는 옵티마이저가 인덱스 스캔 계획 유형을 사용할 수 없습니다. enable_indexscan 값을 1로 설정하는 것이 좋습니다.	enable_indexscan 파라미터 값을 1로 설정하세요.	No	[PostgreSQL용 플래너 메서드 구성]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
innodb_flush_log_at_trx 파라미터가 비활성화되어 있습니다.	DB 인스턴스의 innodb_flush_log_at_trx 파라미터 값은 안전한 값이 아닙니다. 이 파라미터는 디스크에 대한 커밋 작업의 지속성을 제어합니다. innodb_flush_log_at_trx 파라미터를 1으로 설정하는 것이 좋습니다.	innodb_flush_log_at_trx 파라미터 값을 1로 설정하세요.	No	[로그 버퍼를 플러시하는 빈도 구성]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
innodb_stats_persistent 파라미터가 비활성화되어 있습니다.	DB 인스턴스가 InnoDB 통계를 디스크에 유지하도록 구성되지 않았습니다. 통계가 저장되지 않으면 인스턴스가 다시 시작되고 표에 액세스할 때마다 통계가 다시 계산됩니다. 이로 인해 쿼리 실행 계획이 달라질 수 있습니다. 테이블 수준에서 이 글로벌 파라미터의 값을 수정할 수 있습니다. innodb_stats_persistent 파라미터 값을 ON으로 설정하는 것이 좋습니다.	innodb_stats_persistent 파라미터 값을 ON로 설정하세요.	No	Amazon Aurora의 파라미터 그룹

유형	설명	권장 사항	가동 중지 필요	추가 정보
`innodb_open_files` 파라미터가 낮습니다.	`innodb_open_files` 파라미터는 InnoDB가 한 번에 열 수 있는 파일 수를 제어합니다. InnoDB는 mysqld가 실행될 때 모든 로그 및 시스템 테이블스페이스 파일을 엽니다. InnoDB가 한 번에 열 수 있는 최대 파일 수에 대한 DB 인스턴스 값이 낮습니다. `innodb_open_files` 파라미터 값을 65로 설정하는 것이 좋습니다.	`innodb_open_files` 파라미터를 최소값인 65로 설정합니다.	예	[InnoDB에서 MySQL용 파일 열기]()
`max_user_connections` 파라미터가 낮습니다.	DB 인스턴스의 각 데이터베이스 계정에 대한 최대 동시 연결 수 값이 낮습니다. `max_user_connections` 파라미터 설정을 5보다 큰 숫자로 늘리는 것이 좋습니다.	`max_user_connections` 파라미터 값을 5보다 큰 수로 늘리세요.	예	[MySQL용 계정 리소스 제한 설정]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
읽기 전용 복제본이 쓰기 가능 모드에서 열립니다.	DB 인스턴스에 쓰기 가능 모드의 읽기 전용 복제본이 있어 클라이언트의 업데이트를 허용합니다. 읽기 전용 복제본이 쓰기 가능 모드가 되지 않도록 `read_only` 파라미터를 `TrueIfReplica` 로 설정하는 것이 좋습니다.	`read_only` 파라미터 값을 `TrueIfReplica` 로 설정하세요.	No	[Amazon Aurora의 파라미터 그룹]()
`innodb_default_row_format` 파라미터 설정이 안전하지 않습니다.	DB 인스턴스에 8.0.26보다 낮은 MySQL 버전에서 `row_format` 이 `COMPACT` 또는 `REDUNDANT` 로 설정된 표에서 인덱스가 767바이트를 초과한 경우 액세스할 수 없고 복구할 수 없는 알려진 문제가 발생했습니다. `innodb_default_row_format` 파라미터 값을 `DYNAMIC`으로 설정하는 것이 좋습니다.	`innodb_default_row_format` 파라미터 값을 `DYNAMIC`로 설정하세요.	No	[MySQL 8.0.26의 변경 사항]()

유형	설명	권장 사항	가동 중지 필요	추가 정보
`general_logging` 파라미터가 활성화됨	DB 인스턴스에 대해 일반 로깅이 설정됩니다. 이 설정은 데이터베이스 문제를 해결하는 데 유용합니다. 그러나 일반 로깅을 활성화하면 I/O 작업량과 할당된 스토리지 공간이 늘어나 경합이 발생하고 성능이 저하될 수 있습니다. 일반 로깅 사용에 대한 요구 사항을 확인하세요. `general_logging` 파라미터 값을 `0`으로 설정하는 것이 좋습니다.	일반 로깅 사용에 대한 요구 사항을 확인하세요. 필수가 아닌 경우 `general_logging` 파라미터 값을 `0`으로 설정하는 것이 좋습니다.	No	Aurora MySQL 데이터베이스 로그 개요

유형	설명	권장 사항	가동 중지 필요	추가 정보
DB 클러스터가 읽기 워크로드를 위해 충분히 프로비저닝되지 않았습니다.	클러스터의 라이터 DB 인스턴스와 인스턴스 클래스 및 크기가 동일한 리더 DB 인스턴스를 DB 클러스터에 추가하는 것이 좋습니다. 현재 구성에는 주로 읽기 작업으로 인한 데이터베이스 부하가 지속적으로 높은 DB 인스턴스가 하나 있습니다. 클러스터에 다른 DB 인스턴스를 추가하고 읽기 워크로드를 DB 클러스터 읽기 전용 엔드포인트로 전달하여 이러한 작업을 분산하세요.	클러스터에 리더 DB 인스턴스를 추가합니다.	No	DB 클러스터에 Aurora 복제본 추가 Aurora DB 클러스터의 성능 및 확장 관리 Amazon RDS 요금
시스템 메모리 용량에 비해 충분히 프로비저닝되지 않은 RDS 인스턴스	더 적은 메모리를 사용하거나 할당된 메모리가 더 많은 DB 인스턴스 유형을 사용하도록 쿼리를 조정하는 것이 좋습니다. 인스턴스의 메모리가 부족하면 데이터베이스 성능이 영향을 받습니다.	메모리 용량이 더 큰 DB 인스턴스 사용	예	AWS 데이터베이스 블로그의 Scaling Your Amazon RDS Instance Vertically and Horizontally Amazon RDS 인스턴스 유형 Amazon RDS 요금

유형	설명	권장 사항	가동 중지 필요	추가 정보
시스템 CPU 용량에 비해 충분히 프로비저닝되지 않은 RDS 인스턴스	CPU 사용량을 줄이도록 쿼리를 조정하거나 DB 인스턴스를 수정하여 더 높은 vCPU가 할당된 DB 인스턴스 클래스를 사용하도록 DB 인스턴스를 수정하는 것이 좋습니다. DB 인스턴스의 CPU가 부족하면 데이터베이스 성능이 저하될 수 있습니다.	CPU 용량이 더 큰 DB 인스턴스 사용	예	AWS 데이터베이스 블로그의 Scaling Your Amazon RDS Instance Vertically and Horizontally Amazon RDS 인스턴스 유형 Amazon RDS 요금
RDS 리소스가 연결 풀링을 제대로 활용하지 못하는 경우	Amazon RDS 프록시를 활성화하여 기존 데이터베이스 연결을 효율적으로 풀링하고 공유하는 것이 좋습니다. 이미 데이터베이스용 프록시를 사용하고 있다면 프록시를 올바르게 구성하여 여러 DB 인스턴스 간의 연결 풀링과 로드 밸런싱을 개선합니다. RDS 프록시는 가용성과 확장성을 개선하는 동시에 연결 고갈 및 가동 중지 시간의 위험을 줄이는 데 유용할 수 있습니다.	RDS 프록시를 활성화하거나 기존 프록시 구성을 수정합니다.	No	AWS 데이터베이스 블로그의 Scaling Your Amazon RDS Instance Vertically and Horizontally Amazon RDS Proxy for Aurora Amazon RDS 프록시 요금

Amazon RDS 콘솔에서 지표 보기

Amazon RDS는 Amazon CloudWatch와 통합되어 RDS 콘솔의 Aurora DB 클러스터 지표의 다양한 기능을 표시합니다. 일부 지표는 클러스터 수준에서 적용되는 반면 다른 지표는 인스턴스 수준에서 적용됩니다. 인스턴스 수준 및 클러스터 수준 지표에 대한 설명은 Amazon Aurora용 지표 참조 섹션을 참조하세요.

Aurora DB 클러스터의 경우 다음의 지표 범주가 모니터링됩니다.

- CloudWatch - RDS 콘솔에서 액세스할 수 있는 Aurora 에 대한 Amazon CloudWatch 지표를 보여줍니다. CloudWatch 콘솔에서 이러한 지표에 액세스할 수 있습니다. 각 지표에는 특정 시간대에서 지표를 모니터링한 그래프도 포함되어 있습니다. 전체 CloudWatch 지표 목록은 Amazon Aurora에 대한 Amazon CloudWatch 지표 섹션을 참조하세요.
- 향상된 모니터링 - Aurora DB 클러스터가 향상된 모니터링을 활성화할 때 운영 체제 지표의 요약을 보여줍니다. RDS는 지표를 향상된 모니터링에서 Amazon CloudWatch Logs 계정으로 전달합니다. 각 지표에는 특정 시간대에서 지표를 모니터링한 그래프도 포함되어 있습니다. 개요는 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요. 향상된 모니터링 지표 목록은 향상된 모니터링의 OS 지표 섹션을 참조하세요.
- OS 프로세스 목록 – 클러스터에서 실행되는 각 프로세스의 세부 정보를 표시합니다.
- 성능 개선 도우미(Performance Insights) – Aurora DB 클러스터에서 DB 인스턴스에 대한 Amazon RDS 성능 개선 도우미 대시보드를 엽니다. 클러스터 수준에서는 성능 개선 도우미는 지원되지 않습니다. 성능 개선 도우미에 대한 개요는 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요. Lambda Insights 지표 목록은 Amazon RDS 성능 개선 도우미를 위한 Amazon CloudWatch 지표 섹션을 참조하세요.

Amazon RDS는 이제 성능 개선 도우미 대시보드에서 성능 개선 도우미 및 CloudWatch 지표에 대한 통합 보기를 제공합니다. 이 보기를 사용하려면 DB 클러스터에서 성능 개선 도우미가 켜져 있어야 합니다. 모니터링 탭에서 새 모니터링 보기를 선택하거나 탐색 창에서 성능 개선 도우미를 선택할 수 있습니다. 이 뷰를 선택하는 방법에 대한 지침을 보려면 성능 개선 도우미 대시보드와 결합된 지표 보기 섹션을 참조하세요.

성능 개선 도우미 대시보드와 결합된 지표 보기

⚠ Important

AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.

Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.

조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.

CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

Amazon RDS는 성능 개선 도우미 대시보드에서 DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표에 대한 통합 보기를 제공합니다. 사전 구성된 대시보드를 사용하거나 사용자 지정 대시보드를 만들 수 있습니다. 사전 구성된 대시보드는 데이터베이스 엔진의 성능 문제를 진단하는 데 도움이 되는 가장 일반적으로 사용되는 지표를 제공합니다. 또는 분석 요구 사항을 충족하는 데이터베이스 엔진 지표로 사용자 지정 대시보드를 만들 수 있습니다. 그런 다음 AWS 계정에 있는 해당 데이터베이스 엔진 유형의 모든 DB 인스턴스에 대해 이 대시보드를 사용하세요.

모니터링 탭에서 모니터링 보기를 선택하거나 탐색 창에서 성능 개선 도우미를 선택할 수 있습니다.

성능 개선 도우미 대시보드에서 결합 지표를 보려면 DB 클러스터에 대해 성능 개선 도우미를 켜야 합니다. 성능 개선 도우미 켜기에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 섹션을 참조하세요.

다음 섹션에서는 성능 개선 도우미 및 CloudWatch 지표를 표시하는 방법을 배워볼 수 있습니다.

주제

모니터링 탭에서 새 모니터링 보기 선택

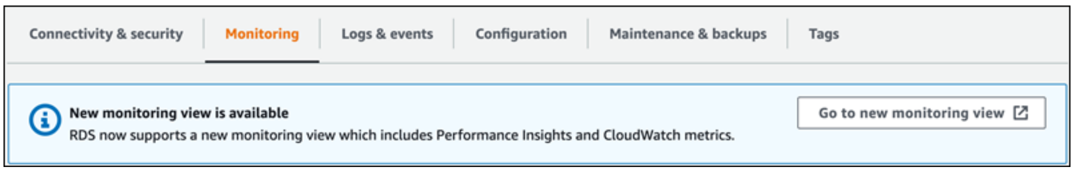
> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
>
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

Amazon RDS 콘솔에서 새 모니터링 보기를 선택하여 DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표를 볼 수 있습니다.

모니터링 탭에서 새 모니터링 보기를 선택하려면

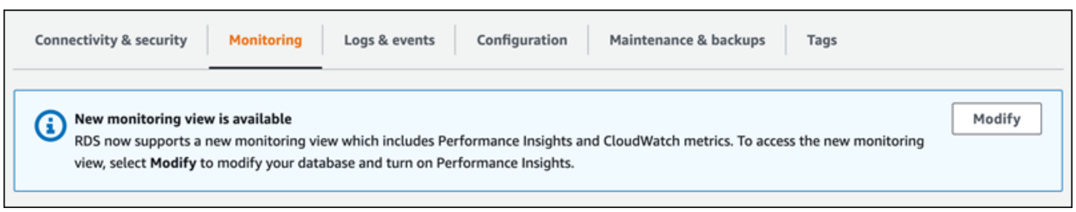
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 데이터베이스를 선택합니다.
3. 모니터링하려는 Aurora DB 클러스터를 선택합니다.
4. 아래로 스크롤하여 모니터링 탭을 선택합니다.

   새 모니터링 보기를 선택할 수 있는 옵션이 포함된 배너가 나타납니다. 다음 예제에서는 새 모니터링 보기를 선택하는 배너를 보여 줍니다.



5. 새 모니터링 보기로 이동을 선택하여 DB 클러스터DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표가 포함된 성능 개선 도우미 대시보드를 엽니다.
6. (선택 사항) DB 인스턴스에 대해 성능 개선 도우미가 꺼져 있는 경우 DB 인스턴스를 수정하고 성능 개선 도우미를 켤 수 있는 옵션이 포함된 배너가 나타납니다.

   다음 예제는 모니터링 탭의 DB 인스턴스를 수정하기 위한 배너를 보여 줍니다.



   수정을 선택하여 DB 인스턴스를 수정하고 성능 개선 도우미를 켭니다. 성능 개선 도우미 켜기에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 섹션을 참조하세요.

성능 개선 도우미 페이지에서 새 모니터링 보기 선택

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니

다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.

Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.

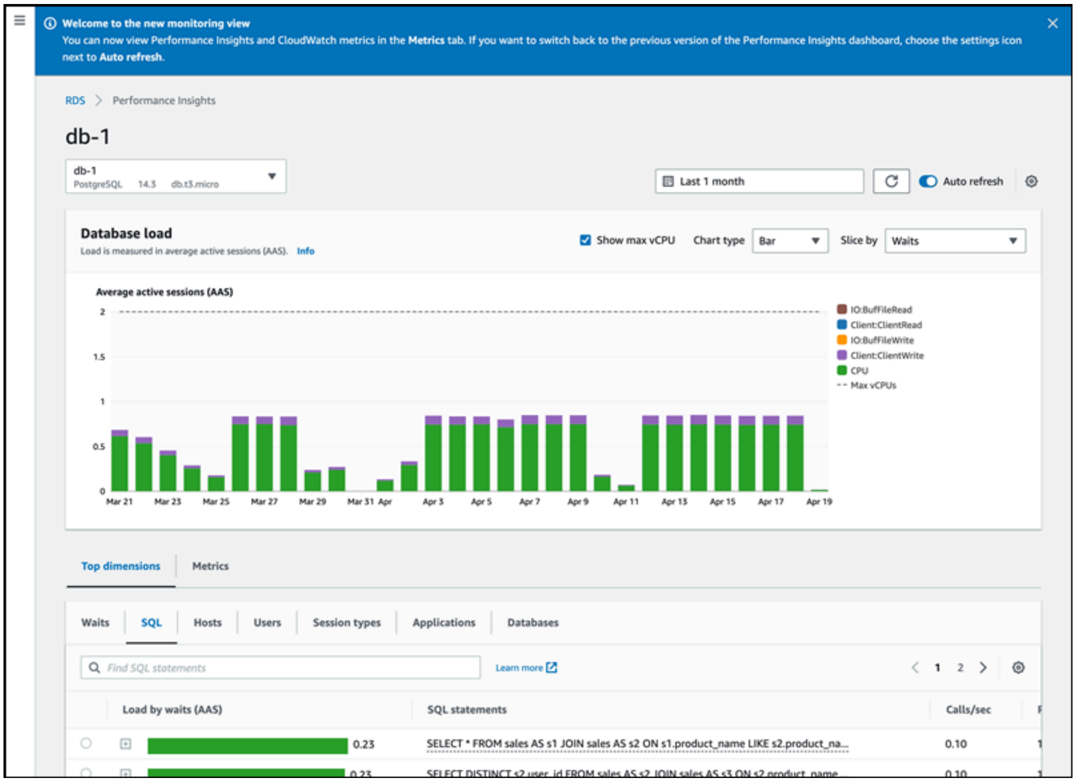
조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.

CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

Amazon RDS 콘솔에서 새 모니터링 보기를 선택하여 DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표를 볼 수 있습니다.

탐색 창의 성능 개선 도우미를 사용하여 새 모니터링 보기를 선택하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택하면 DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표가 모두 표시되는 성능 개선 도우미 대시보드를 볼 수 있습니다.



성능 개선 도우미를 사용하여 사용자 지정 대시보드 생성

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora에 Database Insights의 고급 모드 켜기](#) 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.

CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

새로운 모니터링 보기에서 분석 요구 사항을 충족하는 데 필요한 지표가 포함된 사용자 지정 대시보드를 만들 수 있습니다.

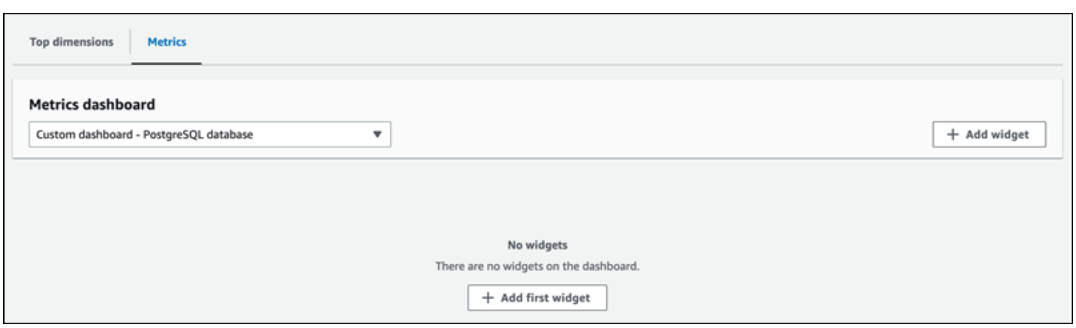
DB 인스턴스에 대한 성능 개선 도우미 및 CloudWatch 지표를 선택하여 사용자 지정 대시보드를 만들 수 있습니다. AWS 계정에 있는 동일한 데이터베이스 엔진 유형의 다른 DB 인스턴스에 대해 이 대시보드를 사용하세요.

> **Note**
>
> 사용자 지정 대시보드는 최대 50개의 지표를 지원합니다.

위젯 설정 메뉴를 사용하여 대시보드를 편집 또는 삭제하고 위젯 창을 이동하거나 규모를 조정합니다.

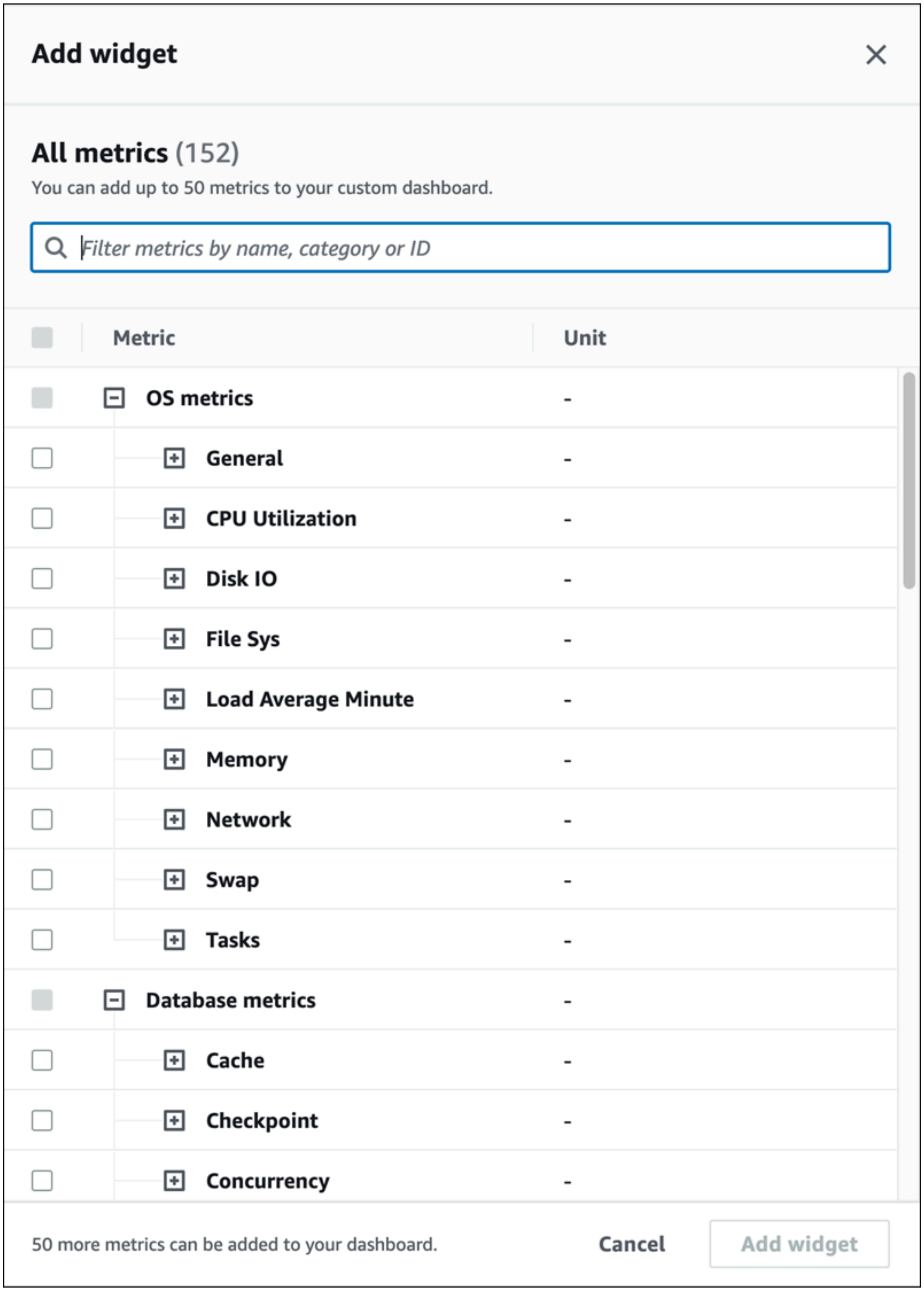
탐색 창의 성능 개선 도우미를 사용하여 사용자 지정 대시보드를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 창의 지표 탭으로 스크롤을 내립니다.
5. 드롭다운 목록에서 사용자 지정 대시보드를 선택합니다. 다음 예제에서는 사용자 지정 대시보드 생성을 보여 줍니다.



6. 위젯 추가를 선택하여 위젯 추가 창을 엽니다. 창에서 사용 가능한 운영 체제(OS) 지표, 데이터베이스 지표 및 CloudWatch 지표를 열고 볼 수 있습니다.

   다음 예제는 지표가 포함된 위젯 추가 창을 보여 줍니다.



7. 대시보드에서 보려는 지표를 선택하고 위젯 추가를 선택합니다. 검색 필드를 사용하여 특정 지표를 찾을 수 있습니다.

   선택한 지표가 대시보드에 표시됩니다.

8. (선택 사항) 대시보드를 수정하거나 삭제하려면 위젯의 오른쪽 상단에 있는 설정 아이콘을 선택한 다음 메뉴에서 다음 작업 중 하나를 선택합니다.

   - 편집 – 창에서 지표 목록을 수정합니다. 대시보드의 지표를 선택한 후 위젯 업데이트를 선택합니다.
   - 삭제 – 위젯을 삭제합니다. 확인 창에서 삭제를 선택합니다.

성능 개선 도우미를 사용하여 사전 구성된 대시보드 선택

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

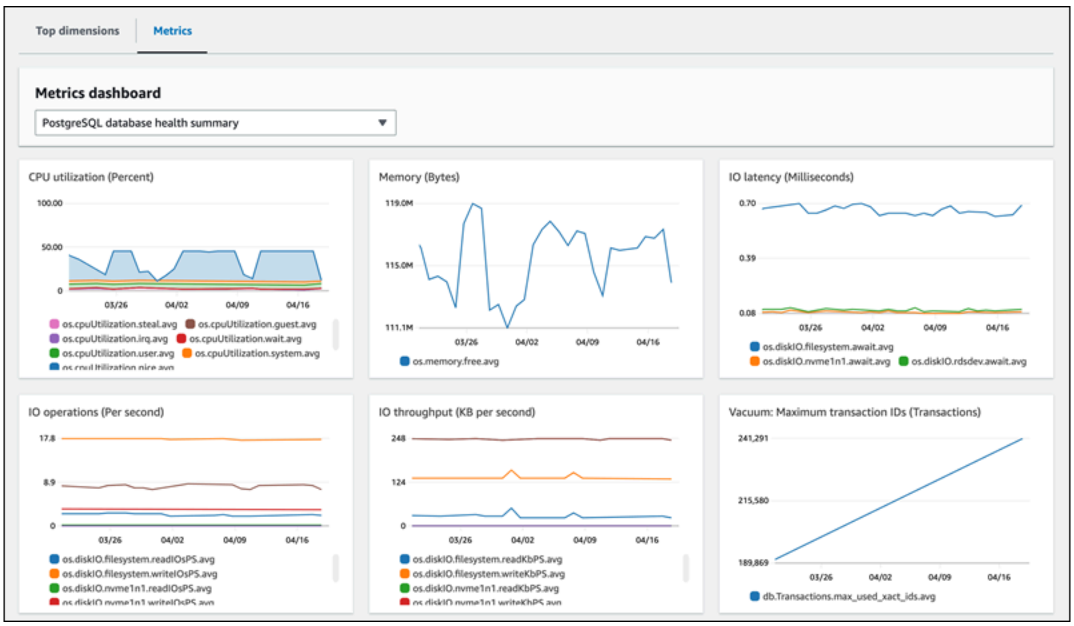
사전 구성된 대시보드에서 자주 사용하는 지표를 볼 수 있습니다. 이 대시보드는 데이터베이스 엔진의 성능 문제를 진단하고 평균 복구 시간을 몇 시간에서 몇 분으로 줄이는 데 도움이 됩니다.

> **Note**
>
> 이 대시보드는 편집할 수 없습니다.

탐색 창의 성능 개선 도우미를 사용하여 사전 구성된 대시보드를 선택하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 창의 지표 탭으로 스크롤을 내립니다.
5. 드롭다운 목록에서 사전 구성된 대시보드를 선택합니다.

   대시보드에서 DB 인스턴스에 대한 지표를 볼 수 있습니다. 다음 예제에서는 사전 구성된 지표 대시보드를 보여 줍니다.



Amazon CloudWatch로 Amazon Aurora 지표 모니터링

Amazon CloudWatch는 지표 리포지토리입니다. 리포지토리는 Amazon Aurora에서 원시 데이터를 수집한 후 판독이 가능한 지표로 실시간에 가깝게 처리합니다. CloudWatch로 전송되는 Amazon Aurora 지표의 전체 목록은 [Amazon Aurora용 지표 참조](#)를 참조하세요.

데이터베이스의 성능을 대규모로 분석하고 문제를 해결하려면 CloudWatch Database Insights를 사용합니다.

주제

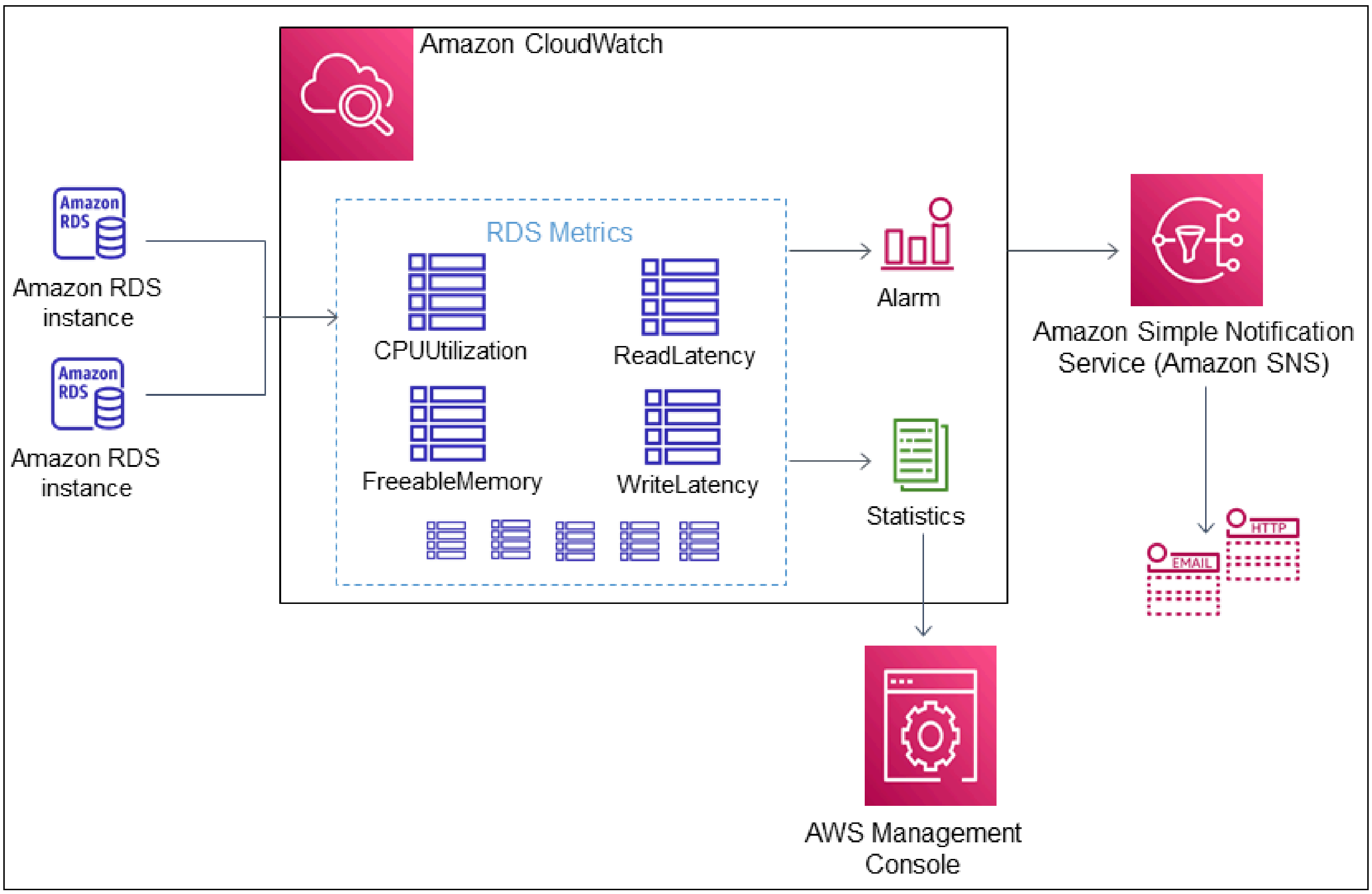
- Amazon Aurora 및 Amazon CloudWatch 개요
- CloudWatch 콘솔 및 AWS CLI에서 DB 클러스터 지표 보기
- CloudWatch에 성능 개선 도우미 지표 내보내기
- Amazon Aurora을 모니터링하는 CloudWatch 경보 생성

Amazon Aurora 및 Amazon CloudWatch 개요

Amazon Aurora는 기본적으로 1분 단위로 CloudWatch에 지표 데이터를 자동 전송합니다. 예를 들어 `CPUUtilization` 지표는 시간 경과에 따른 DB 인스턴스의 CPU 사용률을 기록합니다. 기간이 60초(1분)로 설정된 데이터 요소들은 15일 동안 사용할 수 있습니다. 즉, 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 살펴볼 수 있습니다.

이제 성능 개선 도우미 지표 대시보드를 Amazon RDS에서 Amazon CloudWatch로 내보낼 수 있습니다. 사전 구성된 지표 대시보드 또는 사용자 지정 지표 대시보드를 새 대시보드로 내보내거나 기존 CloudWatch 대시보드에 추가할 수 있습니다. 내보낸 대시보드를 CloudWatch 콘솔에서 볼 수 있습니다. 성능 개선 도우미 지표 대시보드를 CloudWatch로 내보내는 방법에 대한 자세한 내용은 [CloudWatch에 성능 개선 도우미 지표 내보내기](#) 섹션을 참조하세요.

다음 다이어그램에서 볼 수 있듯이 CloudWatch 지표에 대해 경보를 설정할 수 있습니다. 예를 들어 인스턴스의 CPU 사용률이 70% 이상일 때 알리는 경보를 생성할 수 있습니다. 임계값을 초과하면 이메일을 보내도록 Amazon Simple Notification Service를 구성할 수 있습니다.



Amazon RDS는 다음과 같은 유형의 지표를 Amazon CloudWatch에 게시합니다.

- 클러스터 수준과 인스턴스 수준에서의 Aurora 지표

  이들 지표에 대한 표는 Amazon Aurora에 대한 Amazon CloudWatch 지표 단원을 참조하세요.

- 성능 개선 도우미 지표

  이들 지표에 대한 표는 Amazon RDS 성능 개선 도우미를 위한 Amazon CloudWatch 지표 및 성능 개선 도우미 카운터 단원을 참조하세요.

- 향상된 모니터링 지표(Amazon CloudWatch Logs에 게시됨)

  이들 지표에 대한 표는 향상된 모니터링의 OS 지표 단원을 참조하세요.

- 사용자 AWS 계정의 Amazon RDS 서비스 할당량에 대한 사용량 지표

  이들 지표에 대한 표는 Amazon Aurora에 대한 Amazon CloudWatch 사용량 지표 단원을 참조하세요. Amazon RDS 할당량에 대한 자세한 내용은 Amazon Aurora에 대한 할당량 및 제약 조건 단원을 참조하세요.

CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서에서 Amazon CloudWatch란 무엇입니까?를 참조하십시오. CloudWatch 지표 보존 기간에 대한 자세한 내용은 지표 보존 기간을 참조하세요.

CloudWatch 콘솔 및 AWS CLI에서 DB 클러스터 지표 보기

다음은 CloudWatch를 사용하여 DB 인스턴스에 대한 지표를 보는 방법에 대한 세부 정보입니다. CloudWatch Logs을 사용하여 DB 인스턴스의 운영 체제에 대한 지표를 실시간으로 모니터링하는 방법에 대한 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 단원을 참조하십시오.

Amazon Aurora 리소스를 사용할 때 Amazon Aurora는 1분마다 지표와 측정기준을 Amazon CloudWatch로 전송합니다.

이제 Amazon RDS에서 Amazon CloudWatch로 성능 개선 도우미 지표 대시보드를 내보내고 CloudWatch 콘솔에서 해당 지표를 볼 수 있습니다. 성능 개선 도우미 지표 대시보드를 CloudWatch로 내보내는 방법에 대한 자세한 내용은 CloudWatch에 성능 개선 도우미 지표 내보내기 섹션을 참조하세요.

다음 절차에 따라 CloudWatch 콘솔 및 CLI에서 Amazon Aurora에 대한 지표를 확인합니다.

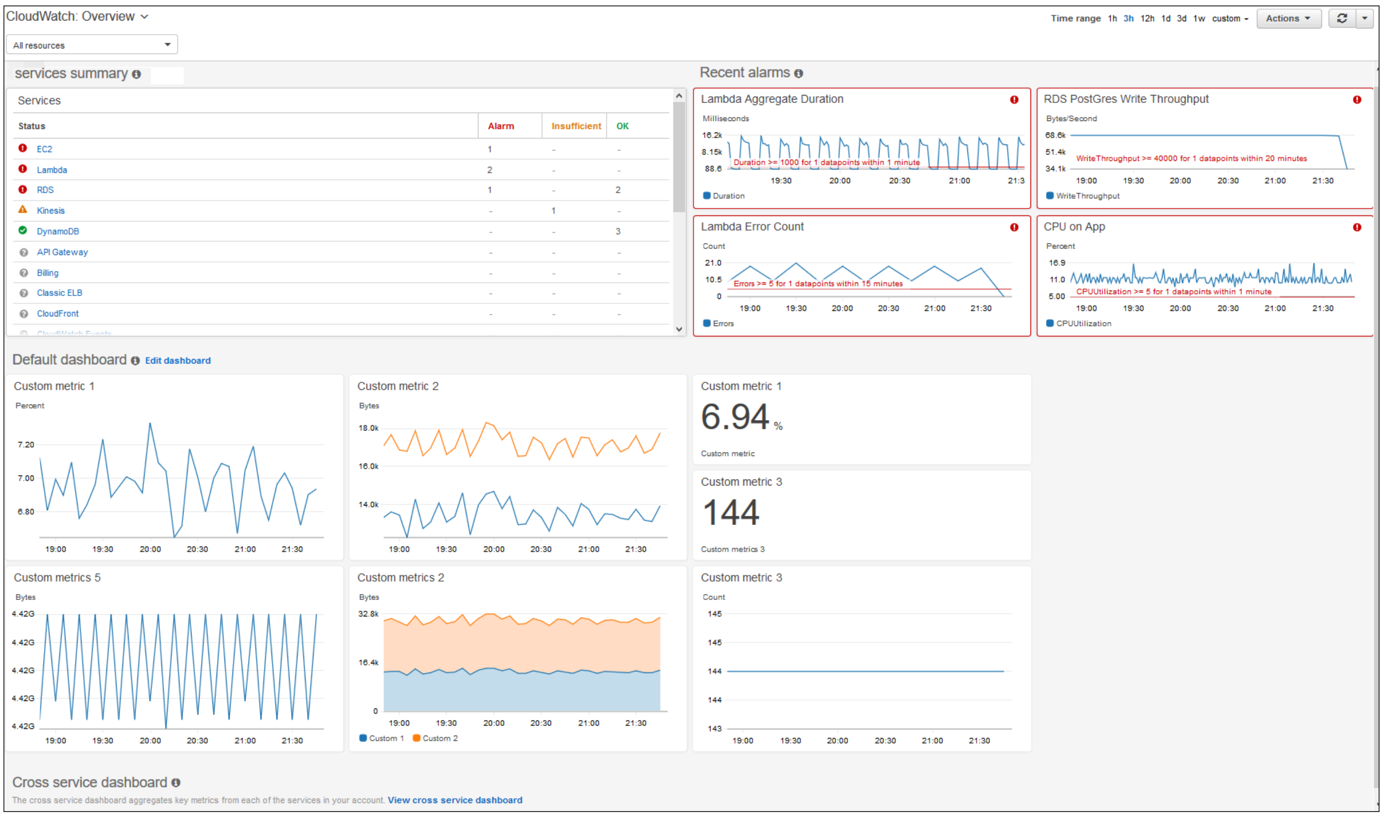
콘솔

Amazon CloudWatch 콘솔을 사용한 메트릭 확인

측정치는 먼저 서비스 네임스페이스별로 그룹화된 다음, 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

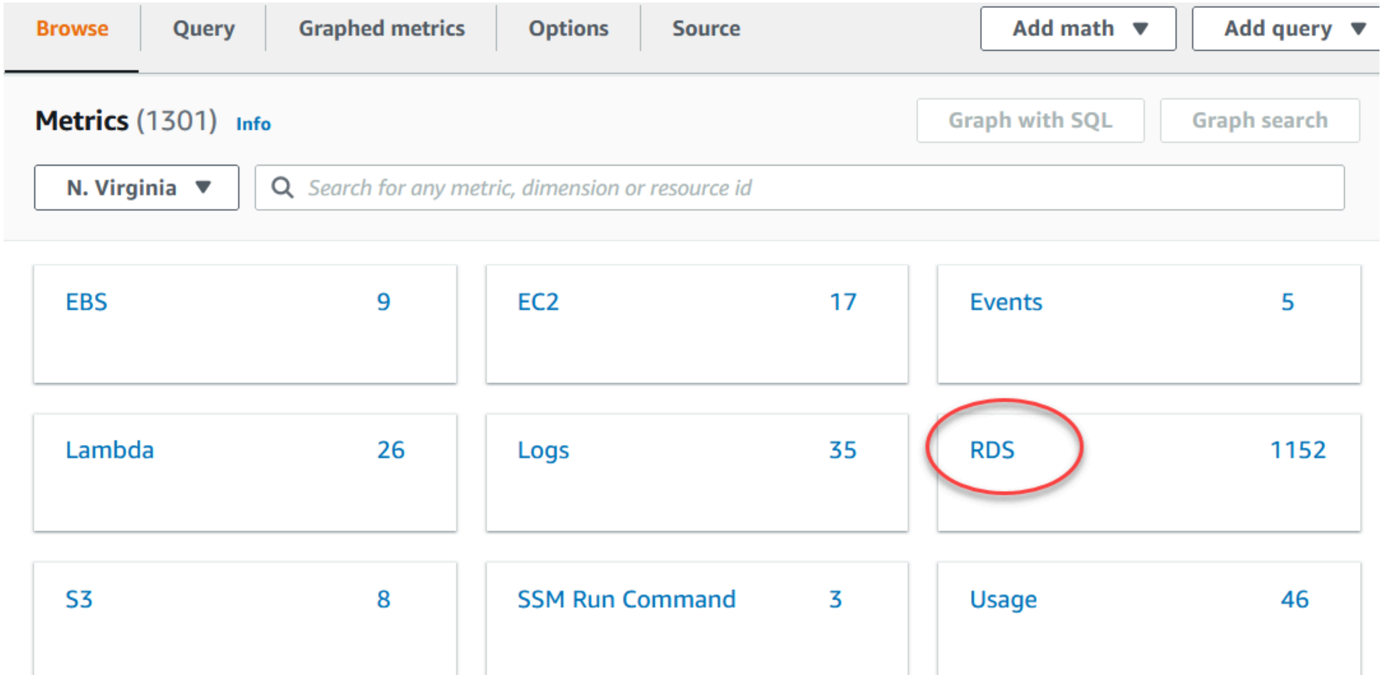
1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

   CloudWatch 개요 홈페이지가 표시됩니다.



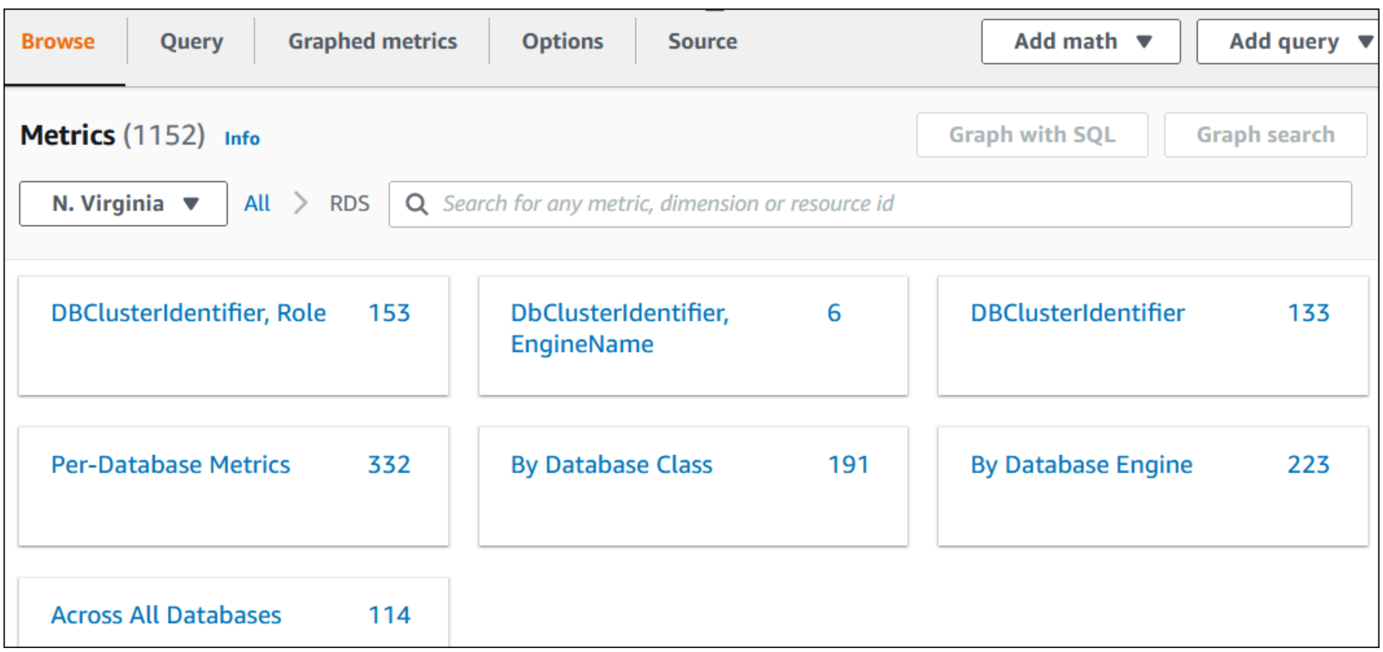
2. 필요한 경우 AWS 리전을 변경합니다. 탐색 모음에서 AWS 리전 리소스가 있는 AWS을 선택합니다. 자세한 내용은 [리전 및 엔드포인트](#) 섹션을 참조하십시오.

3. 탐색 창에서 지표(Metrics)를 선택한 다음 모든 지표(All metrics)를 선택합니다.

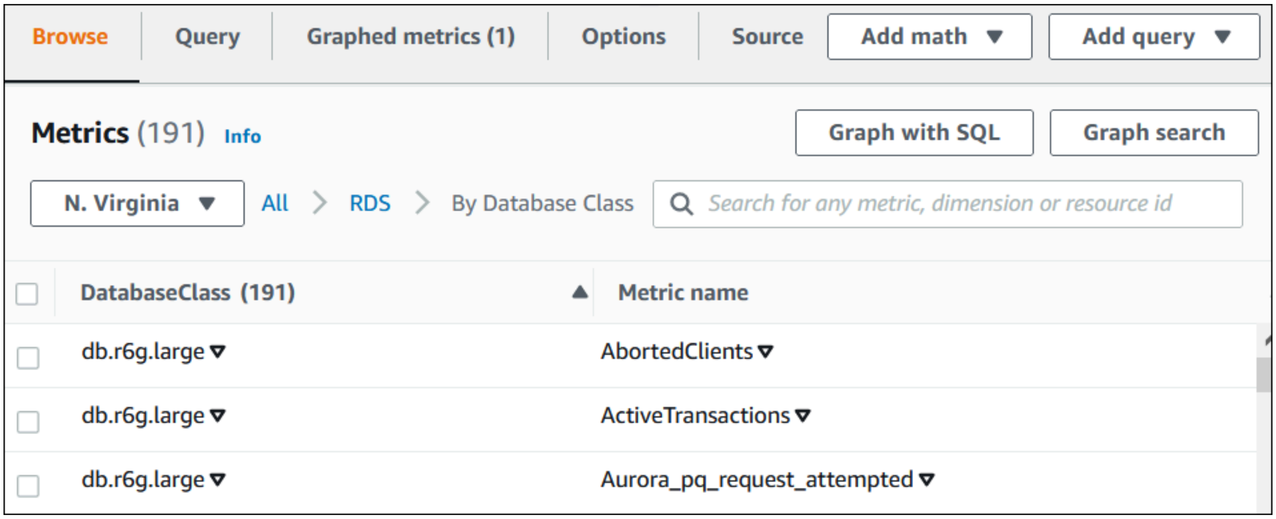


4. 아래로 스크롤하여 RDS 지표 네임스페이스를 선택합니다.

   페이지에 Amazon Aurora 측정기준이 표시됩니다. 이러한 측정기준에 대한 설명은 Aurora에 대한 Amazon CloudWatch 측정기준 목록 섹션을 참조하세요.



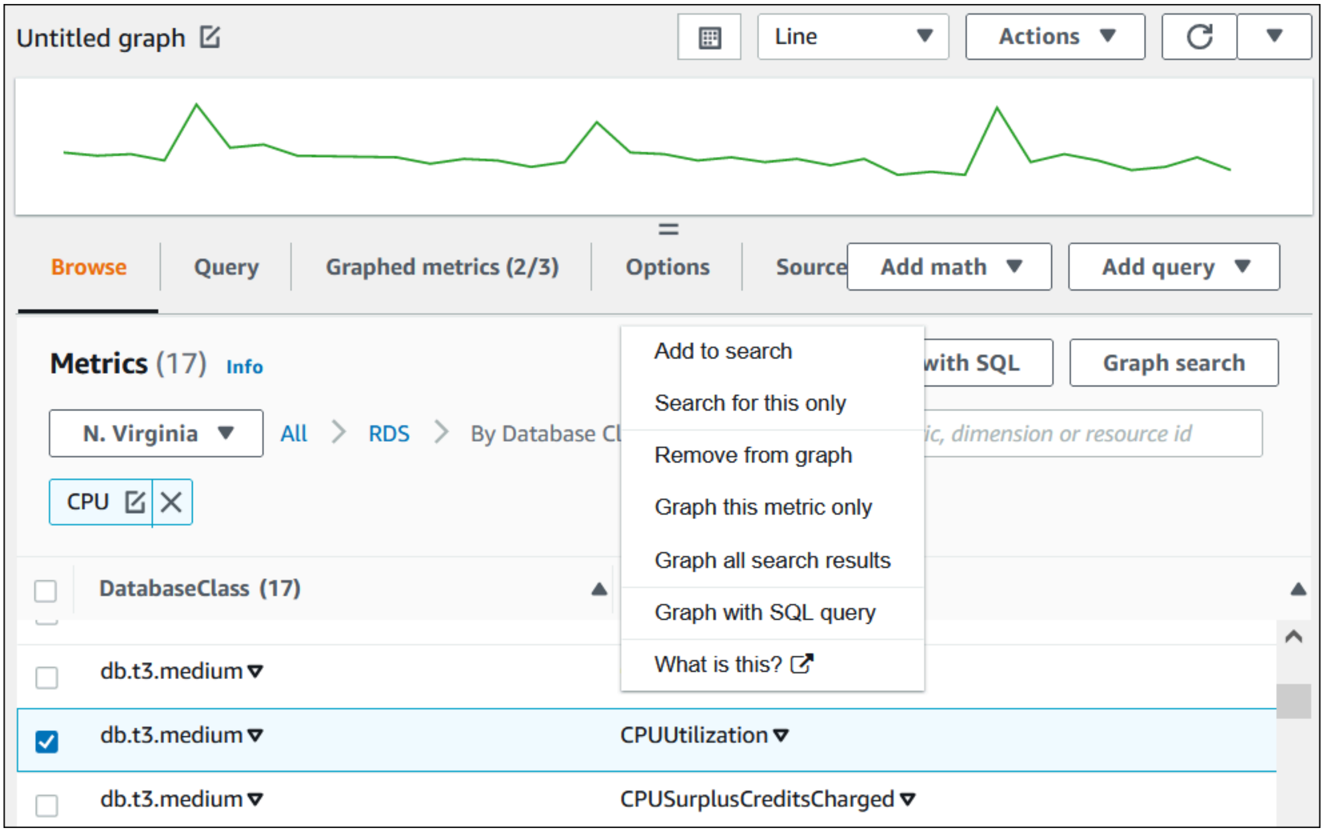
5. 측정치 차원(예: 데이터베이스 클래스별)을 선택합니다.



6. 다음 작업을 수행합니다.

   - 지표를 정렬하려면 열 머리글을 사용합니다.
   - 측정치를 그래프로 표시하려면 측정치 옆에 있는 확인란을 선택합니다.
   - 리소스로 필터링하려면 리소스 ID를 선택한 후 검색에 추가를 선택합니다.
   - 지표로 필터링하려면 지표 이름을 선택한 후 검색에 추가를 선택합니다.

   다음 예제는 db.t3.medium 클래스를 필터링하고 CPUUtilization 지표를 그래프로 나타냅니다.



CloudWatch 지표를 사용하여 Aurora PostgreSQL의 리소스 사용량을 분석하는 방법에 대한 세부 정보를 찾을 수 있습니다. 자세한 내용은 [Amazon CloudWatch 지표를 사용한 Aurora PostgreSQL의 리소스 사용량 분석](#) 단원을 참조하세요.

AWS CLI

AWS CLI를 사용하여 지표 정보를 얻으려면 CloudWatch 명령 `list-metrics`를 사용합니다. 다음 예에서는 `AWS/RDS` 네임스페이스의 모든 지표를 나열합니다.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

지표 데이터를 얻으려면 `get-metric-data` 명령을 사용합니다.

다음 예제는 5분 단위로 특정 24시간 동안의 `my-instance` 인스턴스에 대한 `CPUUtilization` 통계를 가져옵니다.

다음 콘텐츠를 포함하는 `CPU_metric.json` JSON 파일을 생성합니다.

```
{
```

```
    "StartTime" : "2023-12-25T00:00:00Z",
    "EndTime" : "2023-12-26T00:00:00Z",
    "MetricDataQueries" : [{
      "Id" : "cpu",
      "MetricStat" : {
     "Metric" : {
         "Namespace" : "AWS/RDS",
         "MetricName" : "CPUUtilization",
         "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
     },
        "Period" : 360,
        "Stat" : "Minimum"
      }
    }]
}
```

Example

대상 LinuxmacOS, 또는Unix:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

Windows의 경우:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

샘플 출력은 다음과 같이 나타납니다.

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
```

```
            "Values": [
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

자세한 내용은 Amazon CloudWatch 사용 설명서의 [지표에 대한 통계 얻기]를 참조하세요.

CloudWatch에 성능 개선 도우미 지표 내보내기

성능 개선 도우미를 사용하면 DB 인스턴스에 대한 사전 구성된 지표 또는 사용자 지정 지표 대시보드를 Amazon CloudWatch로 내보낼 수 있습니다. 지표 대시보드를 새 대시보드로 내보내거나 기존 CloudWatch 대시보드에 추가할 수 있습니다. 기존 CloudWatch 대시보드에 대시보드를 추가하는 경우 지표가 CloudWatch 대시보드의 별도 섹션에 표시되도록 헤더 레이블을 만들 수 있습니다.

내보낸 지표 대시보드를 CloudWatch 콘솔에서 볼 수 있습니다. 성능 개선 도우미 지표 대시보드를 내보낸 후 새 지표를 추가하는 경우 이 대시보드를 다시 내보내야 CloudWatch 콘솔에서 새 지표를 볼 수 있습니다.

성능 개선 도우미 대시보드에서 지표 위젯을 선택하고 CloudWatch 콘솔에서 지표 데이터를 볼 수도 있습니다.

CloudWatch 콘솔에서 지표를 보는 방법에 대한 자세한 내용은 CloudWatch 콘솔 및 AWS CLI에서 DB 클러스터 지표 보기 섹션을 참조하세요.

다음 섹션에서는 성능 개선 도우미 지표를 CloudWatch에 신규 또는 기존 대시보드로 내보내고 CloudWatch에서 성능 개선 도우미 지표를 확인합니다.

주제

- 성능 개선 도우미 지표를 CloudWatch에 새 대시보드로 내보내기
- 기존 CloudWatch 대시보드에 성능 개선 도우미 지표 추가
- CloudWatch에서 성능 개선 도우미 지표 위젯 보기

성능 개선 도우미 지표를 CloudWatch에 새 대시보드로 내보내기

성능 개선 도우미 대시보드에서 사전 구성된 지표 대시보드 또는 사용자 지정 지표 대시보드를 선택하고 CloudWatch에 새 대시보드로 내보냅니다. 내보낸 대시보드를 CloudWatch 콘솔에서 볼 수 있습니다.

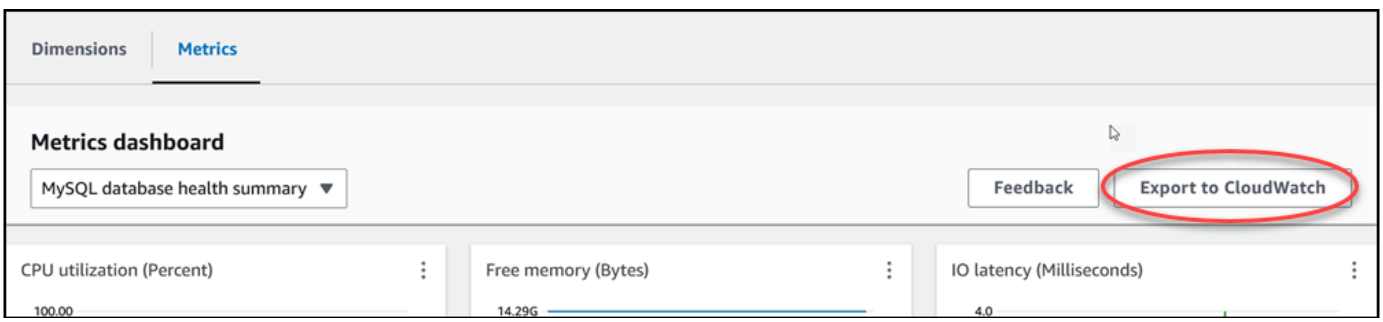
성능 개선 도우미 지표 대시보드를 CloudWatch에 새 대시보드로 내보내는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

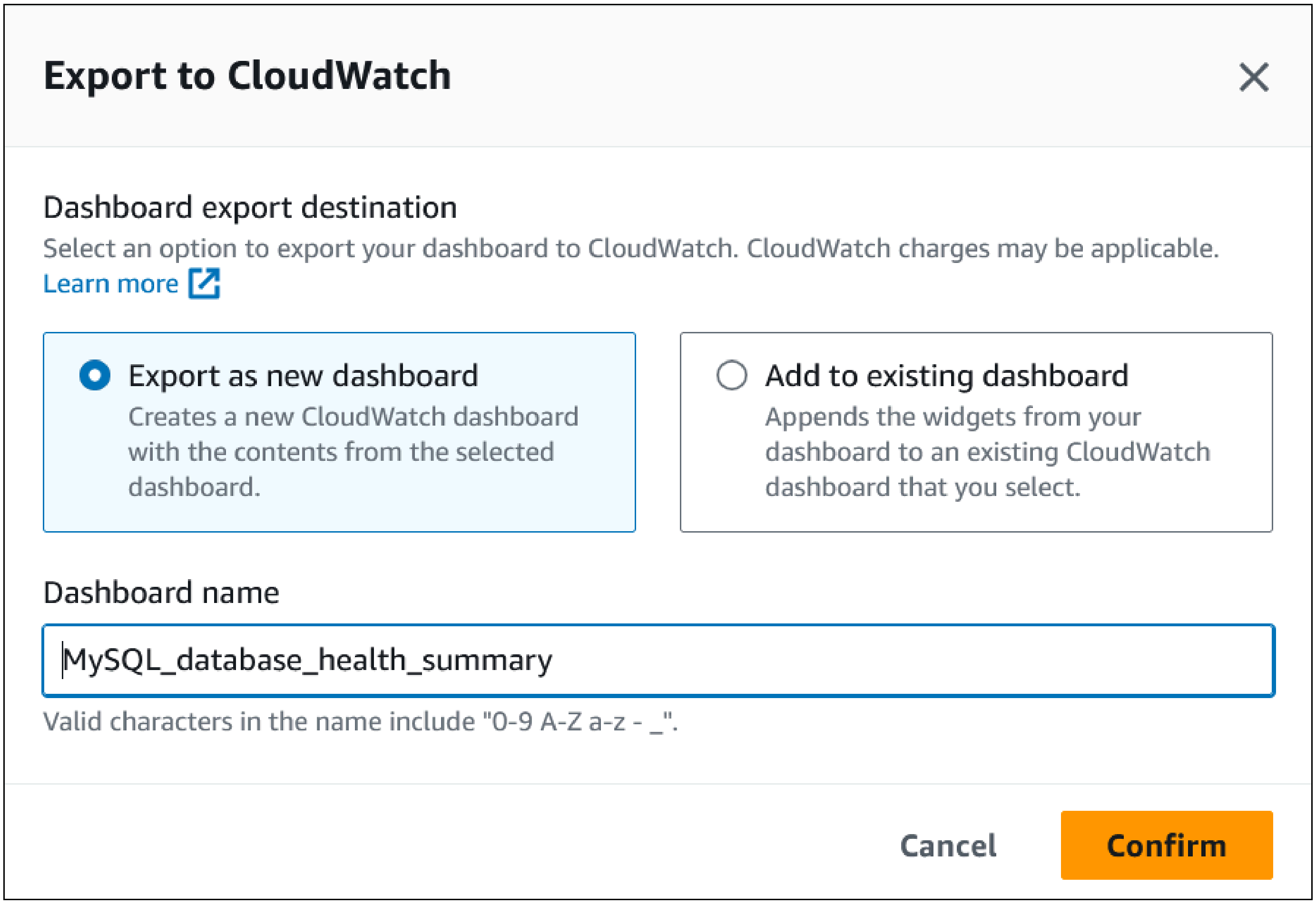
   해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 아래로 스크롤하여 지표를 선택합니다.

   기본적으로 성능 개선 도우미 지표가 포함된 사전 구성된 대시보드가 표시됩니다.
5. 사전 구성된 대시보드 또는 사용자 지정 대시보드를 선택한 다음 CloudWatch로 내보내기를 선택합니다.

   CloudWatch로 내보내기 창이 나타납니다.

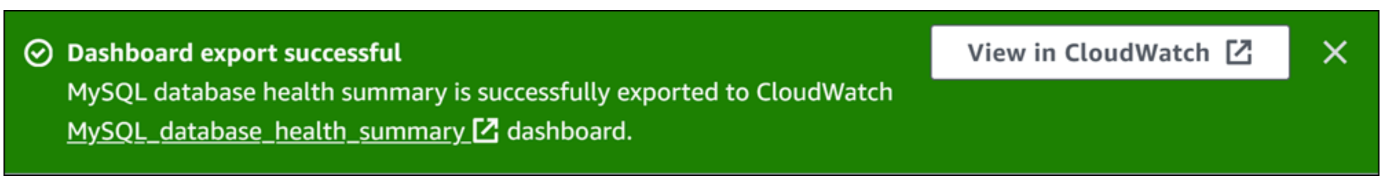
   

6. 새 대시보드로 내보내기를 선택합니다.



7. 대시보드 이름 필드에 새 대시보드의 이름을 입력하고 확인을 선택합니다.

   대시보드 내보내기에 성공하면 배너에 메시지가 표시됩니다.



8. CloudWatch 콘솔에서 지표 대시보드를 보려면 배너에서 링크 또는 CloudWatch에서 보기를 선택합니다.

기존 CloudWatch 대시보드에 성능 개선 도우미 지표 추가

사전 구성된 지표 대시보드 또는 사용자 지정 지표 대시보드를 기존 CloudWatch 대시보드에 추가합니다. CloudWatch 대시보드의 별도 섹션에 표시되도록 지표 대시보드에 레이블을 추가할 수 있습니다.

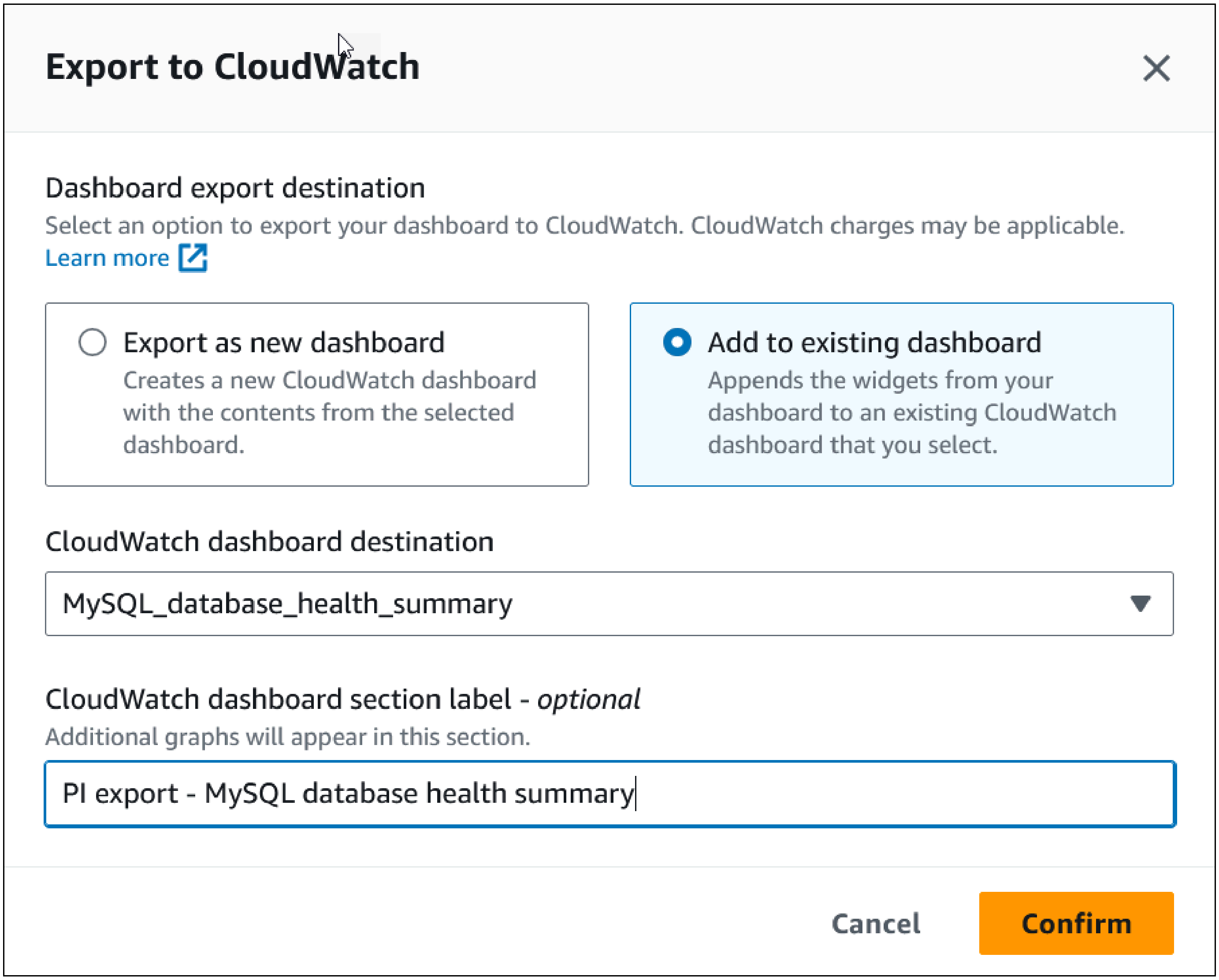
지표를 기존 CloudWatch 대시보드로 내보내는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

   해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 아래로 스크롤하여 지표를 선택합니다.

   기본적으로 성능 개선 도우미 지표가 포함된 사전 구성된 대시보드가 표시됩니다.
5. 사전 구성된 대시보드 또는 사용자 지정 대시보드를 선택한 다음 CloudWatch로 내보내기를 선택합니다.

   CloudWatch로 내보내기 창이 나타납니다.
6. 기존 대시보드에 추가를 선택합니다.



7. 대시보드 대상 및 레이블을 지정한 다음 확인을 선택합니다.

   • CloudWatch 대시보드 대상 - 기존 클라우드워치 대시보드를 선택합니다.
   • CloudWatch 대시보드 섹션 레이블 - 선택 사항 - CloudWatch 대시보드의 이 섹션에 표시할 성능 개선 도우미 지표의 이름을 입력합니다.

   대시보드 내보내기에 성공하면 배너에 메시지가 표시됩니다.

8. CloudWatch 콘솔에서 지표 대시보드를 보려면 배너에서 링크 또는 CloudWatch에서 보기를 선택합니다.

CloudWatch에서 성능 개선 도우미 지표 위젯 보기

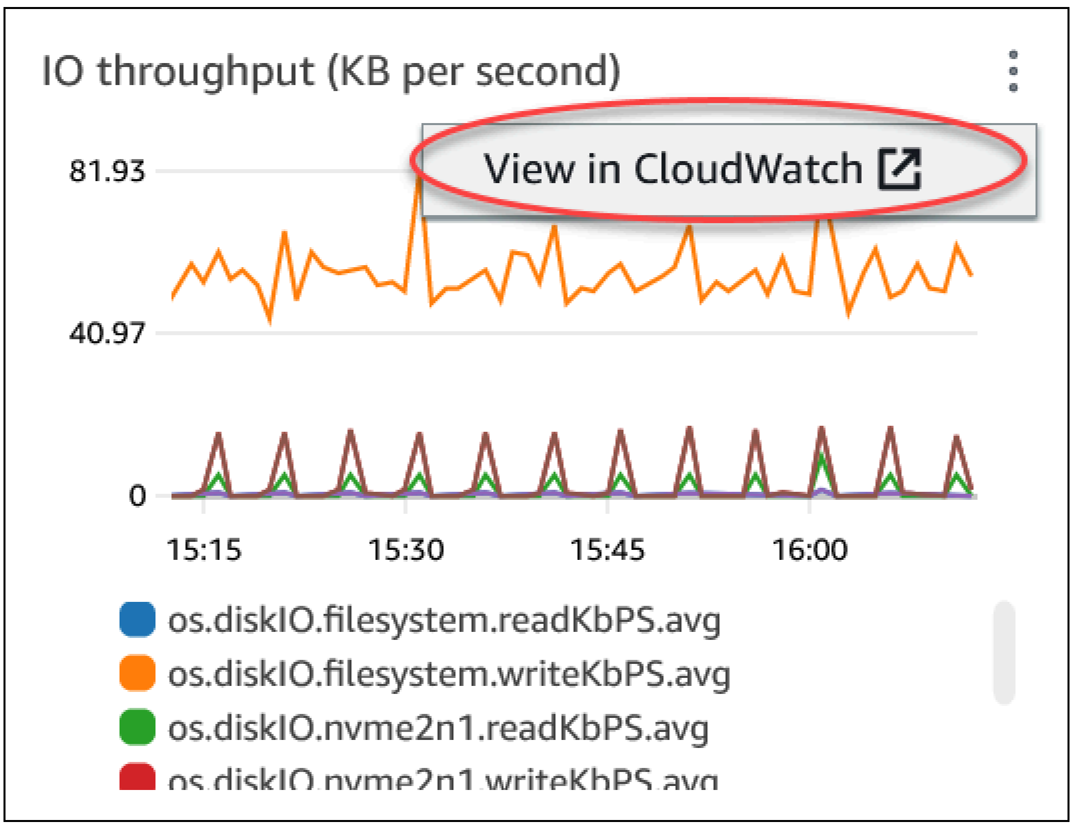
Amazon RDS 성능 개선 도우미 대시보드에서 성능 개선 도우미 지표 위젯을 선택하고 CloudWatch 콘솔에서 지표 데이터를 확인합니다.

지표 위젯을 내보내고 CloudWatch 콘솔에서 지표 데이터를 확인하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

   해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 아래로 스크롤하여 지표로 이동합니다.

   기본적으로 성능 개선 도우미 지표가 포함된 사전 구성된 대시보드가 표시됩니다.
5. 지표 위젯을 선택한 다음 메뉴에서 CloudWatch에서 보기를 선택합니다.



   지표 데이터가 CloudWatch 콘솔에 표시됩니다.

Amazon Aurora을 모니터링하는 CloudWatch 경보 생성

경보로 인해 상태가 변경되면 Amazon SNS 메시지를 보내는 CloudWatch 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 또한 경보는 여러 기간에 대해 주어진 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행할 수 있습니다. 이 작업은 Amazon SNS 주제나 Amazon EC2 Auto Scaling 정책으로 전송되는 알림입니다.

경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 특정 상태에 있다고 해서 작업을 호출하지는 않습니다. 상태가 변경되어 지정된 기간 동안 유지되어야 합니다.

> **Note**
>
> Aurora의 경우 특정 DB 인스턴스의 지표에 의존하지 않고 `WRITER` 또는 `READER` 역할 지표를 사용하여 경보를 설정합니다. Aurora DB 인스턴스 역할은 시간이 지나면서 역할이 변화할 수 있습니다. CloudWatch 콘솔에서 이러한 역할 기반 지표를 찾을 수 있습니다.
> Aurora Auto Scaling은 `READER` 역할 지표를 기반으로 경보를 자동으로 설정합니다. Aurora Auto Scaling에 대한 자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 단원을 참조하십시오.

CloudWatch 콘솔의 DB_PERF_INSIGHTS 지표 수학 함수를 사용하여 Amazon RDS에서 성능 개선 도우미 카운터 지표를 쿼리할 수 있습니다. DB_PERF_INSIGHTS 함수에는 1분 미만의 간격으로 DBLoad 지표도 포함됩니다. 이러한 지표에 대해 CloudWatch 경보를 설정할 수 있습니다.

경보를 만드는 방법에 대한 자세한 내용은 AWS 데이터베이스의 성능 개선 도우미 카운터 지표에 대한 경보 생성을 참조하세요.

AWS CLI를 사용하여 경보 설정

- `put-metric-alarm`을 호출합니다. 자세한 내용은 AWS CLI 명령 참조를 참조하세요.

CloudWatch API를 사용하여 경보 설정

- 을 호출합니다..`PutMetricAlarm` 자세한 내용은 Amazon CloudWatch API 참조를 참조하세요.

Amazon SNS 주제 및 경보 설정에 대한 자세한 내용은 Amazon CloudWatch 경보 사용을 참조하세요.

CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링

Database Insights를 사용하여 Amazon Aurora DB 인스턴스 플릿의 데이터베이스 로드(DB 로드)를 모니터링합니다. DB 로드는 데이터베이스의 세션 활동 수준을 측정합니다. Database Insights를 사용하여 Amazon Aurora 데이터베이스의 성능을 대규모로 분석하고 문제를 해결할 수 있습니다.

Database Insights를 사용하면 데이터베이스 플릿의 DB 로드를 시각화하고 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 로드를 필터링할 수 있습니다.

기본적으로 RDS는 Amazon Aurora 데이터베이스에 대해 Database Insights의 표준 모드를 사용합니다. DB 클러스터에 대한 Database Insights의 고급 모드를 켜면 RDS는 클러스터의 모든 DB 인스턴스에 대해 Database Insights를 사용합니다.

Amazon CloudWatch 콘솔에서 Database Insights를 사용하는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch Database Insights를 참조하세요.

요금

요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하세요.

주제

Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스에서 Database Insights 지원

다음 테이블에는 Database Insights를 지원하는 Amazon Aurora DB 엔진이 나와 있습니다.

Amazon Aurora DB 엔진	지원되는 엔진 버전 및 리전	인스턴스 클래스 제한 사항
Amazon Aurora MySQL 호환 버전	Aurora MySQL에서 사용하는 Database Insights의 버전 및 리전 가용성에 대한 자세한 내용은 Aurora MySQL을 사용하는 성능 개선 도우미 섹션을 참조하세요.	Database Insights에는 다음과 같은 엔진 클래스 제한 사항이 있습니다. • db.t2 - 지원되지 않음 • db.t3 - 지원되지 않음 • db.t4g.micro 및 db.t4g.small – 지원되지 않음
Amazon Aurora PostgreSQL 호환 에디션	Aurora PostgreSQL에서 사용하는 Database Insights의 버전 및 지역 가용성에 대한 자세한 내용은 Aurora PostgreSQL을 사용하는 성능 개선 도우미 섹션을 참조하세요.	해당 사항 없음
Aurora PostgreSQL Limitless Database	Aurora PostgreSQL Limitless Database와 함께 Database Insights를 사용하는 것에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Aurora PostgreSQL Limitless Database 모니터링 섹션을 참조하세요.	해당 사항 없음

Database Insights가 Amazon Aurora Serverless v2를 지원합니다.

Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스에서 Database Insights 기능 지원

다음 표에는 Database Insights 기능을 지원하는 Amazon Aurora DB 엔진이 나와 있습니다.

Feature	요금 티어	지원되는 리전	지원되는 DB 엔진	지원되는 인스턴스 클래스
성능 개선 도우미에 대한 SQL 통계	모두	모두	모두	모두
일정 기간 동안의 데이터베이스 성능 분석	유료 티어만	모두	모두	db.serverless(Aurora Serverless v2)를 제외한 모든 항목
성능 개선 도우미 사전 권장 사항 보기	유료 티어만	• 미국 동부(오하이오) • 미국 동부(버지니아 북부) • 미국 서부(캘리포니아 북부) • 미국 서부(오리건) • 아시아 태평양(뭄바이) • 아시아 태평양(서울) • 아시아 태평양(싱가포르) • 아시아 태평양(시드니) • 아시아 태평양(도쿄) • 캐나다(중부) • 유럽(프랑크푸르트)	모두	db.serverless(Aurora Serverless v2)를 제외한 모든 항목

Feature	요금 티어	지원되는 리전	지원되는 DB 엔진	지원되는 인스턴스 클래스
		• 유럽(아일랜드) • 유럽(런던) • 유럽(파리) • 유럽(스톡홀름) • 남아메리카(상파울루)		

Database Insights를 지원하는 Amazon Aurora 리전

Aurora가 다음 AWS 리전에서 Database Insights를 지원합니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(하이데라바드)
- 아시아 태평양(자카르타)
- 아시아 태평양(말레이시아)
- 아시아 태평양(멜버른)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 캐나다 서부(캘거리)

- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(밀라노)
- 유럽(파리)
- 유럽(스페인)
- 유럽(스톡홀름)
- 유럽(취리히)
- 이스라엘(텔아비브)
- 중동(바레인)
- 중동(UAE)
- 남아메리카(상파울루)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

Amazon Aurora에 Database Insights의 고급 모드 켜기

Amazon Aurora에 Database Insights의 고급 모드를 켜려면 다음 프로시저를 따르세요.

DB 클러스터를 만들 때 Database Insights의 고급 모드 켜기

Amazon Aurora에 대한 데이터베이스를 만들 때 Database Insights의 고급 모드를 켭니다.

Console

콘솔에서 DB 클러스터를 만들 때 Database Insights의 고급 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 데이터베이스 생성를 선택합니다.

4. Database Insights 섹션에서 고급 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.
   - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Database Insights의 고급 모드에서 보존 기간은 15개월이어야 합니다.
   - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
5. 데이터베이스 생성를 선택합니다.

AWS CLI

DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜려면 create-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--database-insights-mode advanced`를 사용하여 Database Insights의 고급 모드를 켭니다.
- `--engine` - DB 클러스터에 대한 데이터베이스 엔진입니다.
- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Database Insights를 켜려면 보존 기간이 최소 465일이어야 합니다.

다음 예시에서는 DB 클러스터를 만들 때 Database Insights의 고급 모드를 사용 설정합니다.

Linux, macOS, Unix의 경우:

```
aws rds create-db-cluster \
    --database-insights-mode advanced \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Windows의 경우:

```
aws rds create-db-cluster ^
```

```
    --database-insights-mode advanced ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜려면 CreateDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~advanced`
- `EnablePerformanceInsights~True`
- `PerformanceInsightsRetentionPeriod` 최소 465일로 설정

DB 클러스터를 수정할 때 Database Insights의 고급 모드 켜기

Amazon Aurora에 대한 데이터베이스를 수정할 때 Database Insights를 켭니다. Database Insights의 고급 모드를 활성화하도록 DB 클러스터를 수정해도 가동 중지가 발생하지 않습니다.

> **Note**
>
> Database Insights를 사용 설정하려면 DB 클러스터의 각 DB 인스턴스에 동일한 성능 개선 도우미 및 향상된 모니터링 설정이 있어야 합니다.

Console

콘솔에서 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. DB 인스턴스를 선택하고 수정(Modify)을 선택합니다.
4. Database Insights 섹션에서 고급 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.

  - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Database Insights의 고급 모드에서 보존 기간은 15개월이어야 합니다.
  - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.

5. Continue(계속)를 선택합니다.
6. 수정 스케줄링에 대해 즉시 적용을 선택합니다. 예약된 다음 유지 관리 기간에 적용을 선택하면 데이터베이스에서 이 설정을 무시하고 Database Insights의 고급 모드를 즉시 켭니다.
7. 클러스터 수정을 선택합니다.

AWS CLI

DB 클러스터를 수정 때 Database Insights의 고급 모드를 켜려면 modify-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--database-insights-mode advanced`를 사용하여 Database Insights의 고급 모드를 켭니다.
- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Database Insights의 고급 모드를 켜려면 보존 기간이 최소 465일이어야 합니다.

다음 예시에서는 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 사용 설정합니다.

Linux, macOS, Unix의 경우:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Windows의 경우:

```
aws rds modify-db-cluster ^
```

```
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

DB 클러스터를 수정 때 Database Insights의 고급 모드를 켜려면 ModifyDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~advanced`
- `EnablePerformanceInsights~True`
- `PerformanceInsightsRetentionPeriod` 최소 465일로 설정

Amazon Aurora에 Database Insights의 표준 모드 켜기

Amazon Aurora에 Database Insights의 표준 모드를 켜려면 다음 프로시저를 사용합니다.

DB 클러스터를 만들 때 Database Insights의 표준 모드 켜기

Amazon Aurora에 대한 데이터베이스를 만들 때 Database Insights의 표준 모드를 켭니다.

Console

콘솔에서 DB 클러스터를 만들 때 Database Insights의 표준 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 데이터베이스 생성를 선택합니다.
4. Database Insights 섹션에서 표준 모드를 선택합니다. 그런 다음, 다음 옵션 중에서 선택하여 성능 개선 도우미를 켜거나 끕니다.
   - 성능 개선 도우미를 끄려면 성능 개선 도우미 활성화를 선택 취소합니다.
   - 성능 개선 도우미를 켜려면 성능 개선 도우미 활성화를 선택합니다. 성능 개선 도우미를 구성하려면 다음 옵션을 지정합니다.

- 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. 보존 기간은 최소 7일이어야 합니다.
- AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.

5. 데이터베이스 생성를 선택합니다.

AWS CLI

DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜려면 create-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--database-insights-mode standard`를 사용하여 Database Insights의 표준 모드를 켭니다.
- `--engine` - DB 클러스터에 대한 데이터베이스 엔진입니다.
- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--enable-performance-insights` 또는 `--no-enable-performance-insights`를 사용하여 성능 개선 도우미를 켜거나 끕니다. `--enable-performance-insights`를 지정하는 경우 `--performance-insights-retention-period`도 지정해야 함 - DB 클러스터의 데이터 보존 기간입니다. 보존 기간은 최소 7일이어야 합니다.

다음 예시에서는 DB 클러스터를 만들 때 Database Insights의 표준 모드 및 성능 개선 도우미를 사용 설정합니다.

Linux, macOS, Unix의 경우:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
```

```
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

다음 예시에서는 DB 클러스터를 만들 때 Database Insights의 표준 모드를 사용 설정하고 성능 개선 도우미를 사용 해제합니다.

Linux, macOS, Unix의 경우:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜려면 CreateDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~standard`
- `EnablePerformanceInsights`를 `True` 또는 `False`로 지정합니다. `EnablePerformanceInsights`를 `True`로 설정한 경우 `PerformanceInsightsRetentionPeriod`를 7일 이상으로 설정해야 합니다.

DB 클러스터를 수정할 때 Database Insights의 표준 모드 켜기

Amazon Aurora에 대한 데이터베이스를 수정할 때 Database Insights의 표준 모드를 켭니다. Database Insights의 표준 모드를 활성화하도록 DB 클러스터를 수정해도 가동 중지가 발생하지 않습니다.

> **Note**
>
> Database Insights를 사용 설정하려면 DB 클러스터의 각 DB 인스턴스에 동일한 성능 개선 도우미 및 향상된 모니터링 설정이 있어야 합니다.

Console

콘솔에서 DB 클러스터를 수정할 때 Database Insights의 표준 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 수정할 때 Database Insights의 표준 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. DB 인스턴스를 선택하고 수정(Modify)을 선택합니다.
4. Database Insights 섹션에서 표준 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.
   - 성능 개선 도우미를 끄려면 성능 개선 도우미 활성화를 선택 취소합니다.
   - 성능 개선 도우미를 켜려면 성능 개선 도우미 활성화를 선택합니다. 성능 개선 도우미를 구성하려면 다음 옵션을 지정합니다.
     - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. 보존 기간은 최소 7일이어야 합니다.
     - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
5. Continue(계속)를 선택합니다.
6. 수정 스케줄링에 대해 즉시 적용을 선택합니다. 예약된 다음 유지 관리 기간에 적용을 선택하면 데이터베이스에서 이 설정을 무시하고 Database Insights의 표준 모드를 즉시 켭니다.
7. 클러스터 수정을 선택합니다.

AWS CLI

DB 클러스터를 수정 때 Database Insights의 표준 모드를 켜려면 [modify-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html) AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--database-insights-mode standard`를 사용하여 Database Insights의 표준 모드를 켭니다.
- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--enable-performance-insights` 또는 `--no-enable-performance-insights`를 사용하여 성능 개선 도우미를 켜거나 끕니다. `--enable-performance-insights`를 지정하는 경우 `--performance-insights-retention-period`도 지정해야 함 - DB 클러스터의 데이터 보존 기간입니다. 보존 기간은 최소 7일이어야 합니다.

다음 예시에서는 DB 클러스터를 수정할 때 Database Insights의 표준 모드 및 성능 개선 도우미를 사용 설정합니다.

Linux, macOS, Unix의 경우:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

다음 예시에서는 DB 클러스터를 수정할 때 Database Insights의 표준 모드를 사용 설정하고 성능 개선 도우미를 사용 해제합니다.

Linux, macOS, Unix의 경우:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
```

```
    --no-enable-performance-insights
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

DB 클러스터를 수정 때 Database Insights의 표준 모드를 켜려면 ModifyDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~standard`
- `EnablePerformanceInsights`를 `True` 또는 `False`로 지정합니다. `EnablePerformanceInsights`를 `True`로 설정한 경우 `PerformanceInsightsRetentionPeriod`를 7일 이상으로 설정해야 합니다.

Amazon Aurora용 Database Insights를 사용하여 느린 SQL 쿼리를 모니터링하도록 데이터베이스 구성

데이터베이스의 느린 SQL 쿼리를 모니터링하기 위해 Database Insights 대시보드의 느린 SQL 쿼리 섹션을 사용할 수 있습니다. 느린 SQL 쿼리를 모니터링하도록 데이터베이스를 구성하기 전에는 느린 SQL 쿼리 섹션이 비어 있습니다.

Database Insights 대시보드에서 느린 SQL 쿼리를 모니터링하는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch Database Insights용 데이터베이스 인스턴스 대시보드 보기를 참조하시기 바랍니다.

Database Insights를 사용하여 느린 SQL 쿼리를 모니터링하도록 데이터베이스를 구성하려면 다음 단계를 완료합니다.

1. CloudWatch Logs로 로그 내보내기를 활성화합니다.
2. DB 클러스터의 DB 클러스터 파라미터 그룹을 생성하거나 수정합니다.

로그 내보내기 구성에 대한 자세한 내용은 및 Amazon Aurora 사용 설명서의 Amazon CloudWatch Logs에 데이터베이스 로그 게시를 참조하세요.

DB 클러스터 파라미터 그룹을 만들거나 수정하려면 다음 주제를 참조하세요.

- Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성
- Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정

Amazon Aurora MySQL

느린 SQL 쿼리를 모니터링하도록 Amazon Aurora MySQL DB 클러스터를 구성하려면 다음 파라미터를 설정합니다.

- `slow_query_log` - `1`로 설정됨
- `long_query_time` - `1.0`으로 설정됨
- `log_output` - `FILE`로 설정됨

Amazon Aurora PostgreSQL

느린 SQL 쿼리를 모니터링하도록 Amazon Aurora PostgreSQL DB 클러스터를 구성하려면 다음 파라미터를 설정합니다. 이러한 파라미터를 설정하면 DB 클러스터의 성능이 저하될 수 있다는 점에 유의하세요.

- `log_min_duration_statement` - `1000`로 설정됨
- `log_statement` - `none`으로 설정됨
- `log_destination` - `stderr`로 설정됨

> **Note**
>
> Aurora MySQL의 경우 `long_query_time` 파라미터(1마이크로초 단위)를 구성할 수 있습니다. 예를 들어 이 파라미터를 `0.000001`로 설정할 수 있습니다. DB 인스턴스의 쿼리 양에 따라 `long_query_time` 파라미터 값이 성능을 저하시킬 수 있습니다. `1.0` 값으로 시작하고 워크로드에 따라 조정합니다. 이 파라미터를 `0`으로 설정하면 Database Insights는 모든 쿼리를 로깅합니다.

Aurora MySQL 및 Aurora PostgreSQL 로그에 대한 자세한 내용은 다음을 참조하시기 바랍니다.

- Aurora MySQL 데이터베이스 로그 파일
- Aurora PostgreSQL 데이터베이스 로그 파일

Amazon Aurora용 Database Insights의 고려 사항

다음은 Amazon Aurora용 Database Insights의 고려 사항입니다.

- DB 클러스터에서 DB 인스턴스에 대한 Database Insights를 관리할 수 없습니다.
- Database Insights의 고급 모드를 활성화하려면 성능 개선 도우미를 활성화하고 성능 개선 도우미 보존 기간을 최소 465일(15개월)로 설정해야 합니다. Database Insights 비용 외에 성능 개선 도우미 보존 기간을 15개월로 설정하는 데 드는 추가 비용은 없습니다. Database Insights 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하세요.
- Database Insights를 사용 설정하려면 DB 클러스터의 각 DB 인스턴스에 동일한 성능 개선 도우미 및 향상된 모니터링 설정이 있어야 합니다.
- Database Insights 모드 중 하나를 활성화하도록 DB 클러스터를 수정해도 가동 중지가 발생하지 않습니다.

성능 개선 도우미를 통한 Amazon Aurora 모니터링

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
>
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

성능 개선 도우미는 기존 Amazon Aurora 모니터링 기능을 확장한 것으로서 클러스터 성능을 표시하여 분석하는 데 효과적입니다. 성능 개선 도우미 대시보드를 사용하면 Amazon Aurora 클러스터 로드를 시각화하고 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 로드를 필터링할 수 있습니다. Amazon DocumentDB에서 성능 개선 도우미를 사용하는 방법에 대한 자세한 내용은 Amazon DocumentDB 개발자 가이드를 참조하세요.

주제

- [성능 개선 도우미 사전 권장 사항 보기]
- [Aurora용 성능 개선 도우미 API를 사용하여 지표 검색]
- [AWS CloudTrail을 사용하여 Performance Insights 호출 로깅]
- [성능 개선 도우미 API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)]

Amazon Aurora의 성능 개선 도우미 개요

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
>
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

기본적으로 RDS는 콘솔 생성 마법사에서 모든 Amazon RDS 엔진에 대해 성능 개선 도우미를 활성화합니다. DB 클러스터 수준에서 성능 개선 도우미를 활성화하면 RDS는 클러스터의 모든 DB 인스턴스에 대해 성능 개선 도우미를 활성화합니다. DB 인스턴스에 둘 이상의 데이터베이스가 있는 경우 성능 개선 도우미는 성능 데이터를 집계합니다.

다음 비디오에서 Amazon Aurora 성능 개선 도우미의 개요를 볼 수 있습니다.

성능 개선 도우미를 사용하여 Amazon Aurora PostgreSQL의 성능 분석

주제

- 데이터베이스 부하
- 최대 CPU
- Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 Performance Insights 지원
- 성능 개선 도우미의 요금 및 데이터 보존

데이터베이스 부하

데이터베이스 로드(DB 로드)는 데이터베이스의 세션 활동 수준을 측정합니다. DBLoad는 성능 개선 도우미의 주요 지표이며, 성능 개선 도우미는 1초마다 DB 로드를 수집합니다.

주제

- 활성 세션
- 평균 활성 세션
- 평균 활성 실행
- 차원

활성 세션

데이터베이스 세션은 관계형 데이터베이스와 애플리케이션의 대화를 나타냅니다. 활성 세션이란 DB 엔진에 작업을 제출하여 현재 응답 대기 중인 연결 세션을 말합니다.

세션은 CPU에서 실행 중이거나 리소스가 계속 진행될 수 있도록 대기 중일 때 활성화됩니다. 예를 들어 활성 세션은 페이지(또는 블록)가 메모리로 읽힐 때까지 기다린 다음 페이지에서 데이터를 읽는 동안 CPU를 사용할 수 있습니다.

평균 활성 세션

평균 활성 세션(AAS)은 성능 개선 도우미의 DBLoad 지표에 대한 단위입니다. 데이터베이스에서 동시에 활성화된 세션 수를 측정합니다.

성능 개선 도우미는 매초 쿼리를 동시에 실행하는 세션 수를 샘플링합니다. 각 활성 세션에 대해 성능 개선 도우미는 다음 데이터를 수집합니다.

- SQL 문
- 세션 상태(CPU에서 실행 중이거나 대기 중)

- Host
- SQL을 실행하는 사용자

성능 개선 도우미는 특정 기간 동안의 총 세션 수를 총 샘플 수로 나눠 AAS를 계산합니다. 예를 들어 다음 표는 1초 간격으로 채취된 실행 중인 쿼리의 연속된 5개 샘플을 보여 줍니다.

샘플	쿼리를 실행 중인 세션 수	AAS	계산
1	2	2	총 세션 2회/샘플 1개
2	0	1	총 세션 2회/샘플 2개
3	4	2	총 세션 6회/샘플 3개
4	0	1.5	총 세션 6회/샘플 4개
5	4	2	총 세션 10회/샘플 5개

이전 예시에서 시간 간격에 대한 DB 로드는 2 AAS입니다. 이 측정은 5개의 샘플을 채취한 간격 동안 평균적으로 2회의 세션이 동시에 활성화 상태였음을 의미합니다.

평균 활성 실행

초당 평균 활성 실행(AAE)은 AAS와 관련이 있습니다. Performance Insights는 AAE를 계산하기 위해 쿼리의 총 실행 시간을 시간 간격으로 나눕니다. 다음 표는 앞의 표와 동일한 쿼리의 AAE 계산을 보여 줍니다.

경과 시간(초)	총 실행 시간(초)	AAE	계산
60	120	2	120초 실행/60초 경과
120	120	1	120초 실행/120초 경과
180	380	2.11	380초 실행/180초 경과
240	380	1.58	380초 실행/240초 경과
300	600	2	600초 실행/300초 경과

대부분의 경우 쿼리의 AAS와 AAE는 거의 동일합니다. 하지만 계산에 입력되는 데이터 원본이 다르기 때문에 계산 결과가 약간 다른 경우가 많습니다.

차원

`db.load` 지표는 차원이라는 하위 구성 요소로 구분할 수 있다는 점에서 다른 시계열 지표와 다릅니다. 차원을 `DBLoad` 지표의 다양한 특성에 대한 "분할 기준" 범주로 생각할 수 있습니다.

성능 문제를 진단할 때 다음과 같은 차원이 가장 유용한 경우가 많습니다.

주제

- 대기 이벤트
- 상위 SQL

Aurora 엔진의 전체 차원 목록은 차원을 기준으로 분할된 DB 로드 섹션을 참조하세요.

대기 이벤트

대기 이벤트란 SQL 문이 계속 실행되려면 특정 이벤트가 발생할 때까지 기다려야 합니다. 대기 이벤트는 작업이 방해되는 위치를 나타내므로 DB 로드에 대해 중요한 측정기준이나 범주입니다.

모든 활성 세션이 CPU에서 실행 중이거나 대기 중입니다. 예를 들어 세션은 메모리에서 버퍼를 검색하거나 계산을 수행하거나 프로시저 코드를 실행할 때 CPU를 사용합니다. 세션에서 CPU를 사용하지 않는 경우 메모리 버퍼가 비어 있거나 읽을 데이터 파일 또는 기록할 로그가 나올 때까지 대기할 수 있습니다. 세션이 리소스를 기다리는 시간이 길수록 CPU에서 실행되는 시간이 줄어듭니다.

데이터베이스를 튜닝할 때 세션이 기다리는 리소스를 찾으려고 하는 경우가 많습니다. 예를 들어 두 개 또는 세 개의 대기 이벤트가 DB 로드의 90%를 차지할 수 있습니다. 이 측정은 평균적으로 활성 세션이 소수의 리소스를 기다리는 데 대부분의 시간을 소비한다는 것을 의미합니다. 이러한 대기의 원인을 찾을 수 있는 경우 해결 방법을 시도할 수 있습니다.

대기 이벤트는 DB 엔진마다 다릅니다.

- Aurora MySQL의 일반적인 대기 이벤트 목록을 보려면 Aurora MySQL 대기 이벤트 섹션을 참조하세요. 이러한 대기 이벤트를 사용하여 튜닝하는 방법에 대한 자세한 내용은 Aurora MySQL 튜닝 섹션을 참조하세요.
- 모든 MySQL 대기 이벤트에 대한 자세한 내용은 MySQL 설명서의 대기 이벤트 요약 테이블을 참조하세요.

- Aurora PostgreSQL의 일반적인 대기 이벤트 목록을 보려면 Amazon Aurora PostgreSQL 대기 이벤트 섹션을 참조하세요. 이러한 대기 이벤트를 사용하여 튜닝하는 방법에 대한 자세한 내용은 Aurora PostgreSQL의 대기 이벤트를 사용한 튜닝 섹션을 참조하세요.
- 모든 PostgreSQL 대기 이벤트에 대한 자세한 내용은 PostgreSQL 설명서의 통계 수집기 > 대기 이벤트 테이블을 참조하세요.

상위 SQL

대기 이벤트는 병목 현상을 표시하는 반면, 상위 SQL은 DB 로드에 가장 많이 기여하는 쿼리를 보여줍니다. 예를 들어 현재 데이터베이스에서 여러 쿼리가 실행 중이더라도 단일 쿼리가 DB 부하의 99%를 소비할 수 있습니다. 이 경우 부하가 높으면 쿼리에 문제가 있음을 나타낼 수 있습니다.

기본적으로 성능 개선 도우미 콘솔에는 데이터베이스 부하에 영향을 미치는 상위 SQL 쿼리가 표시됩니다. 콘솔에는 각 문에 관련된 통계도 표시됩니다. 특정 문의 성능 문제를 진단하기 위해 실행 계획을 검사할 수 있습니다.

최대 CPU

대시보드에서 데이터베이스 로드 차트는 세션 정보를 수집, 집계 및 표시합니다. 활성 세션이 최대 CPU를 초과하는지 확인하려면 최대 vCPU 선과의 관계를 확인합니다. 성능 개선 도우미는 최대 vCPU 값을 DB 인스턴스에서 vCPU(가상 CPU) 코어의 수로 결정합니다. Aurora Serverless v2의 경우 최대 vCPU(Max vCPU)는 예상 vCPU 수를 나타냅니다.

한 번에 하나의 프로세스를 vCPU에서 실행할 수 있습니다. 프로세스 수가 vCPUs 수를 초과하면 프로세스가 대기열에 추가되기 시작합니다. 대기열이 늘어나면 데이터베이스 성능이 저하됩니다. DB 로드가 최대 vCPU 선을 상회하는 경우가 잦아지고 CPU가 기본 대기 상태라면 CPU에서 과부하가 발생한 것입니다. 이 경우 연결 수를 인스턴스에 맞게 조절하거나, CPU 부하가 높은 SQL 쿼리를 모두 조정하거나, 인스턴스 클래스의 크기를 늘리는 것이 좋습니다. 대기 상태의 인스턴스가 높고 일관적이라는 것은 해결해야 할 병목 현상이나 리소스 경합 문제가 있을 수 있음을 나타냅니다. DB 로드가 최대 vCPU 선을 넘지 않는다 하더라도 이러한 문제가 나타날 수 있습니다.

Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 Performance Insights 지원

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니

다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.

Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.

조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.

CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

다음 테이블에는 성능 개선 도우미를 지원하는 Amazon Aurora DB 엔진이 나와 있습니다.

Amazon Aurora DB 엔진	지원되는 엔진 버전 및 리전	인스턴스 클래스 제한 사항
Amazon Aurora MySQL 호환 버전	Aurora MySQL을 사용한 성능 개선 도우미 인사이트의 버전 및 리전 가용성에 대한 자세한 내용은 Aurora MySQL을 사용하는 성능 개선 도우미 섹션을 참조하세요.	성능 개선 도우미에는 다음과 같은 엔진 클래스 제한 사항이 있습니다. • db.t2 - 지원되지 않음 • db.t3 - 지원되지 않음 • db.t4g.micro 및 db.t4g.small – 지원되지 않음
Amazon Aurora PostgreSQL 호환 에디션	Aurora PostgreSQL 기반 성능 개선 도우미 인사이트의 버전 및 지역 가용성에 대한 자세한 내용은 Aurora PostgreSQL을 사용하는 성	N/A

Amazon Aurora DB 엔진	지원되는 엔진 버전 및 리전	인스턴스 클래스 제한 사항
	능 개선 도우미 섹션을 참조하세요.	

성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원

다음 테이블에는 성능 개선 도우미 기능을 지원하는 Amazon Aurora DB 엔진이 나와 있습니다.

Feature	요금 티어	지원되는 리전	지원되는 DB 엔진	지원되는 인스턴스 클래스
성능 개선 도우미에 대한 SQL 통계	모두	모두	모두	모두
일정 기간 동안의 데이터베이스 성능 분석	유료 티어만	모두	모두	db.serverless(Aurora Serverless v2)를 제외한 모든 항목
성능 개선 도우미 사전 권장 사항 보기	유료 티어만	• 미국 동부(오하이오) • 미국 동부(버지니아 북부) • 미국 서부(캘리포니아 북부) • 미국 서부(오리건) • 아시아 태평양(뭄바이) • 아시아 태평양(서울)	모두	db.serverless(Aurora Serverless v2)를 제외한 모든 항목

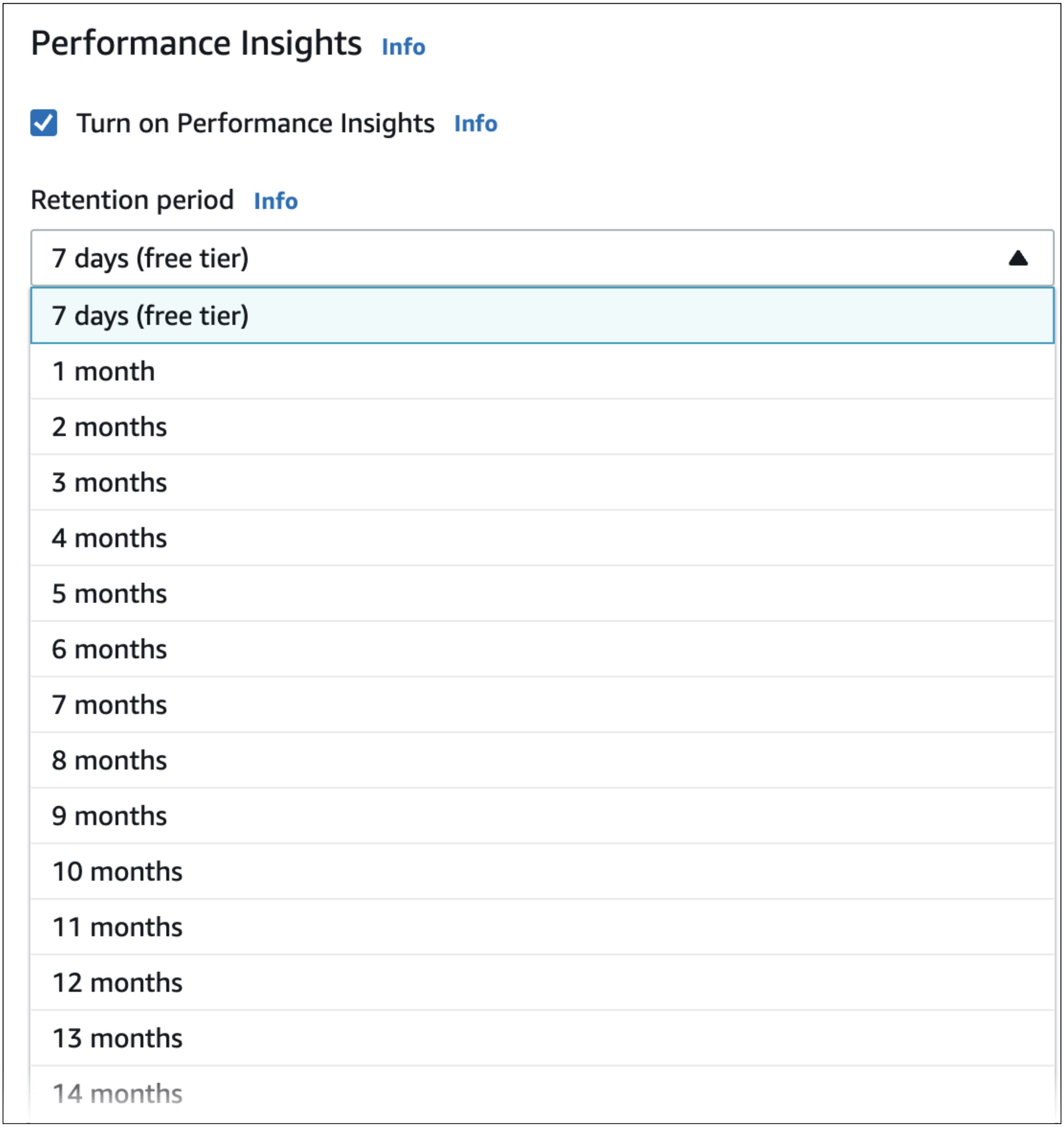
Feature	요금 티어	지원되는 리전	지원되는 DB 엔진	지원되는 인스턴스 클래스
		• 아시아 태평양(싱가포르) • 아시아 태평양(시드니) • 아시아 태평양(도쿄) • 캐나다(중부) • 유럽(프랑크푸르트) • 유럽(아일랜드) • 유럽(런던) • 유럽(파리) • 유럽(스톡홀름) • 남아메리카(상파울루)		

성능 개선 도우미의 요금 및 데이터 보존

기본적으로 성능 개선 도우미는 7일간의 성능 데이터 기록과 매달 1백만 건의 API 요청이 포함된 프리 티어를 제공합니다. 더 긴 보존 기간을 구매할 수도 있습니다. 전체 요금 정보는 성능 개선 도우미 요금을 참조하세요.

RDS 콘솔에서는 성능 개선 도우미 데이터에 대해 다음과 같은 보존 기간을 선택할 수 있습니다.

- 기본값(7일)
- ***n***개월(***n***은 1~24 사이의 숫자)



AWS CLI를 사용하여 보존 기간을 설정하는 방법은 ??? 섹션에서 알아보세요.

Aurora의 성능 개선 도우미 설정 및 해제

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 Amazon Aurora에 Database Insights의 고급 모드 켜기 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
>
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

DB 클러스터를 생성할 때 Performance Insights를 활성화할 수 있습니다. 필요한 경우 콘솔에서 DB 클러스터 를 수정하여 나중에 비활성화할 수 있습니다. 성능 개선 도우미를 활성화하거나 비활성화해도 가동 중지, 재부팅 또는 장애 조치가 발생하지 않습니다.

> **Note**
>
> 성능 스키마는 Aurora MySQL에서 사용하는 선택적 성능 도구입니다. 성능 스키마를 켜거나 끄면 재부팅해야 합니다. 그러나 성능 개선 도우미를 켜거나 끌 경우에는 재부팅할 필요가 없습니다. 자세한 내용은 Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요 섹션을 참조하세요.

Aurora Global Database와 함께 Performance Insights를 사용하는 경우, 각 AWS 리전에서 데이터베이스에 대해 개별적으로 Performance Insights를 활성화해야 합니다. 자세한 내용은 Amazon RDS 성능 인사이트를 사용하여 Amazon Aurora Global Database 모니터링을 참조하세요.

성능 개선 도우미 에이전트는 DB 호스트에서 제한된 CPU 및 메모리를 사용합니다. DB 로드가 높을 경우 에이전트는 데이터 수집 빈도를 줄여 성능에 미치는 영향을 제한합니다.

Console

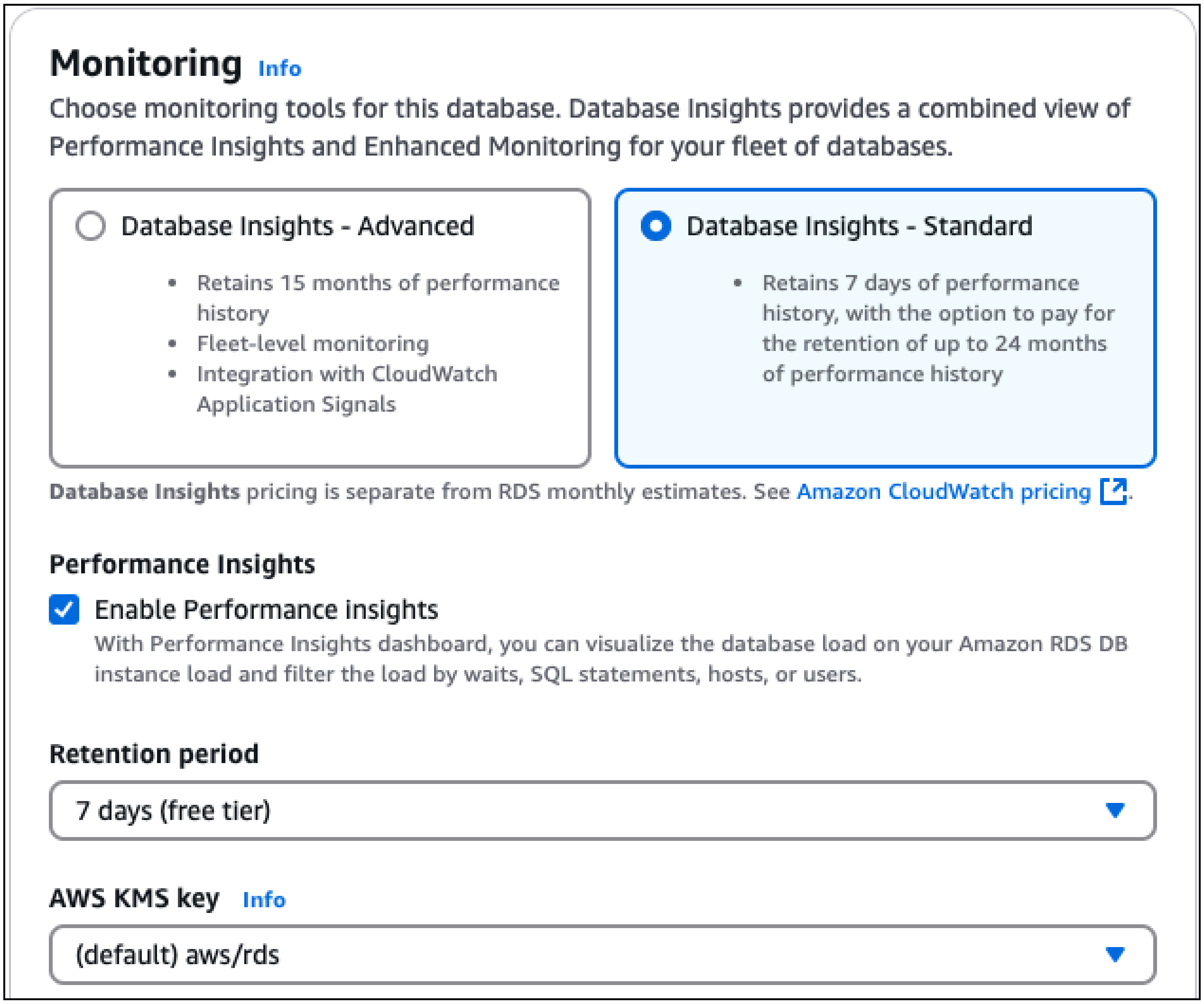
콘솔에서 DB 클러스터를 생성할 때 성능 개선 도우미를 설정하거나 해제할 수 있습니다. 성능 개선 도우미를 활성화하면 DB 클러스터에 대해 성능 개선 도우미 설정 및 옵션을 관리할 수 있습니다. 클러스터 수준 설정은 클러스터의 모든 DB 인스턴스에 적용됩니다.

DB 클러스터를 만들 때 성능 개선 도우미 활성화 및 비활성화

새 DB 클러스터를 만든 후 Amazon RDS는 기본적으로 성능 개선 도우미를 활성화합니다. DB 클러스터에 대해 성능 개선 도우미를 끄려면 Database Insights – 표준 옵션을 선택하고 성능 개선 도우미 활성화 옵션을 선택 취소합니다.

DB 클러스터를 만들려면 Amazon Aurora DB 클러스터 생성의 DB 엔진에 대한 지침을 따르세요.

다음 이미지는 성능 개선 도우미 섹션의 스크린샷입니다.



성능 개선 도우미 활성화를 선택하면 다음 옵션이 있습니다.

- 보존(Database Insights의 표준 모드에만 해당) - 성능 개선 도우미 데이터를 보존하는 시간입니다. 프리 티어의 보존 설정은 기본값(7일)입니다. 성능 데이터를 더 오래 보존하려면 1~24개월을 지정하십시오. 보존 기간에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.
- AWS KMS key – AWS KMS key을(를) 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 성능 개선 도우미를 위한 AWS KMS 정책 변경 섹션을 참조하세요.

DB 클러스터를 수정할 때 Performance Insights 켜기 또는 끄기

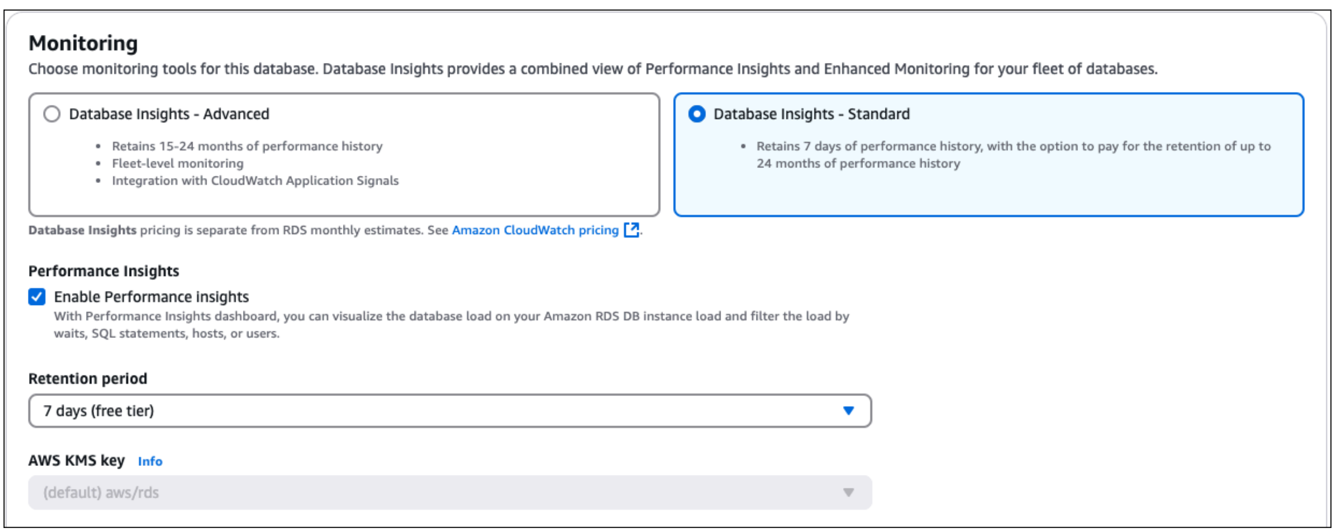
콘솔에서 DB 클러스터 를 수정하여 Performance Insights를 관리할 수 있습니다.

콘솔을 사용하여 DB 클러스터에 대해 Performance Insights를 관리하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. DB 클러스터를 선택하고 수정을 선택합니다.
4. 성능 개선 도우미를 켜려면 성능 개선 도우미 활성화를 선택합니다. DB 클러스터에 대해 성능 개선 도우미를 끄려면 Database Insights – 표준 옵션을 선택하고 성능 개선 도우미 활성화 옵션을 선택 취소합니다.

   성능 개선 도우미 활성화를 선택하면 다음 옵션이 있습니다.

   - 보존(Database Insights의 표준 모드에만 해당) - 성능 개선 도우미 데이터를 보존하는 시간입니다. 프리 티어의 보존 설정은 기본값(7일)입니다. 성능 데이터를 더 오래 보존하려면 1~24개월을 지정하십시오. 보존 기간에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.
   - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.



5. Continue(계속)를 선택합니다.
6. Scheduling of Modifications(수정 사항 예약)에 대해 Apply immediately(즉시 적용)를 선택합니다. 예약된 다음 유지 관리 윈도우에 적용을 선택하면 인스턴스에서 이 설정을 무시하고 성능 개선 도우미를 즉시 활성화합니다.

7. Modify Instance(인스턴스 수정)를 선택합니다.

AWS CLI

create-db-instance AWS CLI 명령을 사용하는 경우, `--enable-performance-insights`를 지정하고 `--database-insights-mode`를 `advanced` 또는 `standard`로 설정하여 성능 개선 도우미를 활성화합니다. 성능 개선 도우미를 비활성화하려면 `--no-enable-performance-insights`를 지정하고 `database-insights-mode`를 `standard`로 설정합니다.

다음 AWS CLI 명령을 사용해 다음 값을 지정할 수도 있습니다.

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replica
- modify-db-instance
- restore-db-instance-from-s3

AWS CLI를 사용하여 DB 클러스터의 성능 개선 도우미를 관리하려면

- modify-db-cluster AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` - DB 클러스터에서 DB 인스턴스의 이름
  - `--enable-performance-insights`로 활성화 또는 `--no-enable-performance-insights`로 비활성화
  - `database-insights-mode` - DB 클러스터에 대한 Database Insights 모드입니다. 성능 개선 도우미를 끄려면 이 값을 `standard`로 설정합니다.

  다음 예제에서는 `sample-db-cluster`에 대한 성능 개선 도우미를 활성화합니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights
```

  Windows의 경우:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights
```

CLI에서 성능 개선 도우미를 활성화할 때 필요에 따라 `--performance-insights-retention-period` 옵션을 사용하여 성능 개선 도우미 데이터를 보존할 시간을 일 단위로 지정할 수 있습니다. 7, *#* * 31(*#*은 1~23 사이의 숫자여야 함) 또는 731이라고 지정할 수 있습니다. 예를 들어 성능 데이터를 3개월 동안 보존하려면 3에 31을 곱한 93을 지정하면 됩니다. 기본값은 7일입니다. 보존 기간에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.

다음 예에서는 `sample-db-cluster`의 성능 개선 도우미를 활성화하고 성능 개선 도우미 데이터를 93일(3개월) 동안 보존하도록 지정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

Windows의 경우:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

보존 기간을 94일처럼 유효하지 않은 값으로 지정하면 RDS에서 오류가 발생합니다.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

> **Note**
>
> 성능 개선 도우미가 클러스터 수준에서 관리되지 않는 DB 클러스터의 인스턴스에 대해서만 성능 개선 도우미를 전환할 수 있습니다.

RDS API

CreateDBInstance 작업 Amazon RDS API 작업을 사용하여 DB 클러스터 내에서 새 DB 인스턴스를 생성할 때 `EnablePerformanceInsights`를 `True`로 설정하면 성능 개선 도우미가 활성화됩니다. 성능 개선 도우미를 끄려면 `EnablePerformanceInsights`를 `False`로, `DatabaseInsightsMode`를 `standard`로 설정합니다.

다음 API 작업으로 `EnablePerformanceInsights` 값을 지정할 수도 있습니다.

- CreateDBCluster
- ModifyDBCluster
- ModifyDBInstance
- CreateDBInstanceReadReplica
- RestoreDBInstanceFromS3

성능 개선 도우미를 활성화할 때 선택적으로 `PerformanceInsightsRetentionPeriod` 파라미터를 사용하여 성능 개선 도우미 데이터를 보존할 시간을 일 단위로 지정할 수 있습니다. 7, *# ** 31(#은 1~23 사이의 숫자여야 함) 또는 731이라고 지정할 수 있습니다. 예를 들어 성능 데이터를 3개월 동안 보존하려면 3에 31을 곱한 93을 지정하면 됩니다. 기본값은 7일입니다. 보존 기간에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.

Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요

성능 스키마는 Aurora MySQL 런타임 성능을 낮은 세부 수준에서 모니터링하기 위한 선택적 기능입니다. 성능 스키마는 데이터베이스 성능에 미치는 영향을 최소화하도록 설계되었습니다. 성능 개선 도우미는 성능 스키마 사용 여부와 상관없이 사용할 수 있는 별도의 기능입니다.

주제

- 성능 스키마 개요
- 성능 개선 도우미 및 성능 스키마

- [성능 개선 도우미의 성능 스키마 자동 관리]
- [성능 스키마에서 리부팅할 때의 효과]
- [성능 개선 도우미의 성능 스키마 관리 여부 확인]
- [Aurora MySQL에서 성능 스키마 활성화]

성능 스키마 개요

성능 스키마가 Aurora MySQL 데이터베이스의 이벤트를 모니터링합니다. 이벤트는 시간을 소비하고 타이밍 정보를 수집할 수 있도록 계측된 데이터베이스 서버 작업입니다. 이벤트의 예는 다음과 같습니다.

- 함수 호출
- 운영 체제 대기
- SQL 실행 단계
- SQL 문 그룹

`PERFORMANCE_SCHEMA` 스토리지 엔진은 성능 스키마 기능을 구현하기 위한 메커니즘입니다. 이 엔진은 데이터베이스 소스 코드의 계측을 사용하여 이벤트 데이터를 수집합니다. 엔진은 수집된 이벤트를 `performance_schema` 데이터베이스의 메모리 전용 테이블에 저장합니다. 다른 테이블과 마찬가지로 `performance_schema`를 쿼리할 수 있습니다. 자세한 내용은 MySQL 참조 설명서에서 [MySQL 성능 스키마]를 참조하세요.

성능 개선 도우미 및 성능 스키마

성능 개선 도우미와 성능 스키마는 별개의 기능이지만 연결되어 있습니다. Aurora MySQL의 성능 개선 도우미의 동작은 성능 스키마가 켜져 있는지 여부와 켜져 있으면 성능 개선 도우미가 성능 스키마를 자동으로 관리하는지 여부에 따라 다릅니다. 다음 표는 동작에 대한 설명입니다.

성능 스키마 켜짐	성능 개선 도우미 관리 모드	성능 개선 도우미 행동
예	자동	• 세부적인 저수준 모니터링 정보 수집 • 1초마다 활성 세션 지표 수집 •

성능 스키마 켜짐	성능 개선 도우미 관리 모드	성능 개선 도우미 행동
		병목 현상을 식별하는 데 사용할 수 있는 세부 대기 이벤트별로 분류된 DB 로드 표시
예	수동	• 대기 이벤트 및 SQL별 지표 수집 • 1초마다 활성 세션 지표 수집 • 삽입 및 전송과 같은 사용자 상태를 보고하므로 병목 현상을 식별하는 데 도움이 되지 않음
No	N/A	• 대기 이벤트, SQL별 지표 또는 기타 세부적인 저수준 모니터링 정보를 수집하지 않음 • 1초가 아닌 5초마다 활성 세션 지표 수집 • 삽입 및 전송과 같은 사용자 상태를 보고하므로 병목 현상을 식별하는 데 도움이 되지 않음

성능 개선 도우미의 성능 스키마 자동 관리

성능 개선 도우미를 활성화한 상태에서 Aurora MySQL DB 인스턴스를 생성하면 성능 스키마도 활성화됩니다. 이 경우 성능 개선 도우미는 성능 스키마 파라미터를 자동으로 관리합니다. 이는 권장되는 구성입니다.

성능 개선 도우미에서 성능 스키마를 자동으로 관리하는 경우 `performance_schema`의 소스는 `System default`입니다.

> **Note**
>
> t4g.medium 인스턴스 클래스에는 성능 스키마의 자동 관리가 지원되지 않습니다.

성능 스키마를 수동으로 관리할 수도 있습니다. 이 옵션을 선택하는 경우 다음 표에 나와 있는 값에 따라 매개변수를 설정합니다.

파라미터 이름	파라미터값
performance_schema	1 (소스(Source) 열의 값이 Modified임)
performance-schema-consumer-events-waits-current	ON
performance-schema-instrument	wait/%=ON
performance_schema_consumer_global_instrumentation	1
performance_schema_consumer_thread_instrumentation	1

performance_schema 파라미터 값을 수동으로 변경한 후 나중에 자동 관리로 되돌리려면 Aurora MySQL에서 성능 스키마 활성화 섹션을 참조하세요.

> **Important**
> 성능 개선 도우미가 성능 스키마를 활성화하더라도 파라미터 그룹 값은 변경되지 않습니다. 그러나 실행 중인 DB 인스턴스에 대한 값이 변경됩니다. 변경된 값을 볼 수있는 유일한 방법은 SHOW GLOBAL VARIABLES 명령을 실행하는 것입니다.

성능 스키마에서 리부팅할 때의 효과

성능 개선 도우미와 성능 스키마는 DB 인스턴스 재부팅에 대한 요구 사항이 다릅니다.

성능 스키마

이 기능을 활성화하거나 비활성화하기 위해 DB 인스턴스를 재부팅해야 합니다.

성능 개선 도우미

이 기능을 활성화하거나 비활성화하기 위해 DB 인스턴스를 재부팅하지 않아도 됩니다.

현재 성능 스키마가 활성화되어 있지 않고 DB 인스턴스를 재부팅하지 않고 성능 개선 도우미를 활성화하면 성능 스키마가 활성화되지 않습니다.

성능 개선 도우미의 성능 스키마 관리 여부 확인

성능 개선 도우미가 현재 지원되는 모든 메이저 엔진 버전에 대한 성능 스키마를 관리하고 있는지 확인하려면 다음 표를 검토하세요.

performance_schema 파라미터 설정	소스 열 설정	성능 개선 도우미가 성능 스키마를 관리하는가?
`0`	`System default`	예
`0` 또는 `1`	`Modified`	No

다음 절차에서는 성능 개선 도우미가 성능 스키마를 자동으로 관리하는지 확인해 봅니다.

성능 개선 도우미가 성능 스키마를 자동으로 관리하는지 확인하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 파라미터 그룹을 선택합니다.
3. DB 인스턴스에 대한 파라미터 그룹 이름을 선택합니다.
4. 검색줄에 **`performance_schema`**를 입력합니다.
5. 소스가 시스템 기본값이고 값이 0인지 확인합니다. 그렇다면 성능 개선 도우미가 성능 스키마를 자동으로 관리하고 있습니다.

   여기에 표시된 예제에서는 성능 개선 도우미가 성능 스키마를 자동으로 관리하지 않습니다.

Aurora MySQL에서 성능 스키마 활성화

DB 인스턴스 에 대해 성능 개선 도우미가 활성화되어 있지만 현재 성능 스키마를 관리하고 있지 않다고 가정합시다. 성능 개선 도우미가 성능 스키마를 자동으로 관리하도록 허용하려면 다음 단계를 완료하세요.

자동 관리를 위한 성능 스키마 구성 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 파라미터 그룹을 선택합니다.
3. DB 인스턴스에 대한 파라미터 그룹 이름을 선택합니다.
4. 편집을 선택합니다.
5. 검색줄에 **performance_schema**를 입력합니다.
6. `performance_schema` 파라미터를 선택합니다.
7. 기본값으로 설정을 선택합니다.
8. 기본값으로 설정을 선택하여 확인합니다.
9. 변경 사항 저장(Save Changes)을 선택합니다.
10. DB 인스턴스를 재부팅합니다.

> **Important**
>
> 성능 스키마를 활성화하거나 비활성화할 때마다 DB 인스턴스를 재부팅해야 합니다.

인스턴스 파라미터 수정에 대한 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하십시오. 대시보드에 대한 자세한 내용은 성능 개선 도우미 대시보드를 사용한 지표 분석 단원을 참조하십시오. MySQL 성능 스키마에 대한 자세한 내용은 MySQL 설명서에서 MySQL Performance Schema(8.0의 경우) 및 MySQL Performance Schema(8.4의 경우)를 참조하세요.

Performance Insights에 대한 액세스 정책 구성

성능 개선 도우미에 액세스하려면 보안 주체에게 AWS Identity and Access Management(IAM)의 적절한 권한이 있어야 합니다. 다음과 같은 방법으로 액세스를 부여할 수 있습니다.

- `AmazonRDSPerformanceInsightsReadOnly` 관리형 정책을 권한 세트 또는 역할에 연결하여 성능 개선 도우미 API의 모든 읽기 전용 작업에 액세스할 수 있습니다. 다음의 CloudWatch 권한(`GetMetricStatistics`, `ListMetrics` 및 `GetMetricData`)을 연결합니다. CloudWatch 권한에 대한 자세한 내용은 Amazon CloudWatch 권한 참조를 참조하시기 바랍니다.
- `AmazonRDSPerformanceInsightsFullAccess` 관리형 정책을 권한 세트 또는 역할에 연결하여 성능 개선 도우미 API의 모든 작업에 액세스할 수 있습니다. 다음의 CloudWatch 권한

(`GetMetricStatistics`, `ListMetrics` 및 `GetMetricData`)을 연결합니다. CloudWatch 권한에 대한 자세한 내용은 Amazon CloudWatch 권한 참조를 참조하시기 바랍니다.

- 사용자 지정 IAM 정책을 생성하고 권한 세트 또는 역할에 연결합니다.

성능 개선 도우미를 활성화할 때 고객 관리형 키를 지정한 경우 계정의 사용자에게 AWS KMS key에서 `kms:Decrypt` 및 `kms:GenerateDataKey` 권한이 있는지 확인합니다.

다음 섹션에서는 IAM 보안 주체에 AWS 관리형 정책을 연결하고, 사용자 지정 IAM 정책을 생성하고, AWS KMS 정책을 변경하고, 성능 개선 도우미에 대한 세분화된 액세스 권한을 부여합니다.

주제

- IAM 보안 주체에 AmazonRDSPerformanceInsightsReadOnly 정책 연결
- IAM 보안 주체에 AmazonRDSPerformanceInsightsFullAccess 정책 연결
- 성능 개선 도우미를 위한 사용자 지정 IAM 정책 만들기
- 성능 개선 도우미를 위한 AWS KMS 정책 변경
- 성능 개선 도우미에 대한 세분화된 액세스 권한 부여

IAM 보안 주체에 AmazonRDSPerformanceInsightsReadOnly 정책 연결

`AmazonRDSPerformanceInsightsReadOnly`는 Amazon RDS 성능 개선 도우미 API의 모든 읽기 전용 작업에 대한 액세스 권한을 부여하는 AWS 관리형 정책입니다.

`AmazonRDSPerformanceInsightsReadOnly`를 권한 세트 또는 역할에 연결하면 수신자는 성능 개선 도우미를 다른 콘솔 기능과 함께 사용할 수 있습니다.

자세한 내용은 AWS 관리형 정책: AmazonRDSPerformanceInsightsReadOnly 단원을 참조하십시오.

IAM 보안 주체에 AmazonRDSPerformanceInsightsFullAccess 정책 연결

`AmazonRDSPerformanceInsightsFullAccess`는 Amazon RDS 성능 개선 도우미 API의 모든 작업에 대한 액세스 권한을 부여하는 AWS 관리형 정책입니다.

`AmazonRDSPerformanceInsightsFullAccess`를 권한 세트 또는 역할에 연결하면 수신자는 성능 개선 도우미를 다른 콘솔 기능과 함께 사용할 수 있습니다.

자세한 내용은 AWS 관리형 정책: AmazonRDSPerformanceInsightsFullAccess 단원을 참조하십시오.

성능 개선 도우미를 위한 사용자 지정 IAM 정책 만들기

`AmazonRDSPerformanceInsightsReadOnly` 또는 `AmazonRDSPerformanceInsightsFullAccess` 정책이 없는 사용자의 경우, 사용자 관리형 IAM 정책을 생성 또는 수정하여 성능 개선 도우미에 대한 액세스 권한을 부여할 수 있습니다. IAM 권한 세트 또는 역할에 이 정책을 연결하면 수신자가 성능 개선 도우미를 사용할 수 있습니다.

사용자 지정 정책을 생성하는 방법

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 Policies를 선택합니다.
3. 정책 생성을 선택합니다.
4. 정책 생성 페이지에서 JSON 옵션을 선택합니다.
5. AmazonRDSPerformanceInsightsReadOnly 또는 AmazonRDSPerformanceInsightsFullAccess 정책에 대한 AWS 관리형 정책 참조 안내서의 JSON 정책 문서 섹션에 제공된 텍스트를 복사하여 붙여넣습니다.
6. 정책 검토를 선택합니다.
7. 정책의 이름과 설명(선택 사항)을 지정한 다음 정책 검토를 선택합니다.

이제 정책을 권한 세트 또는 역할에 연결할 수 있습니다. 다음 절차에서는 이 목적으로 사용할 수 있는 사용자가 이미 있다고 가정합니다.

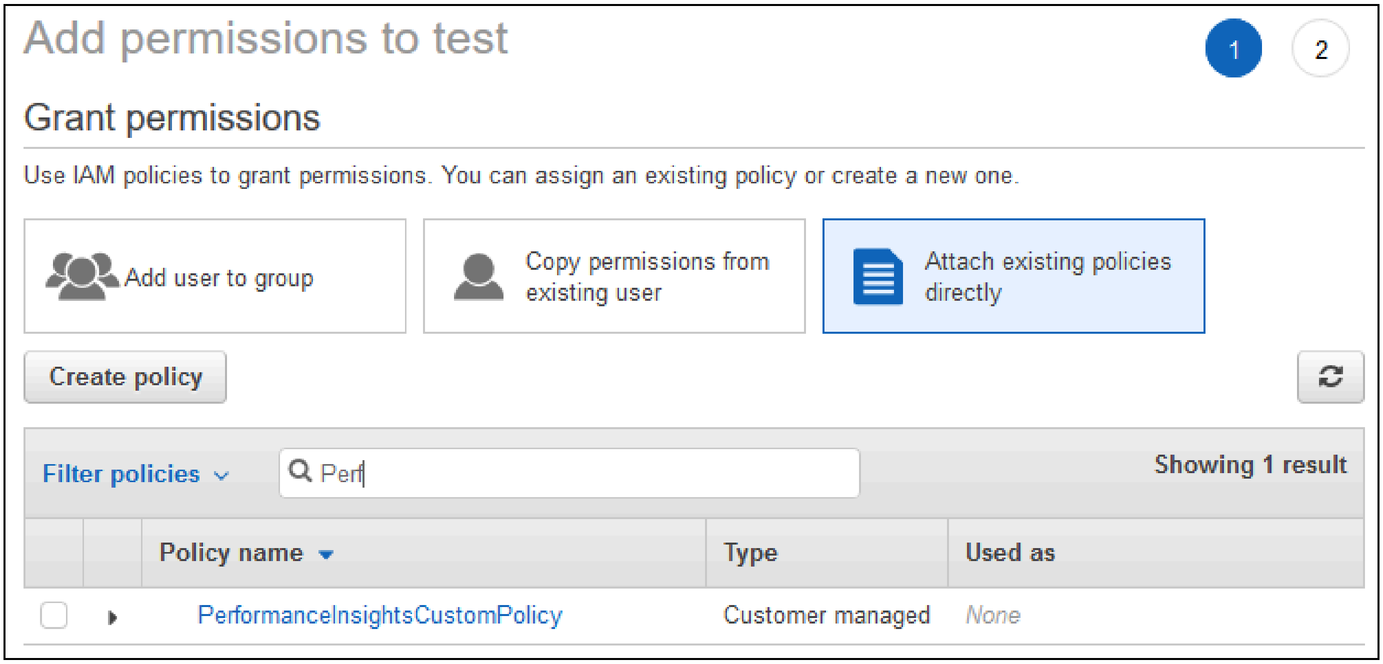
사용자에게 정책을 연결

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 목록에서 기존 사용자를 선택합니다.

> **Important**
>
> 성능 개선 도우미를 사용하려면 사용자 지정 정책 외에 Amazon RDS에 대한 액세스 권한이 있어야 합니다. 예를 들어 `AmazonRDSPerformanceInsightsReadOnly` 사전 정의 정책은 Amazon RDS에 대한 읽기 전용 액세스를 제공합니다. 자세한 내용은 정책을 사용하여 액세스 관리 섹션을 참조하세요.

4. [Summary] 페이지에서 [Add permissions]를 선택합니다.

5. 기존 정책 직접 첨부를 선택합니다. 검색에 다음 이미지와 같이 정책 이름의 첫 문자 몇 개를 입력합니다.



6. 정책을 선택하고 다음: 검토를 선택합니다.
7. 권한 추가를 선택합니다.

성능 개선 도우미를 위한 AWS KMS 정책 변경

성능 개선 도우미는 AWS KMS key을(를) 사용하여 민감한 데이터를 암호화합니다. API 또는 콘솔을 통해 성능 개선 도우미를 활성화하면 다음 중 한 가지를 수행할 수 있습니다.

- 기본 AWS 관리형 키를 선택합니다.

  Amazon RDS는 새 DB 인스턴스에 대해 AWS 관리형 키을(를) 사용합니다. Amazon RDS는 AWS 계정에 대해 AWS 관리형 키를 생성합니다. AWS 계정에 각 AWS 리전의 Amazon RDS에 대해 각기 다른 AWS 관리형 키가 있습니다.

- 고객 관리형 키를 선택합니다.

  고객 관리형 키를 지정하는 경우 성능 개선 도우미 API를 호출하는 계정의 사용자는 KMS 키에 대한 `kms:Decrypt` 및 `kms:GenerateDataKey` 권한이 필요합니다. IAM 정책을 통해 이러한 권한을 구성할 수 있습니다. 그러나 KMS 키 정책을 통해 이러한 권한을 관리하는 것이 좋습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS KMS의 키 정책을 참조하세요.

Example

다음 예는 KMS 키 정책에 문을 추가하는 방법을 보여줍니다. 이러한 문을 통해 Performance Insights에 액세스할 수 있습니다. KMS 키를 사용하는 방법에 따라 일부 제한 사항을 변경할 수 있습니다. 정책에 문을 추가하기 전에 모든 문을 제거하세요.

```
{
"Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS instance
{
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition" : {
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace region with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific RDS instance.
            //The value is a DbiResourceId.
```

```
            "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
         }
     }
}
```

성능 개선 도우미에서 AWS KMS 고객 관리형 키를 사용하는 방법

성능 개선 도우미는 고객 관리형 키를 사용하여 민감한 데이터를 암호화합니다. 성능 개선 도우미를 켜면 API를 통해 AWS KMS 키를 입력할 수 있습니다. 성능 개선 도우미는 이 키에 대한 KMS 권한을 생성합니다. 여기서는 키를 사용하고 민감한 데이터를 처리하는 데 필요한 작업을 수행합니다. 민감한 데이터에는 사용자, 데이터베이스, 애플리케이션, SQL 쿼리 텍스트 등의 필드가 포함됩니다. 성능 개선 도우미는 저장된 데이터와 전송 중인 데이터 모두에서 데이터가 암호화된 상태로 유지되도록 합니다.

성능 개선 도우미 IAM가 AWS KMS와 작동하는 방식

IAM은 특정 API에 권한을 부여합니다. 성능 개선 도우미에는 IAM 정책을 사용하여 제한할 수 있는 다음과 같은 공개 API가 포함됩니다.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

다음 API 요청을 사용하여 민감한 데이터를 가져올 수 있습니다.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

API를 사용하여 민감한 데이터를 가져오는 경우 성능 개선 도우미는 호출자의 보안 인증 정보를 활용합니다. 이 확인을 통해 민감한 데이터에 대한 액세스가 KMS 키에 액세스할 수 있는 사용자로 제한됩니다.

이러한 API를 호출하려면 IAM 정책을 통해 API를 호출할 수 있는 권한과 AWS KMS 키 정책을 통해 `kms:decrypt` 작업을 직접적으로 호출할 수 있는 권한이 있어야 합니다.

`GetResourceMetrics` API는 민감한 데이터와 그렇지 않은 데이터를 모두 반환할 수 있습니다. 요청 파라미터로는 응답에 민감한 데이터를 포함해야 하는지 여부를 결정합니다. 요청 시 필터 또는 그룹별 파라미터에 민감한 차원이 포함된 경우 API는 민감한 데이터를 반환합니다.

`GetResourceMetrics` API와 함께 사용할 수 있는 차원에 관한 자세한 내용은 DimensionGroup을 참조하세요.

Example 예시

다음 예시에서는 다음 `db.user` 그룹에서 민감한 데이터를 요청합니다.

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

다음 예시에서는 다음 `db.load.avg` 지표에서 민감하지 않은 데이터를 요청합니다.

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

성능 개선 도우미에 대한 세분화된 액세스 권한 부여

세분화된 액세스 제어를 사용하면 추가적인 방법으로 성능 개선 도우미에 대한 액세스를 제어할 수 있습니다. 이 액세스 제어는 `GetResourceMetrics`, `DescribeDimensionKeys` 및 `GetDimensionKeyDetails` 성능 개선 도우미 작업의 개별 차원에 대한 액세스를 허용하거나 거부할 수 있습니다. 세분화된 액세스를 사용하려면 조건 키를 사용하여 IAM 정책에서 차원을 지정하세요. 액세스 평가는 IAM 정책 평가 로직을 따릅니다. 자세한 내용은 IAM 사용 설명서의 [정책 평가 로직]을 참조하세요. IAM 정책 문에 차원이 지정되지 않은 경우 해당 문은 지정된 작업의 모든 차원에 대한 액세스를 제어합니다. 사용 가능한 차원 목록은 [DimensionGroup] 섹션을 참조하세요.

자격 증명에 액세스 권한이 부여된 차원을 확인하려면 `ListAvailableResourceDimensions`에서 `AuthorizedActions` 파라미터를 사용하고 작업을 지정하세요. `AuthorizedActions`에 허용되는 값은 다음과 같습니다.

- `GetResourceMetrics`
- `DescribeDimensionKeys`

- `GetDimensionKeyDetails`

예를 들어 `AuthorizedActions` 파라미터에 `GetResourceMetrics`를 지정하면 `ListAvailableResourceDimensions`는 `GetResourceMetrics` 작업에 액세스 권한이 부여된 차원 목록이 반환됩니다. `AuthorizedActions` 파라미터에 여러 작업을 지정하는 경우 `ListAvailableResourceDimensions`는 해당 작업에 액세스 권한이 부여된 차원의 교차 항목을 반환합니다.

Example

다음 예시에서는 `GetResourceMetrics` 및 `DescribeDimensionKeys` 작업에 지정된 차원에 대한 액세스를 제공합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    // only these dimensions are allowed. Dimensions not included in
                    // a policy with "Allow" effect will be denied
```

```
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

요청된 차원에 대한 응답은 다음과 같습니다.

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

다음 예시에서는 차원에 대해 액세스 허용 1개와 거부 액세스 거부 2개를 지정합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },

          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
```

```
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

요청된 차원에 대한 응답은 다음과 같습니다.

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
```

```
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
```

```
                   // removed from response because denied by the IAM Policy
                   //  { "Identifier": "db.application.name" }
                 ]
             },
             {
                 "Group": "db.sql_tokenized",
                 "Dimensions": [
                     { "Identifier": "db.sql_tokenized.id" },
                     { "Identifier": "db.sql_tokenized.db_id" },

                   // allowed for DescribeDimensionKeys because our IAM Policy
                   // denies it only for GetResourceMetrics
                     { "Identifier": "db.sql_tokenized.statement" }
                 ]
             },
             ...
         ] }
     ]
}
```

성능 개선 도우미 대시보드를 사용한 지표 분석

> **Important**
>
> AWS는 Performance Insights의 수명 종료일을 2025년 11월 30일로 발표했습니다. 이 날짜 이후에는 Amazon RDS가 더 이상 Performance Insights 콘솔 환경, 유연한 보존 기간(1~24개월) 및 연결 요금을 지원하지 않습니다. Performance Insights API는 요금 변경 없이 계속 제공됩니다. Performance Insights API 비용은 CloudWatch Database Insights 비용과 함께 AWS 청구서에 표시됩니다.
>
> Performance Insights의 유료 티어를 사용하는 DB 클러스터를 2025년 11월 30일 이전에 Database Insights의 고급 모드로 업그레이드하는 것이 좋습니다. Database Insights의 고급 모드로 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora에 Database Insights의 고급 모드 켜기](#) 섹션을 참조하세요.
>
> 조치를 취하지 않으면 Performance Insights를 사용하는 DB 클러스터가 기본적으로 Database Insights의 표준 모드를 사용하게 됩니다. Database Insights의 표준 모드를 사용하는 경우 7일이 지난 성능 데이터 기록에 액세스할 수 없으며 Amazon RDS 콘솔에서 실행 계획 및 온디맨드 분석 기능을 사용하지 못할 수 있습니다. 2025년 11월 30일 이후에는 Database Insights의 고급 모드만 실행 계획과 온디맨드 분석을 지원합니다.
>
> CloudWatch Database Insights를 사용하면 데이터베이스 플릿의 데이터베이스 로드를 모니터링하며 대규모로 성능을 분석하고 문제를 해결할 수 있습니다. Database Insights에 대한 자세

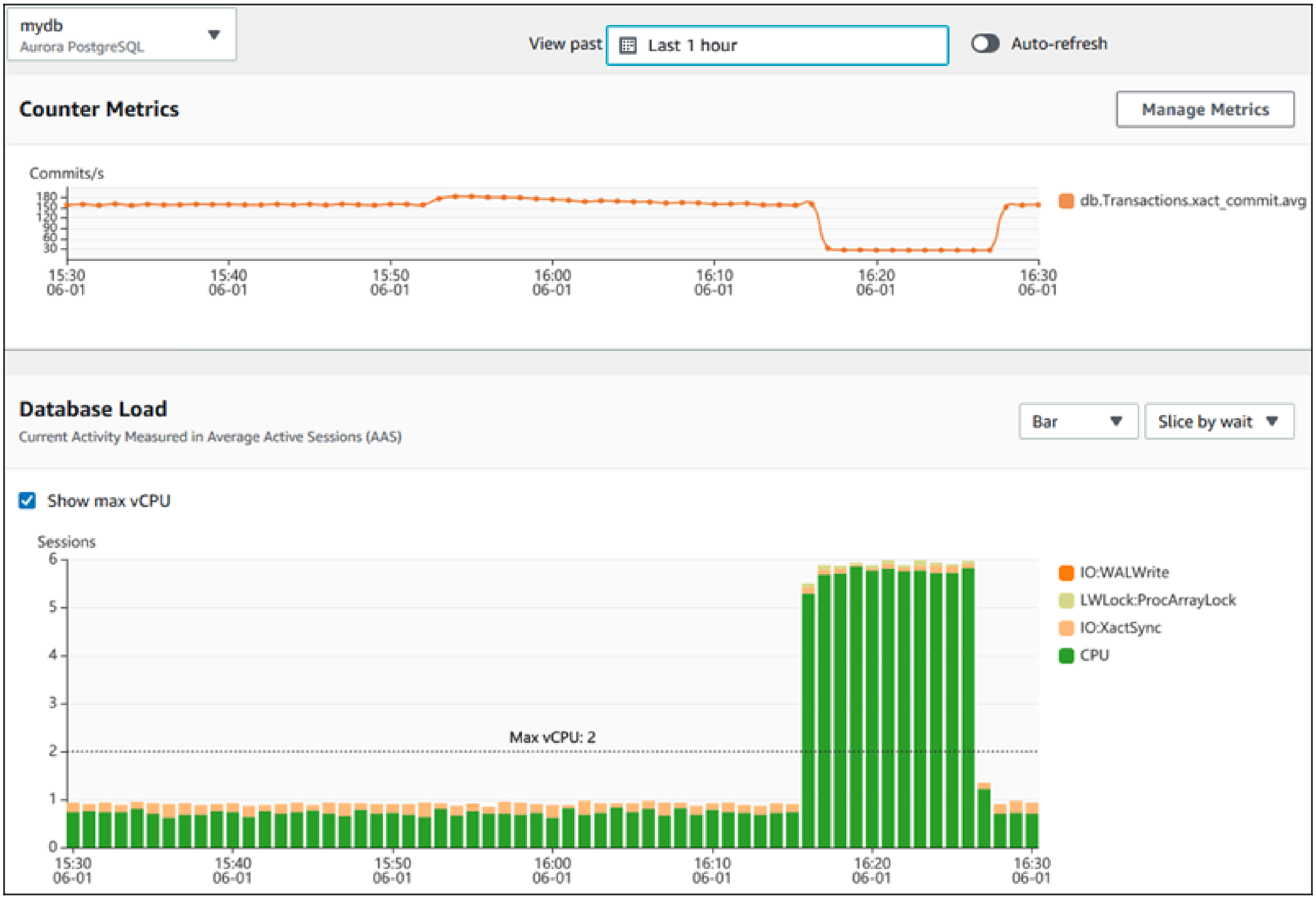
한 내용은 CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링 섹션을 참조하세요. 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

성능 개선 도우미 대시보드에는 성능 문제를 분석하여 해결할 수 있는 데이터베이스 성능 정보가 포함됩니다. 메인 대시보드 페이지에서 데이터베이스 부하에 대한 정보를 확인할 수 있습니다. 대기 이벤트 또는 SQL과 같은 차원을 기준으로 DB 로드를 "분할"할 수 있습니다.

성능 개선 도우미 대시보드

성능 개선 도우미 대시보드 개요

대시보드는 성능 개선 도우미와 상호 작용하는 가장 간편한 방법입니다. 다음 예제는 PostgreSQL DB 인스턴스의 대시보드를 보여줍니다.



주제

- [시간 범위 필터]
- [카운터 지표 차트]
- [데이터베이스 로드 차트]
- [상위 측정기준 테이블]

시간 범위 필터

성능 개선 도우미 대시보드는 기본적으로 마지막 1시간 동안 수집된 데이터를 표시합니다. 이 범위를 최소 5분 또는 최대 2년으로 조정할 수 있습니다. 상대적인 범위를 직접 선택할 수도 있습니다.

시작 및 종료 날짜와 시간이 있는 절대 범위를 선택할 수 있습니다. 다음 예에서는 2024년 9월 25일 자정에 시작해서 2024년 9월 28일 오후 11:59에 끝나는 시간 범위를 보여줍니다.

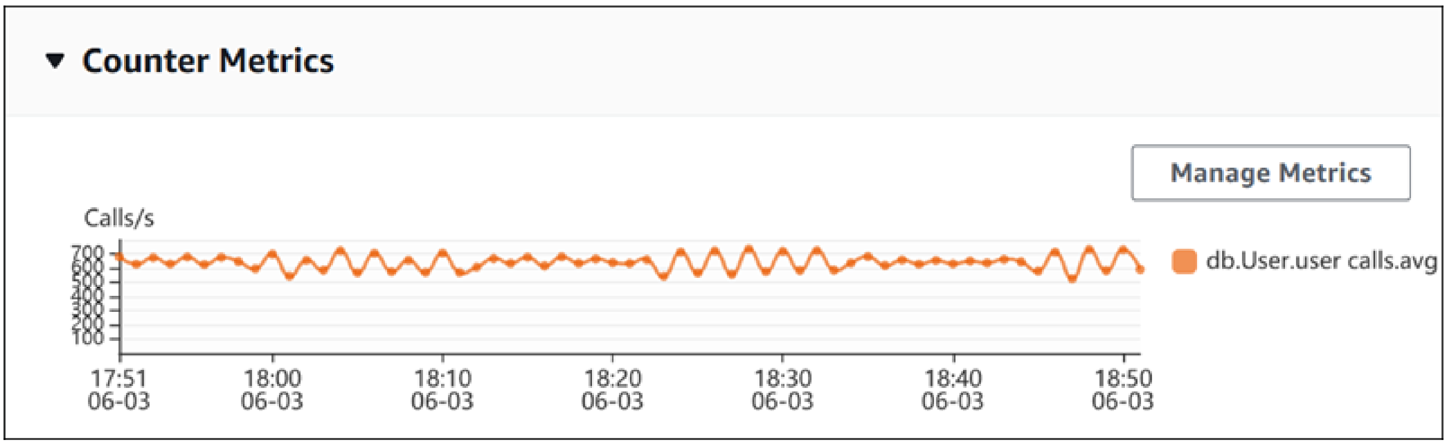
기본적으로 성능 개선 도우미 대시보드의 시간대는 협정 세계시(UTC)입니다. 현지 시간대를 선택할 수도 있습니다.

카운터 지표 차트

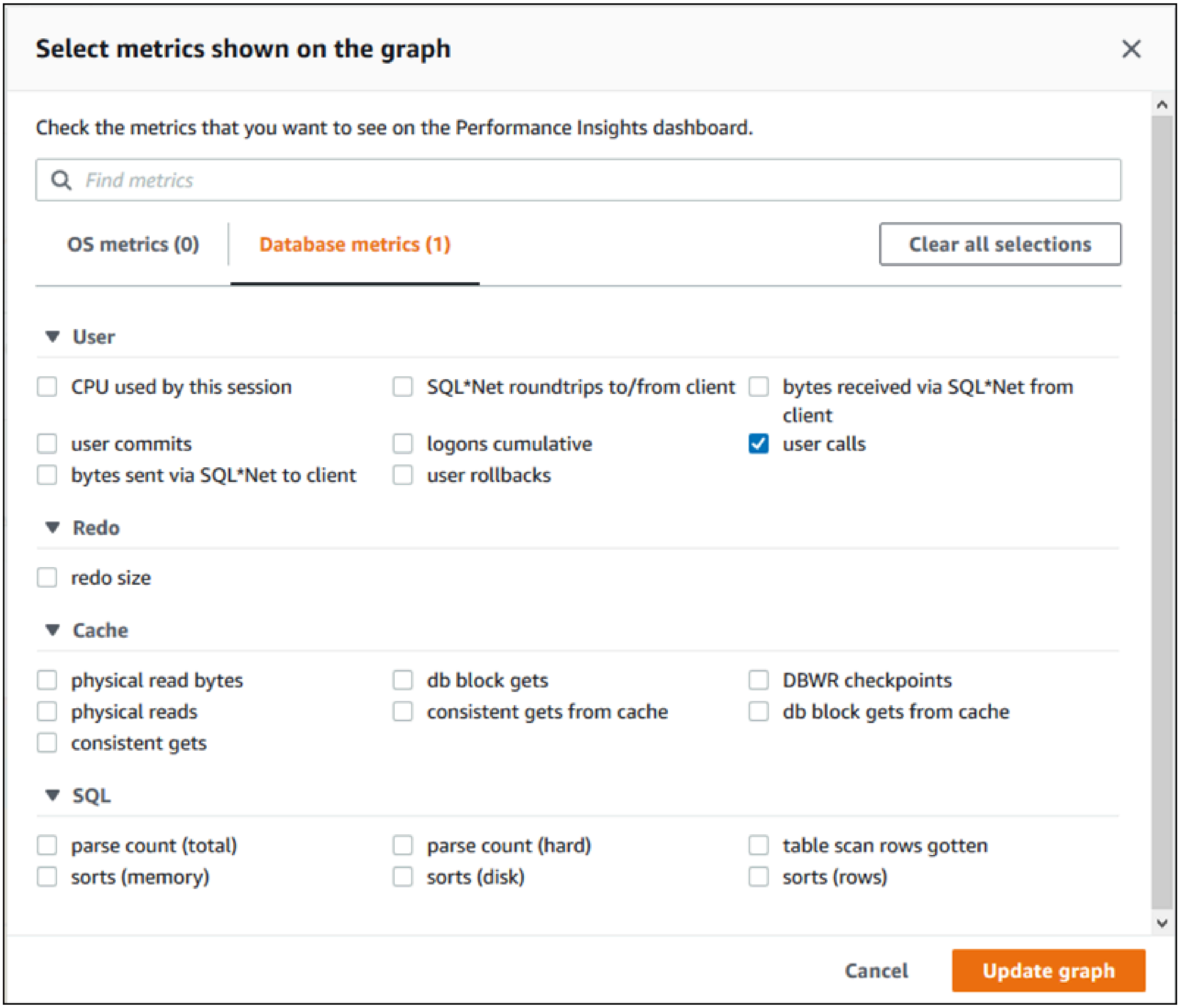
계수기 지표를 통해 성능 개선 도우미 대시보드에 최대 10개의 추가 그래프가 포함되도록 사용자 지정할 수 있습니다. 이 그래프에는 수십 건의 운영 체제 및 데이터베이스 성능 지표 모음이 표시됩니다. 이 정보와 데이터베이스 로드를 연관지으면 성능 문제를 식별하고 분석하는 데 도움이 됩니다.

카운터 지표 차트에는 성능 카운터의 데이터가 표시됩니다. 기본 지표는 DB 엔진에 따라 다릅니다.

- Aurora MySQL - `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL – `db.Transactions.xact_commit.avg`



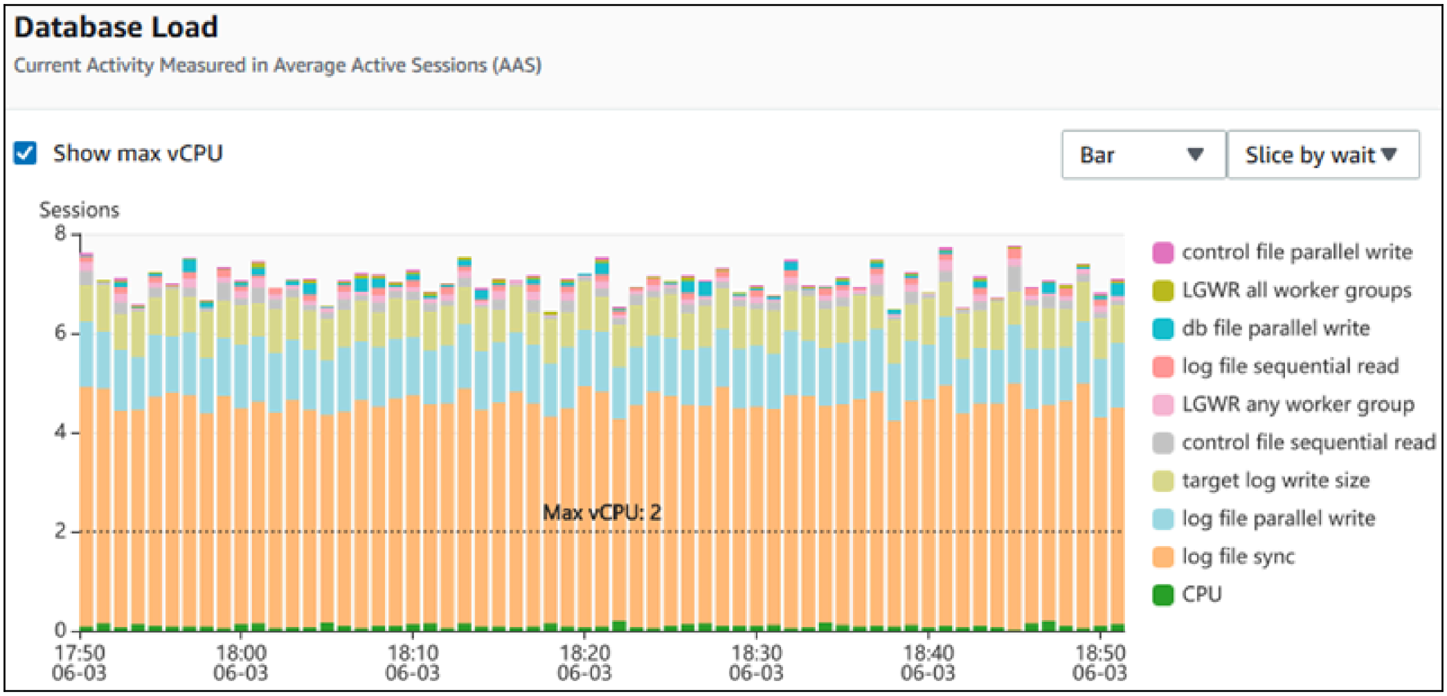
지표 관리(Manage Metrics)를 선택하여 성능 카운터를 변경합니다. 다음 스크린샷과 같이 여러 OS 지표 또는 데이터베이스 지표를 선택할 수 있습니다. 지표에 대한 세부 정보를 보려면 지표 이름 위에 마우스 포인터를 놓습니다.



각 DB 엔진에 추가할 수 있는 카운터 지표에 대한 설명은 [성능 개선 도우미 카운터](#) 섹션을 참조하세요.

데이터베이스 로드 차트

데이터베이스 로드 차트는 데이터베이스 로드와 DB 인스턴스 용량을 비교하여 최대 vCPU 선으로 표시합니다. 기본적으로 누적 꺾은선형 차트는 단위 시간당 평균 활성 세션으로 DB 로드를 나타냅니다. DB 로드는 대기 상태에 따라 슬라이스(그룹화) 됩니다.

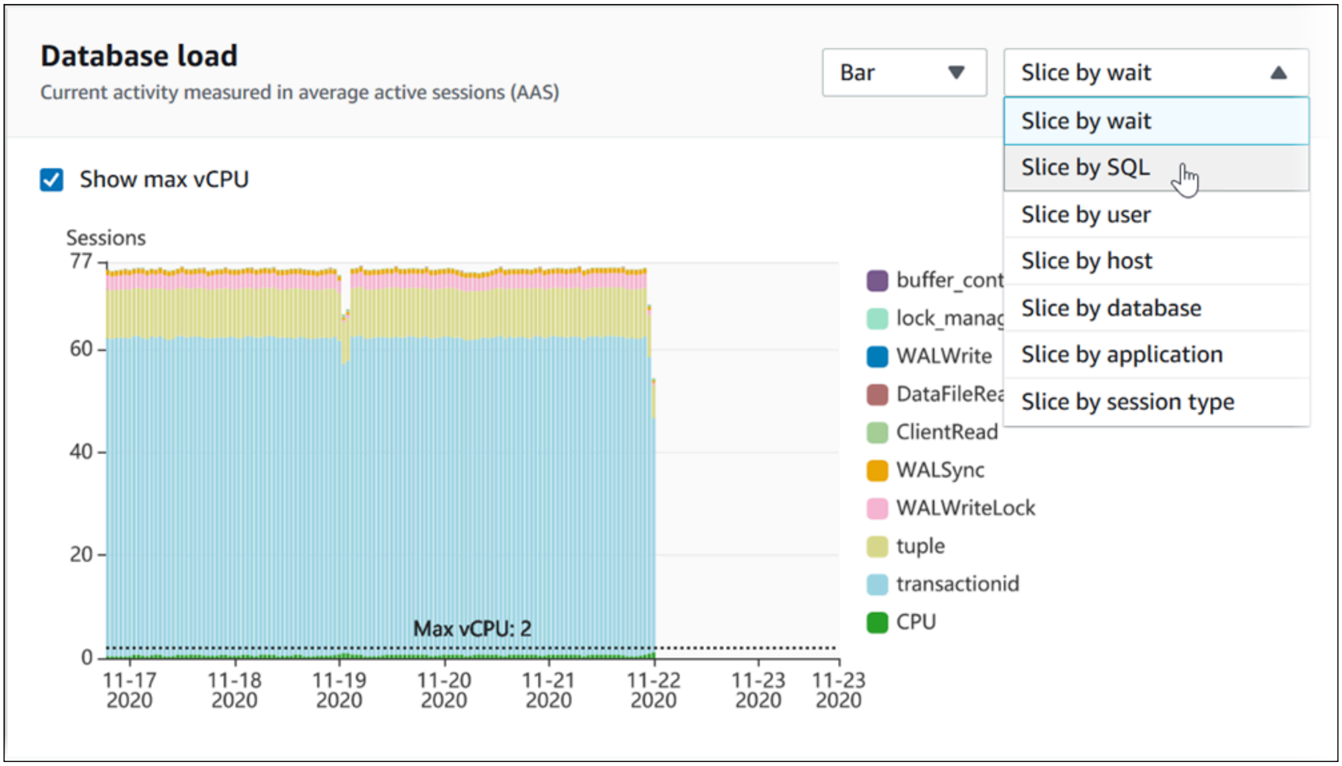


차원을 기준으로 분할된 DB 로드

지원되는 차원별로 그룹화된 활성 세션으로 로드를 표시하도록 선택할 수 있습니다. 다음 표에서는 다양한 엔진에 지원되는 차원을 보여줍니다.

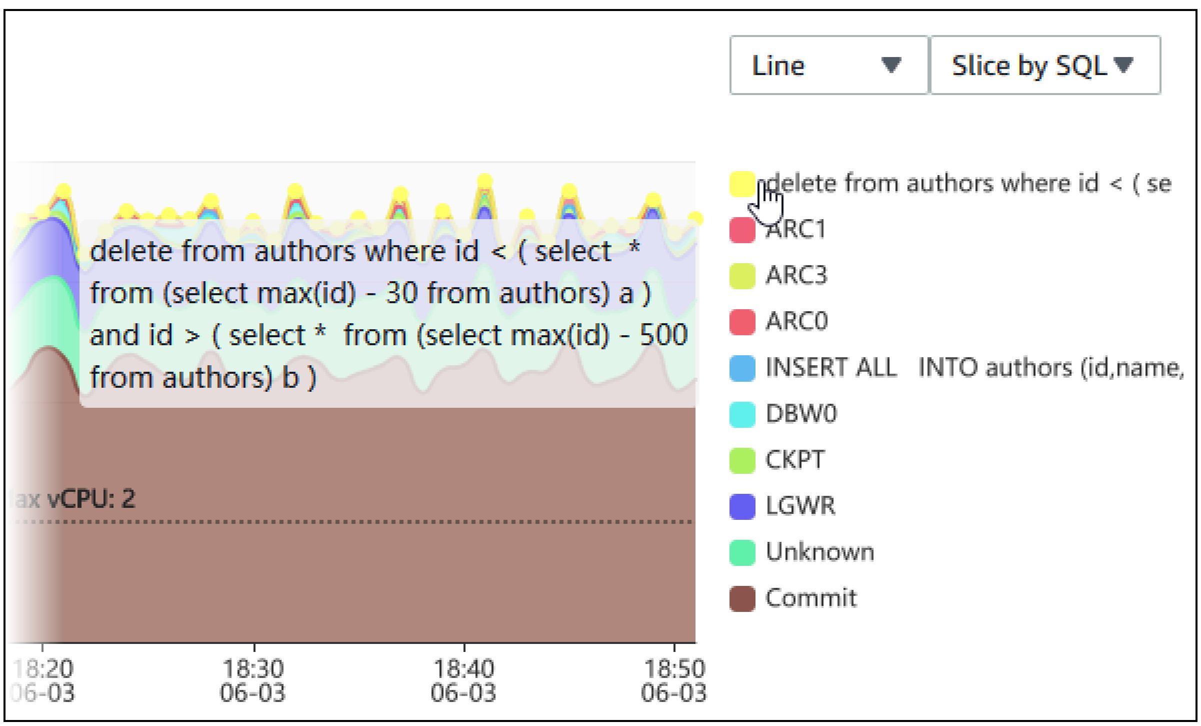
차원	Aurora PostgreSQL	Aurora MySQL
Host	예	예
SQL	예	예
User	예	예
대기	예	예
Application	예	No
데이터베이스	예	예
세션 유형	예	No

다음 이미지에서는 PostgreSQL DB 인스턴스의 차원을 보여줍니다.

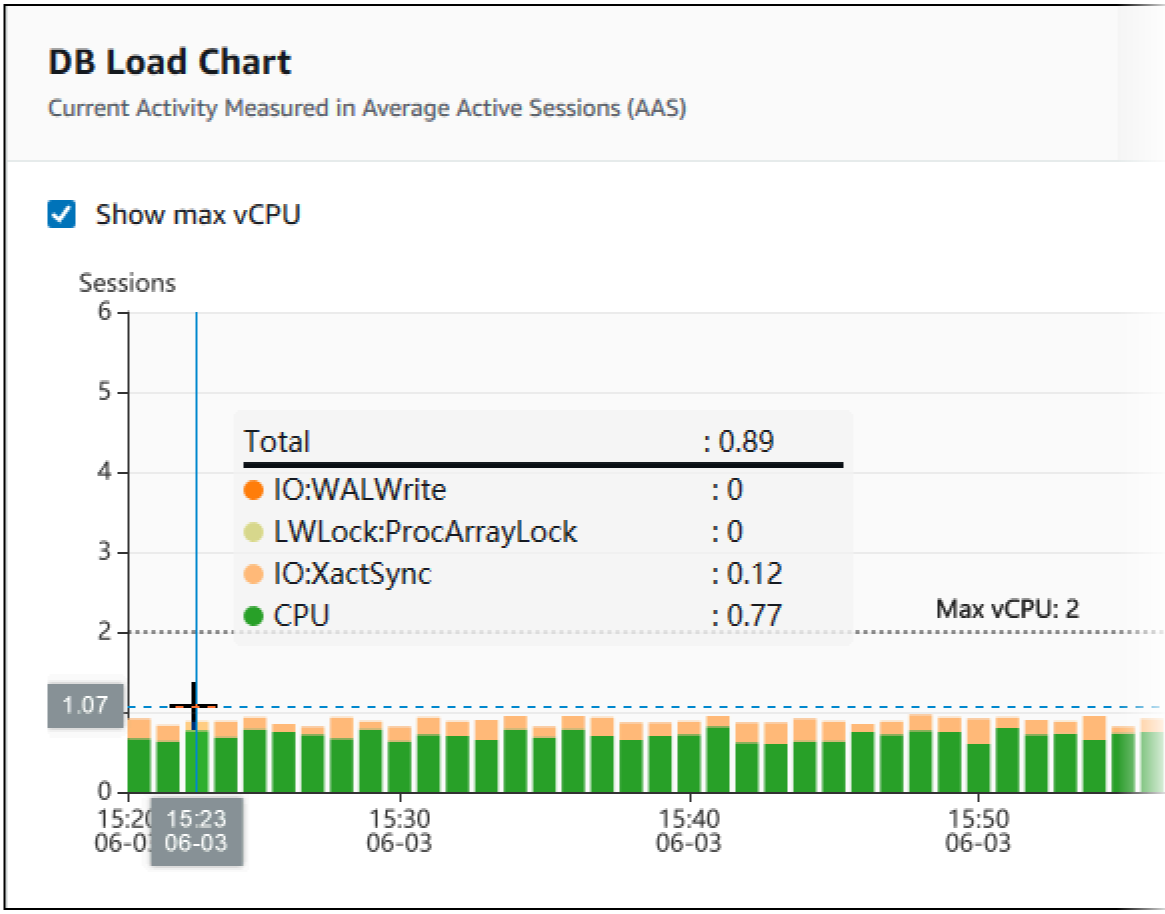


차원 항목에 대한 DB 로드 세부 정보

차원 내의 DB 로드 항목에 대한 세부 정보를 보려면 항목 이름 위로 마우스를 가져갑니다. 다음 이미지에서는 SQL 문의 세부 정보를 보여줍니다.

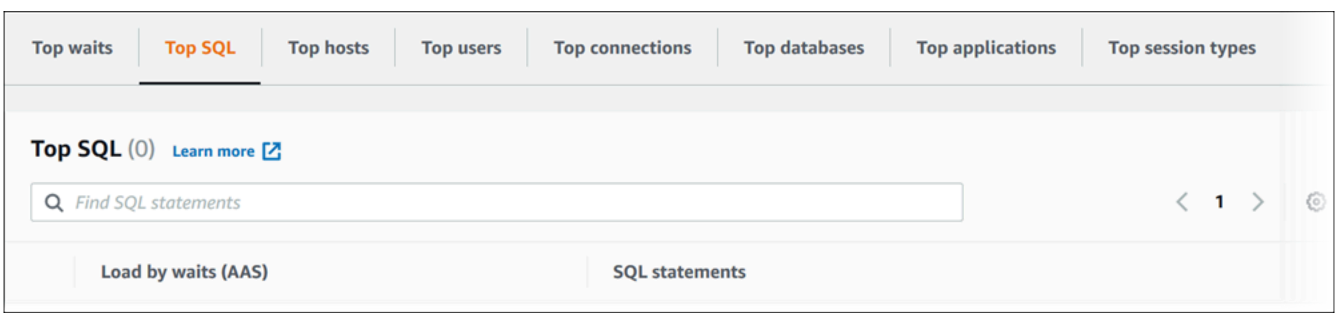


범례에서 선택한 기간의 항목에 대한 세부 정보를 보려면 해당 항목 위에 마우스 포인터를 놓습니다.



상위 측정기준 테이블

상위 측정기준 테이블은 DB 로드를 다른 차원으로 슬라이스합니다. 차원은 DB 로드의 다양한 특성에 대한 범주 또는 "분할 기준"입니다. 측정기준이 SQL인 경우 상위 SQL(Top SQL)에서는 DB 로드에 가장 많이 기여하는 SQL 문을 보여줍니다.



다음 차원 탭 중 하나를 선택합니다.

탭	설명	지원되는 엔진
상위 SQL	현재 실행 중인 SQL 문	모두
상위 대기(Top waits)	데이터베이스 백엔드가 대기 중인 이벤트	모두
상위 호스트(Top hosts)	연결된 클라이언트의 호스트 이름	모두
상위 사용자(Top users)	데이터베이스에 로그인한 사용자	모두
상위 애플리케이션	데이터베이스에 연결된 애플리케이션의 이름	Aurora PostgreSQL 해당
상위 세션 유형(Top session types)	현재 세션의 유형	Aurora PostgreSQL만

상위 SQL(Top SQL) 탭을 사용하여 쿼리를 분석하는 방법을 알아보려면 상위 SQL(Top SQL) 탭 개요 섹션을 참조하세요.

성능 개선 도우미 대시보드 액세스

Amazon RDS는 성능 개선 도우미 대시보드에서 성능 개선 도우미 및 CloudWatch 지표에 대한 통합 보기를 제공합니다.

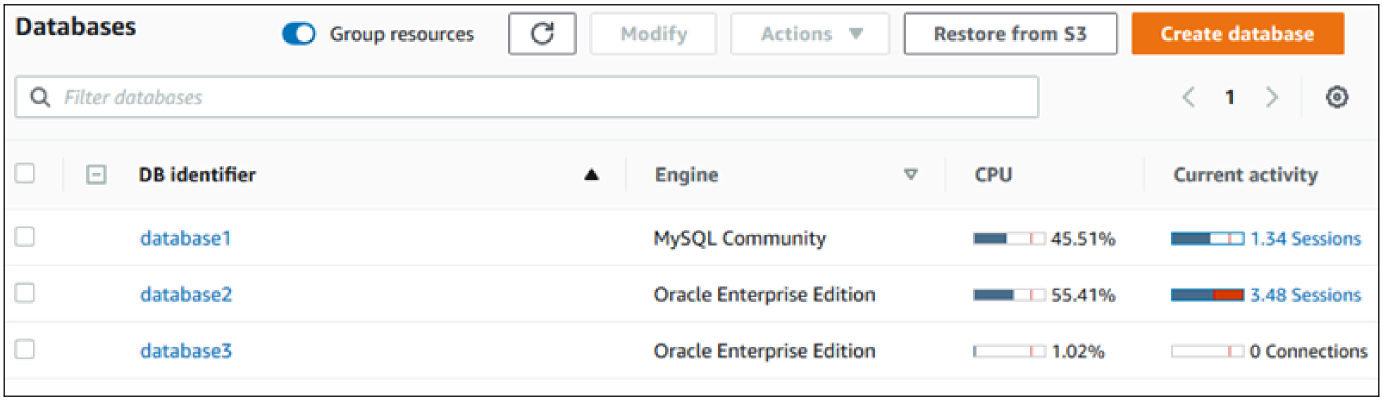
성능 개선 도우미 대시보드에 액세스하려면 다음 절차를 따르세요.

AWS 관리 콘솔에서 성능 개선 도우미 대시보드를 보려면

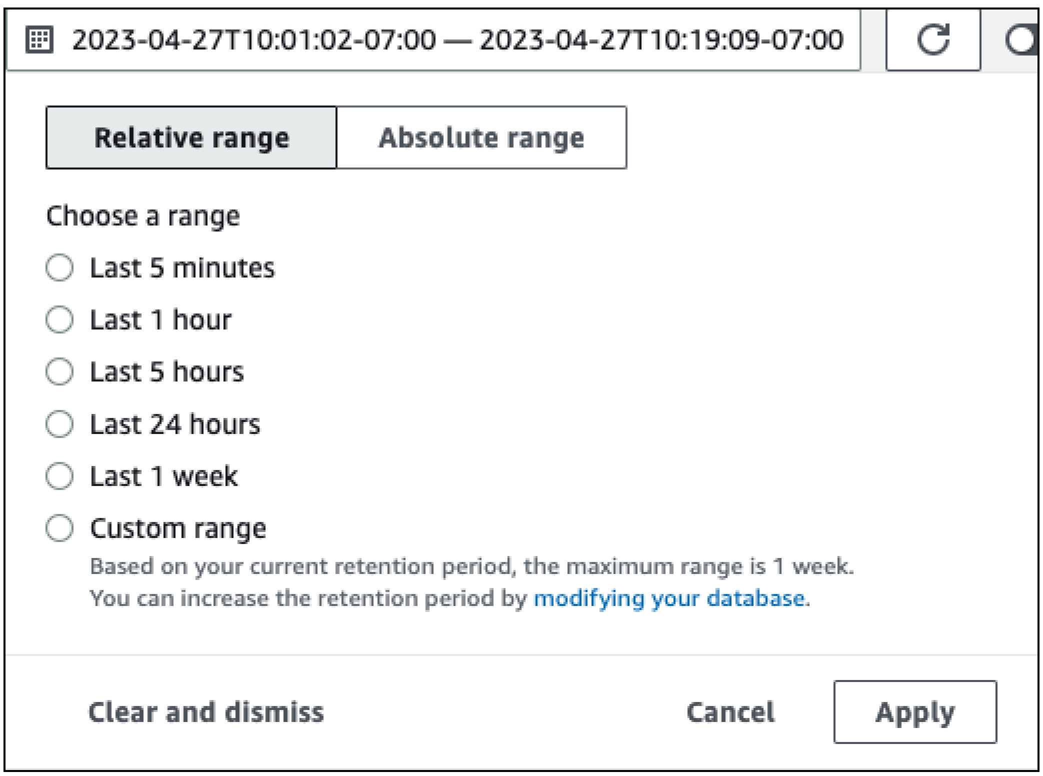
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

   성능 개선 도우미가 켜진 DB 인스턴스의 경우, DB 인스턴스 목록에서 세션 항목을 선택하여 성능 개선 도우미 대시보드에 액세스할 수도 있습니다. 현재 활동에서 세션 항목은 지난 5분 동안 평균 활동 세션의 데이터베이스 로드를 보여 줍니다. 로드가 막대 모양으로 표시됩니다. 막대가 비어 있으면 DB 인스턴스가 유휴 상태입니다. 로드가 증가하면 막대가 파란색으로 채워집니다. 로드에서

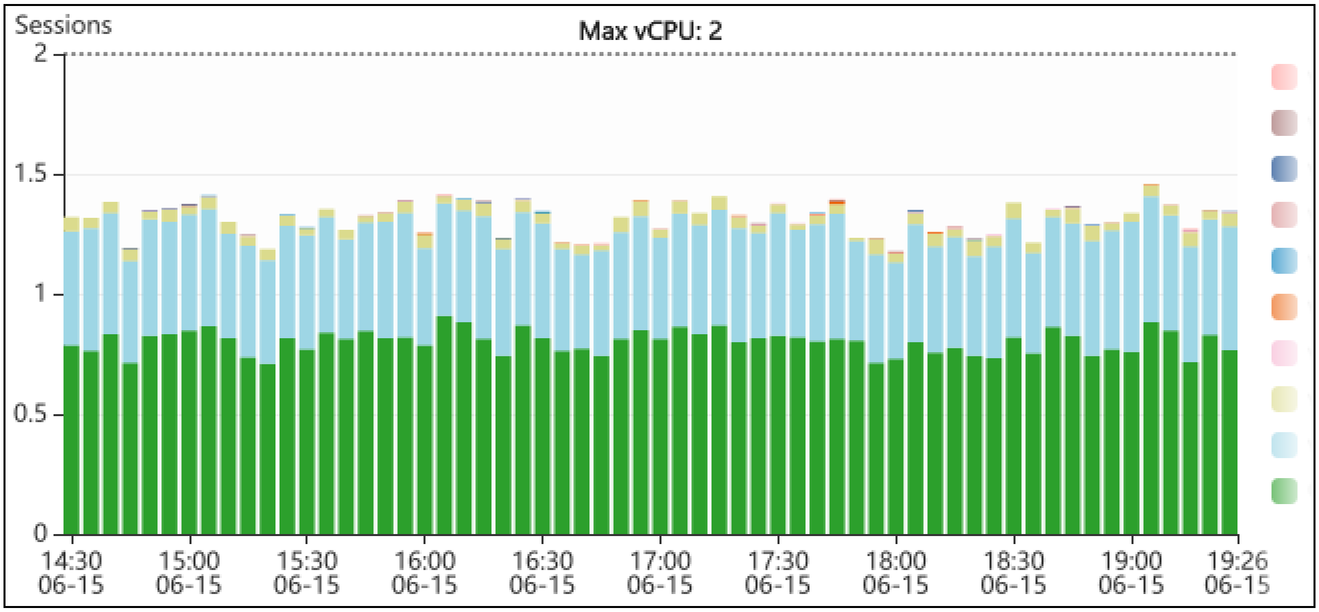
DB 인스턴스 클래스의 가상 CPU(vCPU) 수를 전달하면 막대가 빨간색으로 바뀌고 병목 가능성을 나타냅니다.



4. (선택 사항) 오른쪽 상단의 날짜 또는 시간 범위를 선택하고 다른 상대 또는 절대 시간 간격을 지정합니다. 이제 기간을 지정하고 데이터베이스 성능 분석 보고서를 생성할 수 있습니다. 보고서는 식별된 인사이트와 권장 사항을 제공합니다. 자세한 내용은 성능 개선 도우미에서 성과 분석 보고서 생성 단원을 참조하십시오.

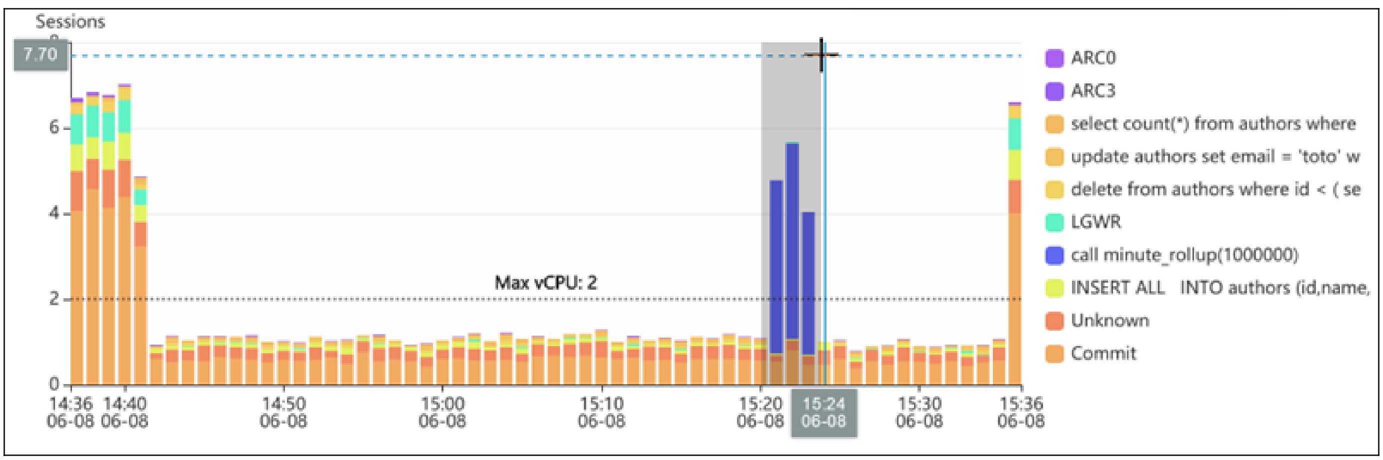


다음 스크린샷에서 DB 부하 간격은 5시간입니다.

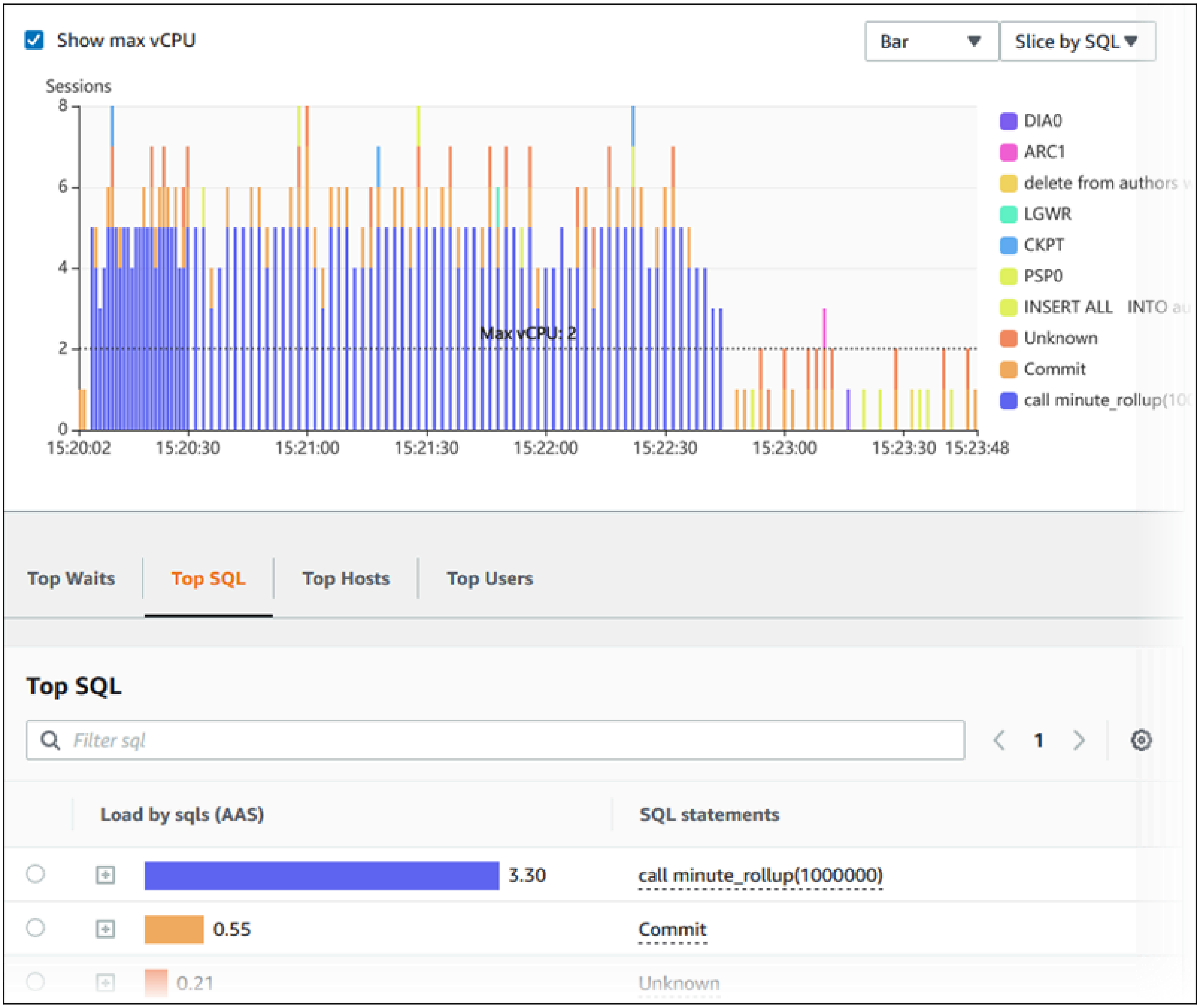


5. (선택 사항) DB 로드 차트의 한 부분을 확대하려면 시작 시간을 선택하고 원하는 기간의 끝까지 끌어옵니다.

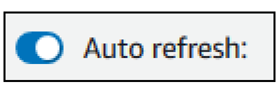
   선택한 영역이 DB 로드 차트에서 강조 표시됩니다.



   마우스를 놓으면 DB 로드드 차트의 선택한 AWS 리전이 확대되고 상위 차원(Top dimensions) 테이블이 다시 계산됩니다.



6. (선택 사항) 데이터를 자동으로 새로 고치려면 자동 새로 고침을 선택합니다.



성능 개선 도우미 대시보드는 새 데이터로 자동으로 고쳐집니다. 새로 고침 속도는 표시되는 데이터의 양에 따라 다릅니다:

- 5분은 10초마다 새로 고침됩니다.
- 1시간은 5분마다 새로 고침됩니다.
- 5시간은 5분마다 새로 고침됩니다.
- 24시간은 30분마다 새로 고침됩니다.
- 1주는 매일 새로 고침됩니다.

- 1개월은 매일 새로 고침됩니다.

대기 이벤트별 DB 로드 분석

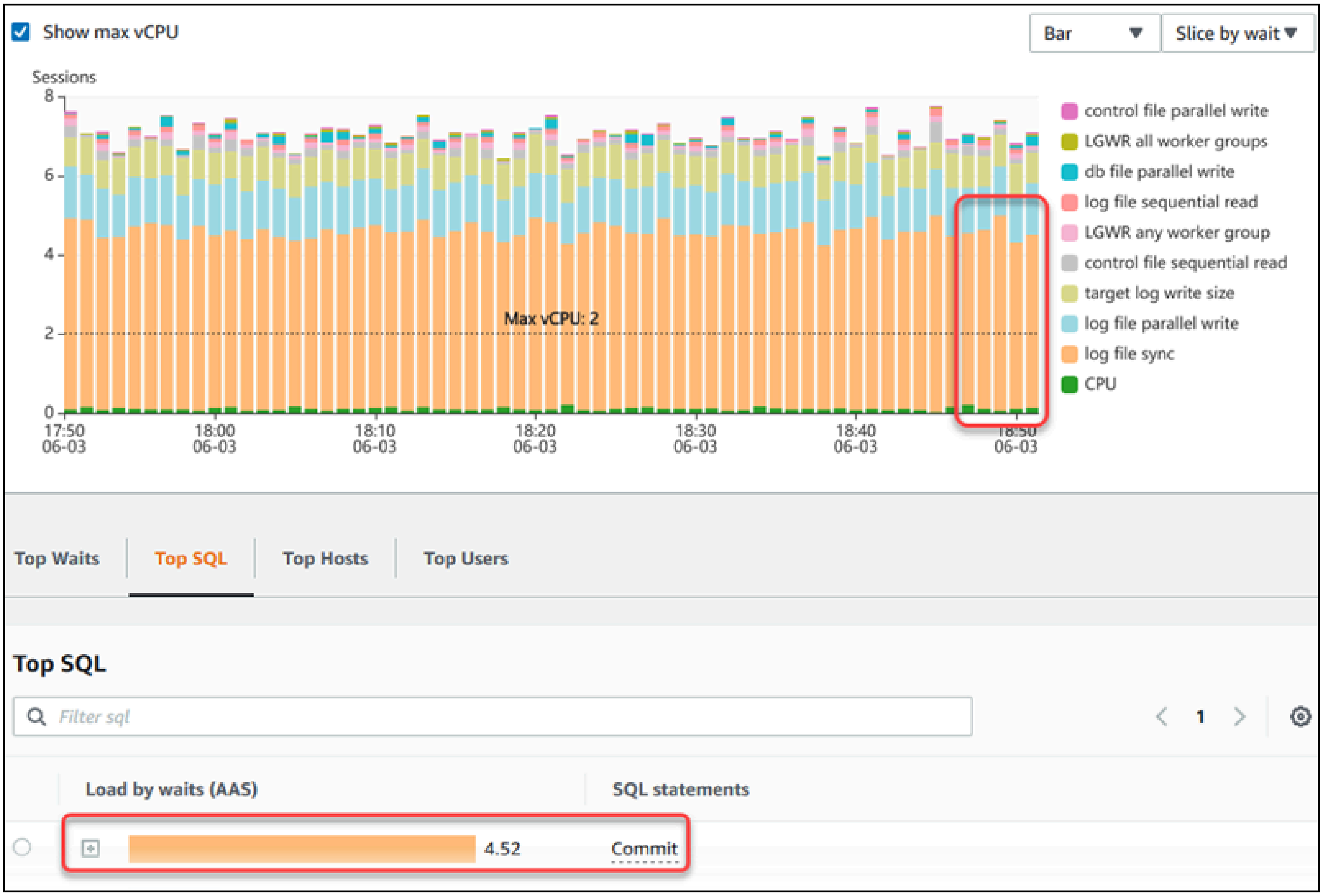
데이터베이스 로드(Database load) 차트가 병목 현상을 보일 때는 로드가 발생하는 위치를 찾아낼 수 있습니다. 이렇게 하려면 데이터베이스 로드(Database load) 차트 아래의 상위 로드 항목 테이블을 살펴보세요. SQL 쿼리나 사용자 같은 특정 항목을 선택하여 드릴다운을 통해 세부 정보까지 확인할 수 있습니다.

대기 상태와 상위 SQL 쿼리를 기준으로 구분된 DB 부하가 기본 Performance Insights 대시보드 보기입니다. 이 보기는 일반적으로 성능 문제를 가장 정확하게 파악할 수 있는 조합입니다. 대기 상태를 기준으로 구분된 DB 부하는 데이터베이스의 리소스 또는 동시성 병목 현상 유무를 표시합니다. 이 경우 상위 항목 테이블의 [SQL] 부하를 야기하는 쿼리를 표시합니다.

성능 문제를 진단하는 일반 워크플로우는 다음과 같습니다:

1. 데이터베이스 로드 차트를 보면서 데이터베이스 로드가 최대 CPU 선을 상회하는지 확인합니다.
2. 상회하는 경우가 있으면 데이터베이스 로드(Database load) 차트를 보면서 원인이 되는 대기 상태를 식별합니다.
3. 상위 부하 항목 테이블의 [SQL] 탭에서 어떤 쿼리가 대기 상태에 가장 큰 영향을 미치는지 모니터링하면서 부하를 야기하는 요약 쿼리를 식별합니다. DB Load by Wait(대기별 DB 로드) 열을 보면 이러한 요약 쿼리를 식별할 수 있습니다.
4. [SQL] 탭에서 요약 쿼리 중 하나를 선택하여 확장한 다음 구성하고 있는 하위 쿼리를 확인합니다.

예를 들어 다음 대시보드에서 로그 파일 동기화 대기 시간은 대부분의 DB 부하를 차지합니다. LGWR 모든 작업자 그룹 대기 시간도 높습니다. 상위 SQL(Top SQL) 차트는 로그 파일 동기화 대기의 원인인 자주 사용된 `COMMIT` 문을 보여줍니다. 이 경우 커밋 빈도를 줄이면 DB 부하가 줄어듭니다.



일정 기간 동안의 데이터베이스 성능 분석

일정 기간에 대한 성능 분석 보고서를 생성하여 온디맨드 분석을 통해 데이터베이스 성능을 분석합니다. 성능 분석 보고서를 확인하여 리소스 병목 현상 또는 DB 인스턴스의 쿼리 변경과 같은 성능 문제를 찾아낼 수 있습니다. 성능 개선 도우미 대시보드를 사용하면 기간을 선택하고 성능 분석 보고서를 생성할 수 있습니다. 보고서에 태그를 하나 이상 추가할 수도 있습니다.

이 기능을 사용하려면 유료 등급의 보존 기간을 사용해야 합니다. 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 단원을 참조하세요.

보고서는 성능 분석 보고서 - 신규 탭에서 선택하여 볼 수 있습니다. 보고서에는 인사이트, 관련 지표, 성능 문제 해결을 위한 권장 사항이 포함되어 있습니다. 보고서는 성능 개선 도우미 보존 기간 동안 볼 수 있습니다.

보고서 분석 기간의 시작 시간이 보존 기간을 벗어나면 보고서가 삭제됩니다. 보존 기간이 끝나기 전에 보고서를 삭제할 수도 있습니다.

DB 인스턴스에 대한 성능 문제를 감지하고 분석 보고서를 생성하려면 성능 개선 도우미를 켜야 합니다. 성능 개선 도우미 켜기에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 섹션을 참조하세요.

이 기능에 대한 리전, DB 엔진 및 인스턴스 클래스 지원 정보는 성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원 섹션을 참조하세요.

다음 섹션에서 성능 분석 보고서를 만들고, 보고, 태그를 추가하고, 삭제할 수 있습니다.

주제

성능 개선 도우미에서 성과 분석 보고서 생성

성능 개선 도우미 대시보드에서 특정 기간에 대한 성과 분석 보고서를 만들 수 있습니다. 기간을 선택하고 하나 이상의 태그를 분석 보고서에 추가할 수 있습니다.

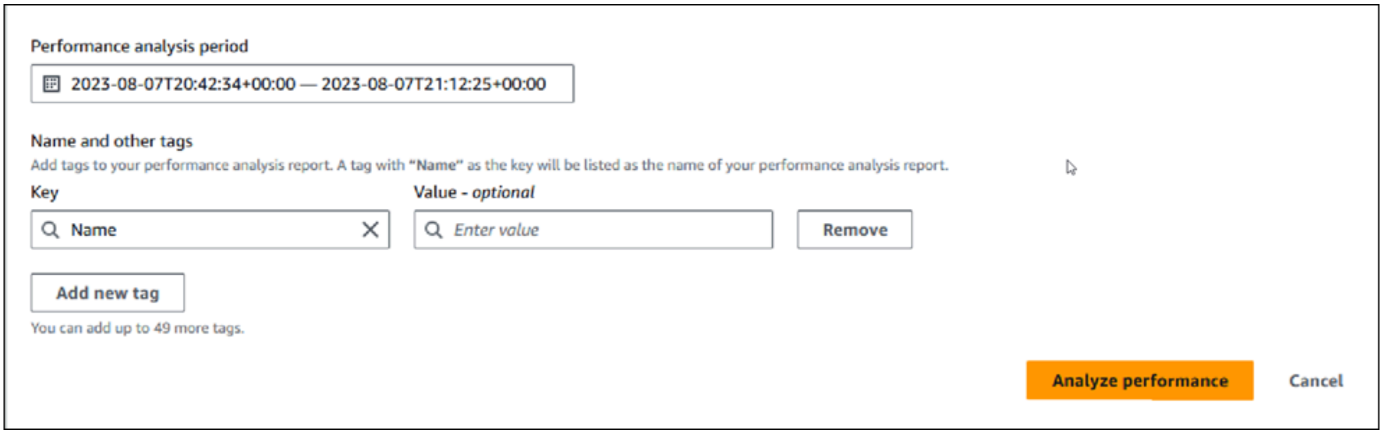
분석 기간은 5분에서 6일 사이로 선택할 수 있습니다. 분석 시작 시간 전에 최소 24시간의 성능 데이터가 있어야 합니다.

이 기능에 대한 리전, DB 엔진 및 인스턴스 클래스 지원 정보는 성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원 섹션을 참조하세요.

특정 기간에 대한 성과 분석 보고서를 생성하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 성능 개선 도우미 대시보드의 데이터베이스 로드 섹션에서 성과 분석을 선택합니다.

   기간을 설정하고 성능 분석 보고서에 하나 이상의 태그를 추가하는 필드가 표시됩니다.

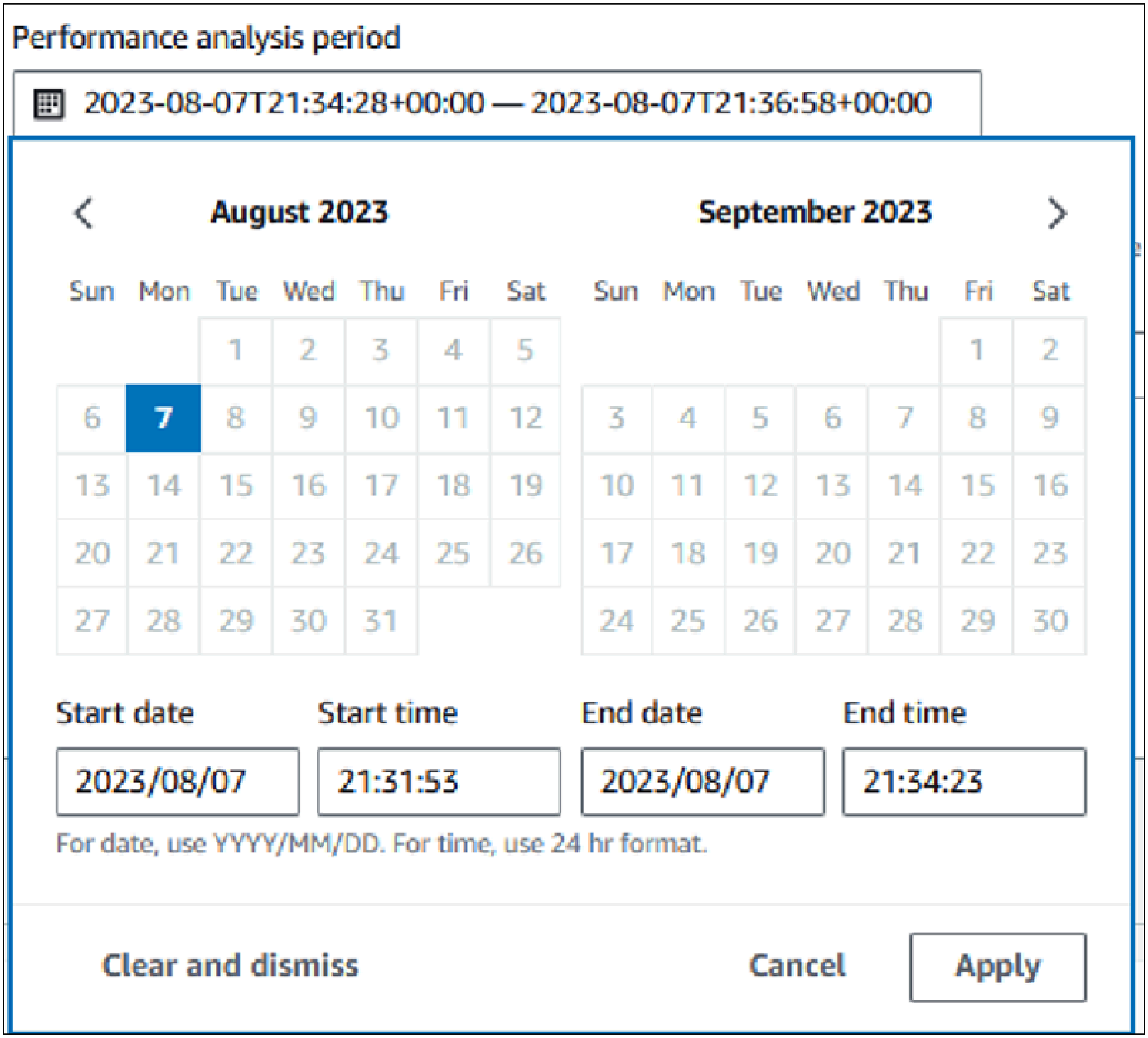


5. 기간을 선택합니다. 오른쪽 상단의 상대 범위 또는절대 범위에서 기간을 설정하면 이 기간 내의 분석 보고서 날짜 및 시간만 입력하거나 선택할 수 있습니다. 이 기간을 벗어나도록 분석 기간을 선택하면 오류 메시지가 표시됩니다.

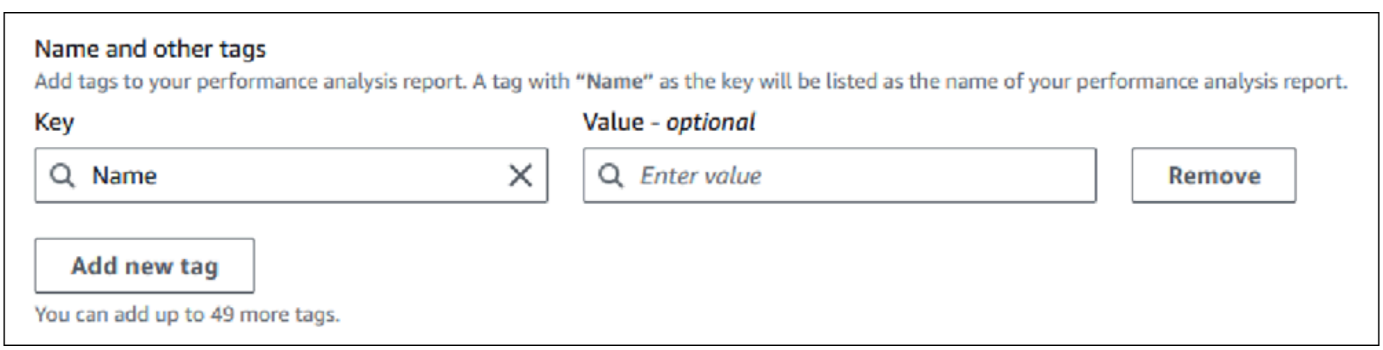
   기간을 설정하려면 다음 중 하나를 수행할 수 있습니다.

   - DB 로드 차트의 슬라이더 중 하나를 눌러 드래그합니다.

     성과 분석 기간 상자에 선택한 기간이 표시되고 DB 로드 차트에 선택한 기간이 강조 표시됩니다.
   - 성과 분석 기간 상자에서 시작 날짜, 시작 시간, 종료 날짜, 종료 시간을 선택합니다.



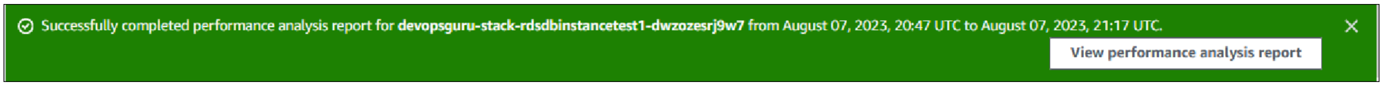
6. (선택 사항) 키와 값-선택 사항을 입력하여 보고서에 태그를 추가합니다.



7. 성과 분석을 선택합니다.

    배너에 보고서 생성에 성공했는지, 실패했는지 나타내는 메시지가 표시됩니다. 이 메시지에는 보고서를 볼 수 있는 링크도 제공됩니다.

    다음 예시에서는 보고서 생성 성공 메시지가 포함된 배너를 보여줍니다.

    보고서는 성능 분석 보고서 - 신규 탭에서 볼 수 있습니다.

AWS CLI를 사용하여 성과 분석 보고서를 생성할 수 있습니다. AWS CLI를 사용하여 보고서를 생성하는 방법에 대한 예는 특정 기간에 대한 성과 분석 보고서 생성 섹션을 참조하세요.

성능 개선 도우미에서 성과 분석 보고서 보기

성능 분석 보고서 - 신규 탭에 DB 인스턴스에 대해 생성된 모든 보고서가 나열됩니다. 각 보고서에는 다음 내용이 표시됩니다.

- ID: 보고서의 고유 식별자입니다.
- 이름: 보고서에 추가된 태그 키입니다.
- 보고서 생성 시간: 보고서를 생성한 시간입니다.
- 분석 시작 시간: 보고서의 분석 시작 시간입니다.
- 분석 종료 시간: 보고서의 분석 종료 시간입니다.

성능 분석 보고서를 보는 방법

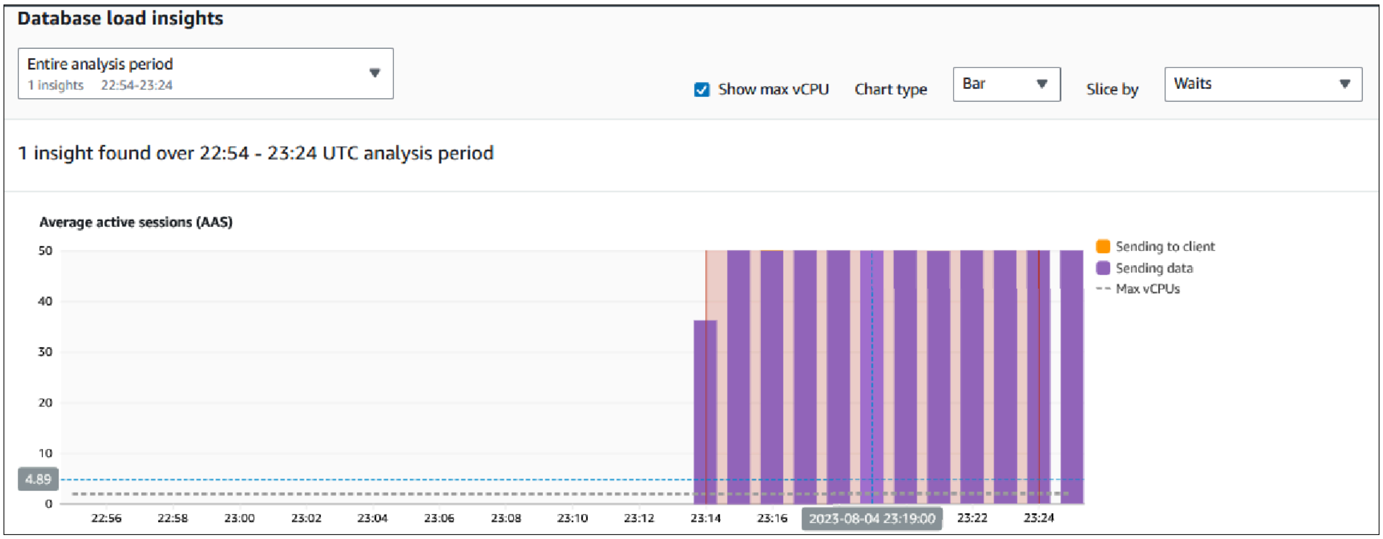
1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. 분석 보고서를 보려는 DB 인스턴스를 선택합니다.
4. 성능 개선 도우미 대시보드에서 아래로 스크롤하여 성능 분석 보고서 - 신규 탭을 선택합니다.

    서로 다른 기간에 대한 모든 분석 보고서가 표시됩니다.
5. 보려는 보고서의 ID를 선택합니다.

하나 이상의 인사이트가 식별된 경우 DB 로드 차트에는 기본적으로 전체 분석 기간이 표시됩니다. 보고서에서 하나의 인사이트를 식별한 경우 DB 로드 차트에는 기본적으로 해당 인사이트가 표시됩니다.

대시보드에는 태그 섹션에 보고서의 태그도 나열되어 있습니다.

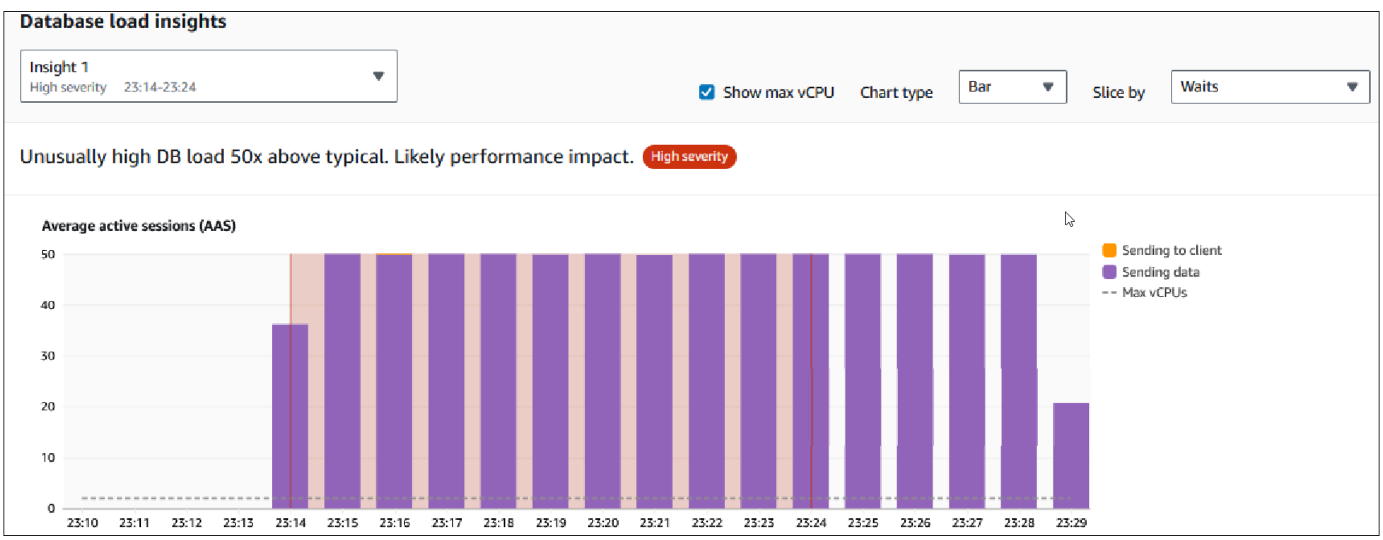
다음 예시는 보고서의 전체 분석 기간을 보여줍니다.

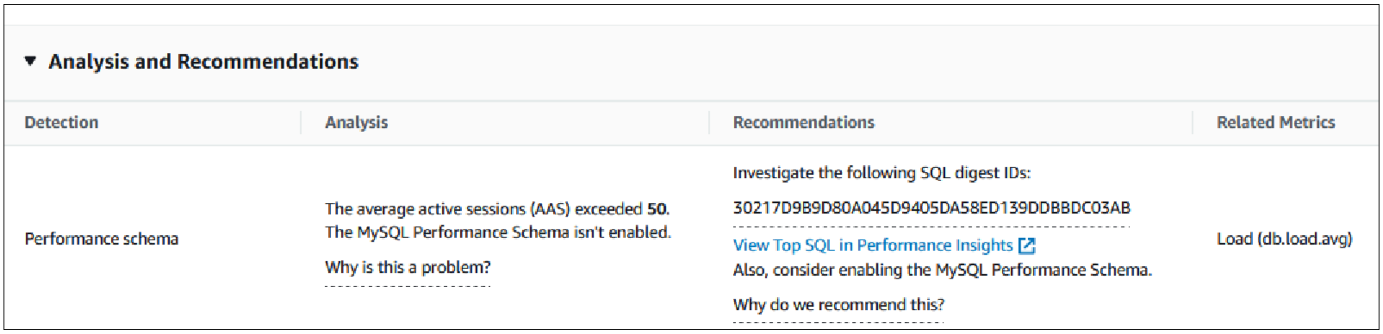


6. 보고서에 인사이트가 하나 이상 식별된 경우 데이터베이스 로드 인사이트 목록에서 보려는 인사이트를 선택하세요.

   대시보드에는 인사이트 메시지, 인사이트 기간이 강조 표시된 DB 로드 차트, 분석 및 권장 사항, 보고서 태그 목록이 표시됩니다.

   다음 예시는 보고서의 DB 로드 인사이트를 보여줍니다.





성능 개선 도우미에서 성과 분석 보고서에 태그 추가

보고서를 만들거나 볼 때 태그를 추가할 수 있습니다. 보고서에는 태그의 수는 최대 50개입니다.

태그를 추가하려면 권한이 있어야 합니다. 성능 개선 도우미의 액세스 정책에 대한 자세한 내용은 Performance Insights에 대한 액세스 정책 구성 섹션을 참조하세요.

보고서를 만드는 동안 하나 이상의 태그를 추가하려면 성능 개선 도우미에서 성과 분석 보고서 생성 프로시저의 6단계를 참조하세요.

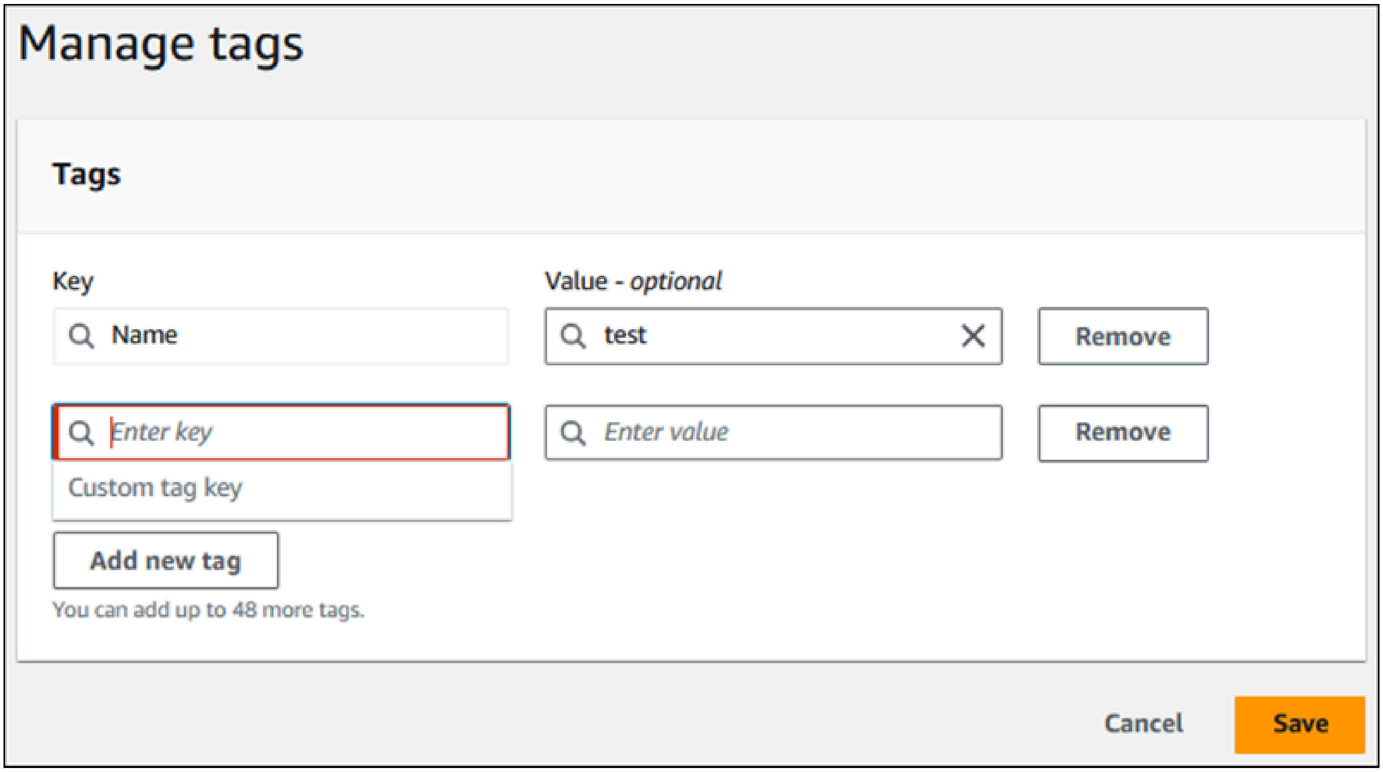
보고서를 볼 때 하나 이상의 태그를 추가하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

   해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 아래로 스크롤하여 성능 분석 보고서 - 신규 탭을 선택합니다.
5. 태그를 추가할 보고서를 선택합니다.

   대시보드에 보고서가 표시됩니다.
6. 태그로 스크롤하여 태그 관리를 선택합니다.
7. 새 태그 추가를 선택합니다.
8. 키와 값 - 선택 사항을 입력하고 새 태그 추가를 선택합니다.

   다음 예시에서는 선택한 보고서에 새 태그를 추가하는 옵션을 보여줍니다.



보고서에 새 태그가 생성됩니다.

보고서의 태그 목록이 대시보드의 태그 섹션에 표시됩니다. 보고서에서 태그를 제거하려면 태그 옆의 제거를 선택합니다.

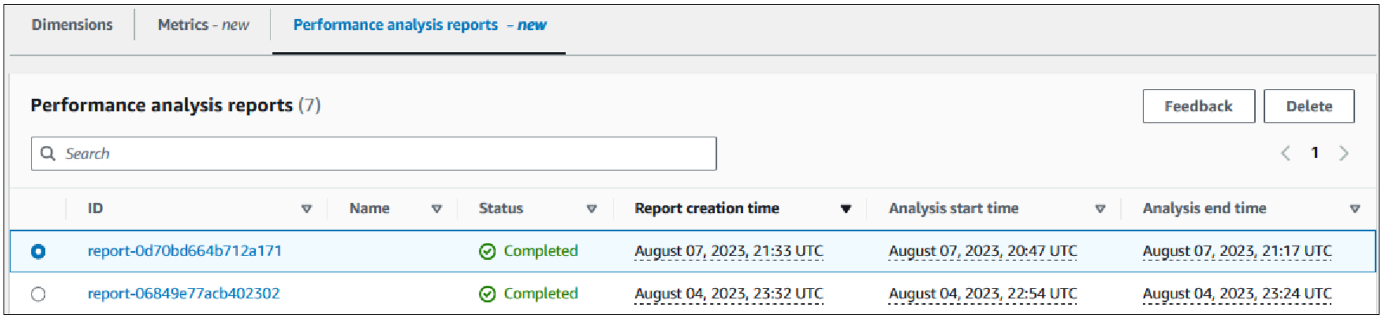
성능 개선 도우미에서 성과 분석 보고서 삭제

성능 분석 보고서 탭에 표시된 보고서 목록에서 보고서를 삭제하거나 보고서를 보는 동안 삭제할 수 있습니다.

보고서를 삭제하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. DB 인스턴스를 선택합니다.

   해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 아래로 스크롤하여 성능 분석 보고서 - 신규 탭을 선택합니다.
5. 삭제하려는 보고서를 선택하고 오른쪽 상단의 삭제를 선택합니다.



확인 창이 표시됩니다. 확인을 선택하면 보고서가 삭제됩니다.

6. (선택 사항) 삭제하려는 보고서의 ID를 선택합니다.

   보고서 페이지의 오른쪽 상단에 있는 삭제를 선택합니다.

   확인 창이 표시됩니다. 확인을 선택하면 보고서가 삭제됩니다.

성능 개선 도우미의 상위 SQL 탭을 사용하여 쿼리 분석

Amazon RDS 성능 개선 도우미 대시보드에서 상위 차원(Top dimensions) 테이블의 상위 SQL(Top SQL) 탭에서 실행 중인 쿼리 및 최근 쿼리에 대한 정보를 확인할 수 있습니다. 이 정보를 사용하여 쿼리를 튜닝할 수 있습니다.

주제

- 상위 SQL(Top SQL) 탭 개요
- 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트에 액세스
- 성능 개선 도우미 대시보드에서 SQL 통계 보기

상위 SQL(Top SQL) 탭 개요

기본적으로 상위 SQL 탭은 DB 로드에 가장 많은 영향을 미치는 25개의 쿼리를 보여줍니다. 쿼리를 조정하는 데 도움이 되도록 쿼리 텍스트 및 SQL 통계와 같은 정보를 분석할 수 있습니다. 상위 SQL(Top SQL) 탭에 표시될 통계를 선택할 수도 있습니다.

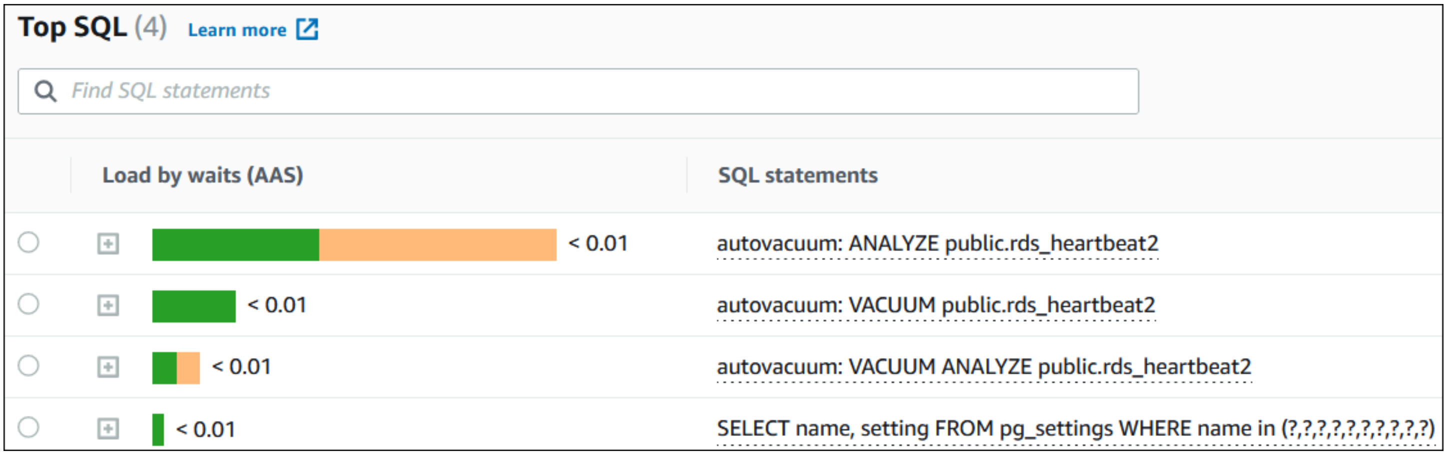
주제

- SQL 텍스트
- SQL 통계
- 대기별 로드(AAS)
- SQL 정보 보기

- [통계 기본 설정 선택]

SQL 텍스트

기본적으로 상위 SQL 테이블의 각 행에는 각 문에 대해 500바이트의 텍스트가 표시됩니다.

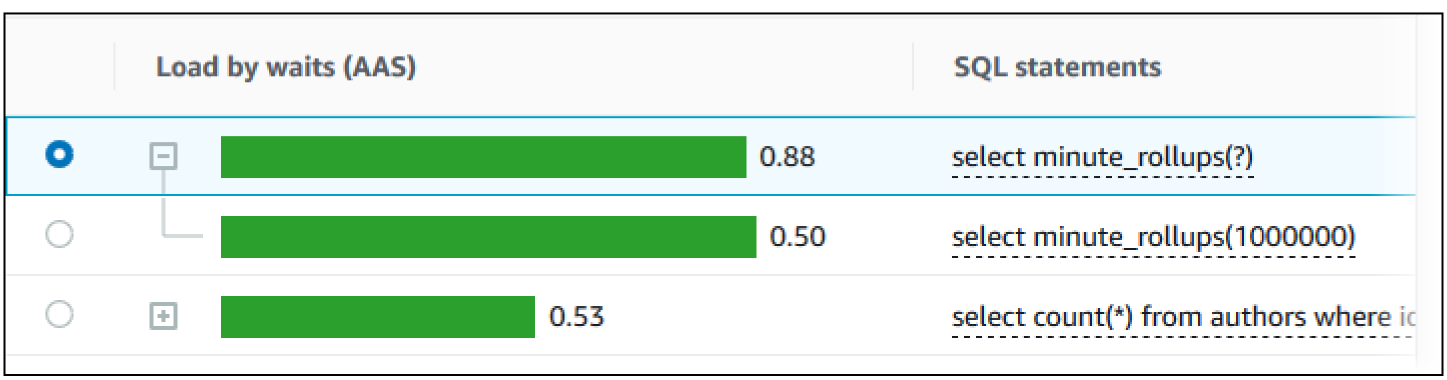


기본 500바이트 이상의 SQL 텍스트를 보는 방법에 대한 자세한 내용은 [성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트에 액세스] 섹션을 참조하세요.

SQL 다이제스트(SQL digest)는 구조적으로 유사하지만 리터럴 값이 다를 수 있는 여러 실제 쿼리의 조합입니다. 다이제스트는 하드 코딩된 값을 물음표로 바꿉니다. 예를 들어, 다이제스트는 `SELECT * FROM emp WHERE lname= ?`가 될 수 있습니다. 이 다이제스트에는 다음 하위 쿼리가 포함될 수 있습니다.

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

다이제스트에서 리터럴 SQL 문을 보려면 쿼리를 선택한 다음 더하기 기호 (+) 를 선택합니다. 다음 예에서 선택한 쿼리는 다이제스트입니다.

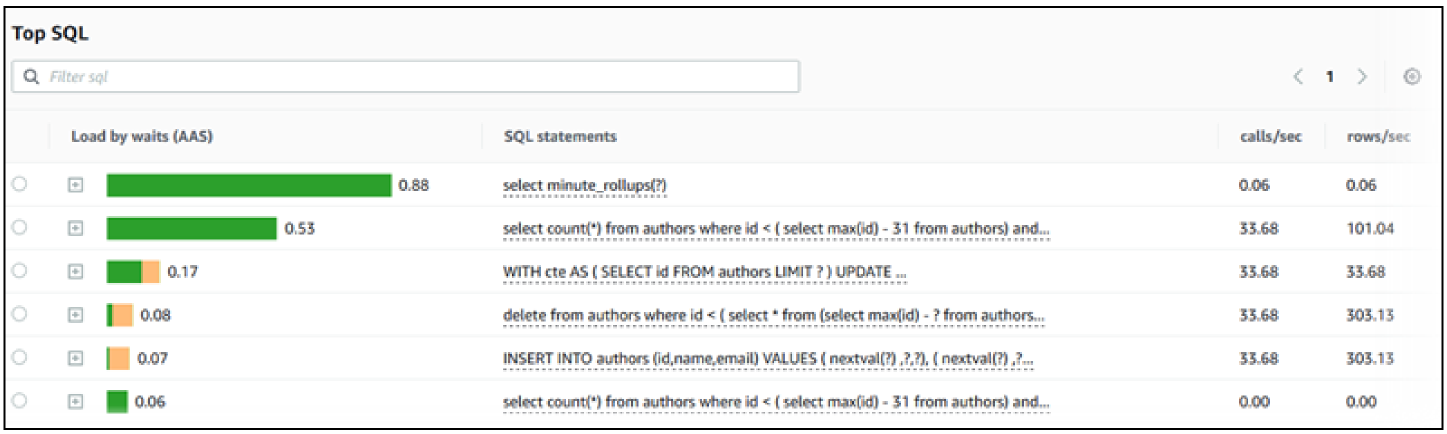


> **Note**
>
> SQL 다이제스트는 유사한 SQL 문을 그룹화하지만, 민감한 정보는 삭제하지 않습니다.

SQL 통계

SQL 통계(SQL statistics)는 SQL 쿼리에 대한 성능 관련 지표입니다. 예를 들어 성능 개선 도우미는 초당 실행 횟수 또는 초당 처리된 행을 표시할 수 있습니다. 성능 개선 도우미는 가장 일반적인 쿼리에 대한 통계만 수집합니다. 일반적으로 이러한 쿼리는 성능 개선 도우미 대시보드에 표시된 부하별로 상위 쿼리와 일치합니다.

상위 SQL(Top SQL) 테이블의 모든 라인은 다음 예에 나온 것처럼 SQL 문 또는 다이제스트에 대한 관련 통계를 보여줍니다.



Top SQL

Load by waits (AAS)	SQL statements	calls/sec	rows/sec
0.88	select minute_rollups(?)	0.06	0.06
0.53	select count(*) from authors where id < (select max(id) - 31 from authors) and...	33.68	101.04
0.17	WITH cte AS (SELECT id FROM authors LIMIT ?) UPDATE ...	33.68	33.68
0.08	delete from authors where id < (select * from (select max(id) - ? from authors...	33.68	303.13
0.07	INSERT INTO authors (id,name,email) VALUES (nextval(?) ,?,?), (nextval(?) ,?...	33.68	303.13
0.06	select count(*) from authors where id < (select max(id) - 31 from authors) and...	0.00	0.00

성능 개선 도우미는 SQL 통계에 대해 `0.00` 및 `-`(알 수 없음)을 보고할 수 있습니다. 이 상황은 다음 조건에서 발생합니다.

- 샘플이 하나만 존재합니다. 예를 들어, 성능 개선 도우미는 `pg_stat_statements` 보기에서 여러 개의 샘플을 기반으로 Aurora PostgreSQL 쿼리의 변경 비율을 계산합니다. 워크로드가 짧은 시간 동안 실행되면 성능 개선 도우미에서 샘플을 하나만 수집할 수 있으며 이런 경우 변경 비율을 계산할 수 없습니다. 값을 알 수 없으므로 대시(`-`)로 표시됩니다.
- 두 샘플의 값이 같은 경우입니다. 변경이 발생하지 않았기 때문에 성능 개선 도우미가 변경 비율을 계산할 수 없으므로 비율을 `0.00`으로 보고합니다.
- Aurora PostgreSQL 문에 유효한 식별자가 없는 경우입니다. PostgreSQL은 구문 분석 및 분석 후에만 문에 대한 식별자를 만듭니다. 따라서 PostgreSQL 내부 인 메모리 구조에 식별자가 없는 상태로 문이 존재할 수 있습니다. 성능 개선 도우미는 초당 한 번 내부 인 메모리 구조를 샘플링하므로 하나의 샘플에 대해서만 대기 시간이 짧은 쿼리가 나타날 수 있습니다. 이 샘플에 대해 쿼리 식별자를 사

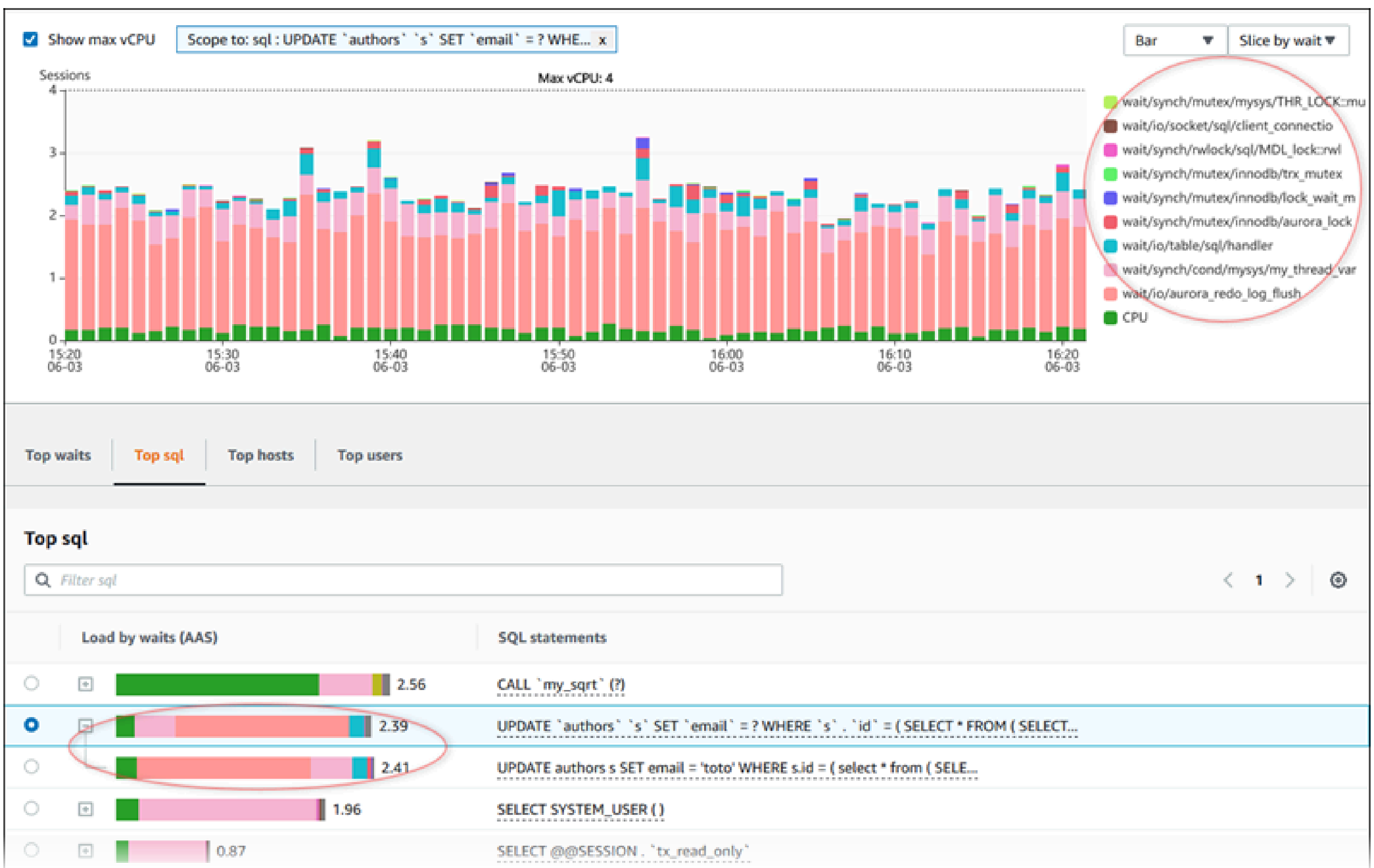
용할 수 없는 경우 성능 개선 도우미는 이 문을 통계와 연결할 수 없습니다. 값을 알 수 없으므로 대시(-)로 표시됩니다.

Aurora엔진의 SQL 통계에 대한 설명을 보려면 [성능 개선 도우미에 대한 SQL 통계](#) 섹션을 참조하세요.

대기별 로드(AAS)

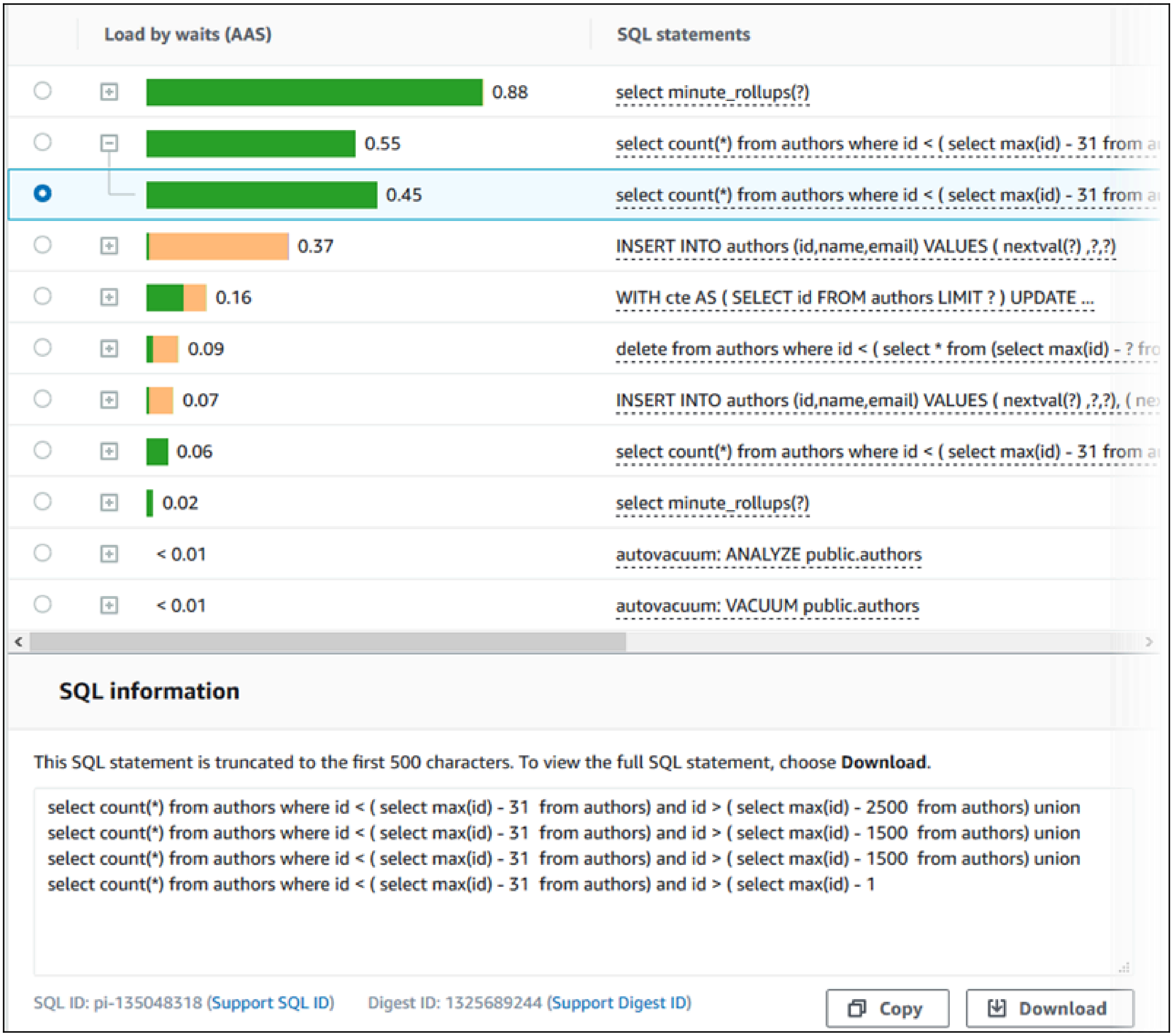
상위 SQL(Top SQL)에서 대기 시간별 로드(AAS)(Load by waits (AAS)) 열은 각 상위 로드 항목과 연결된 데이터베이스 로드의 비율을 나타냅니다. 이 열에는 현재 DB 부하 차트에서 어떤 그룹화 기준을 선택하든 그 기준에 따라 해당 항목의 부하가 반영됩니다. 평균 활성 세션(AAS)에 대한 자세한 내용은 [평균 활성 세션](#) 섹션을 참조하세요.

예를 들어 DB 로드(DB load) 차트를 대기 상태별로 그룹화할 수 있습니다. 상위 부하 항목 테이블에서 SQL 쿼리를 검사합니다. 이 경우 DB Load by Waits(대기별 DB 로드) 막대는 쿼리가 영향을 미치는 대기 상태의 정도를 크기, 세그먼트 및 컬러 코드로 표시합니다. 또한 선택한 쿼리에 영향을 미치는 대기 상태를 표시합니다.



SQL 정보 보기

상위 SQL(Top SQL) 테이블에서 명령문을 열어 해당 정보를 볼 수 있습니다. 맨 아래 창에 정보가 나타납니다.



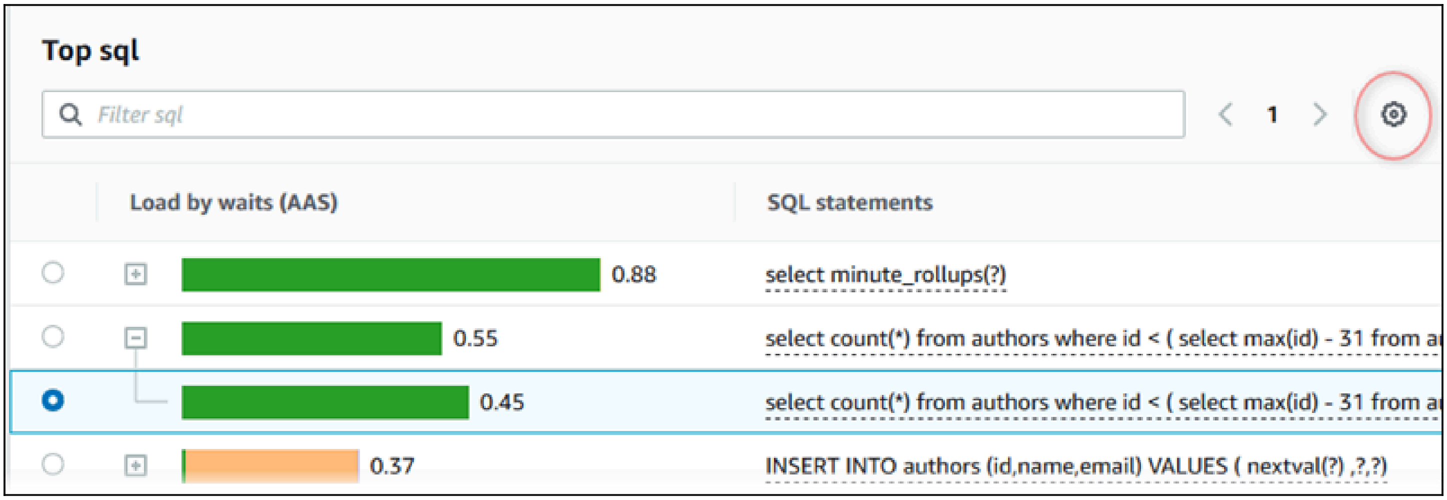
상위 SQL 탭에서 SQL 문과 연결된 다음과 같은 식별자(ID) 유형을 볼 수 있습니다.

- SQL ID 지원 – SQL ID의 해시 값입니다. 이 값은 AWS Support를 이용할 때 SQL ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 SQL ID 및 SQL 텍스트에 액세스할 수 없습니다.

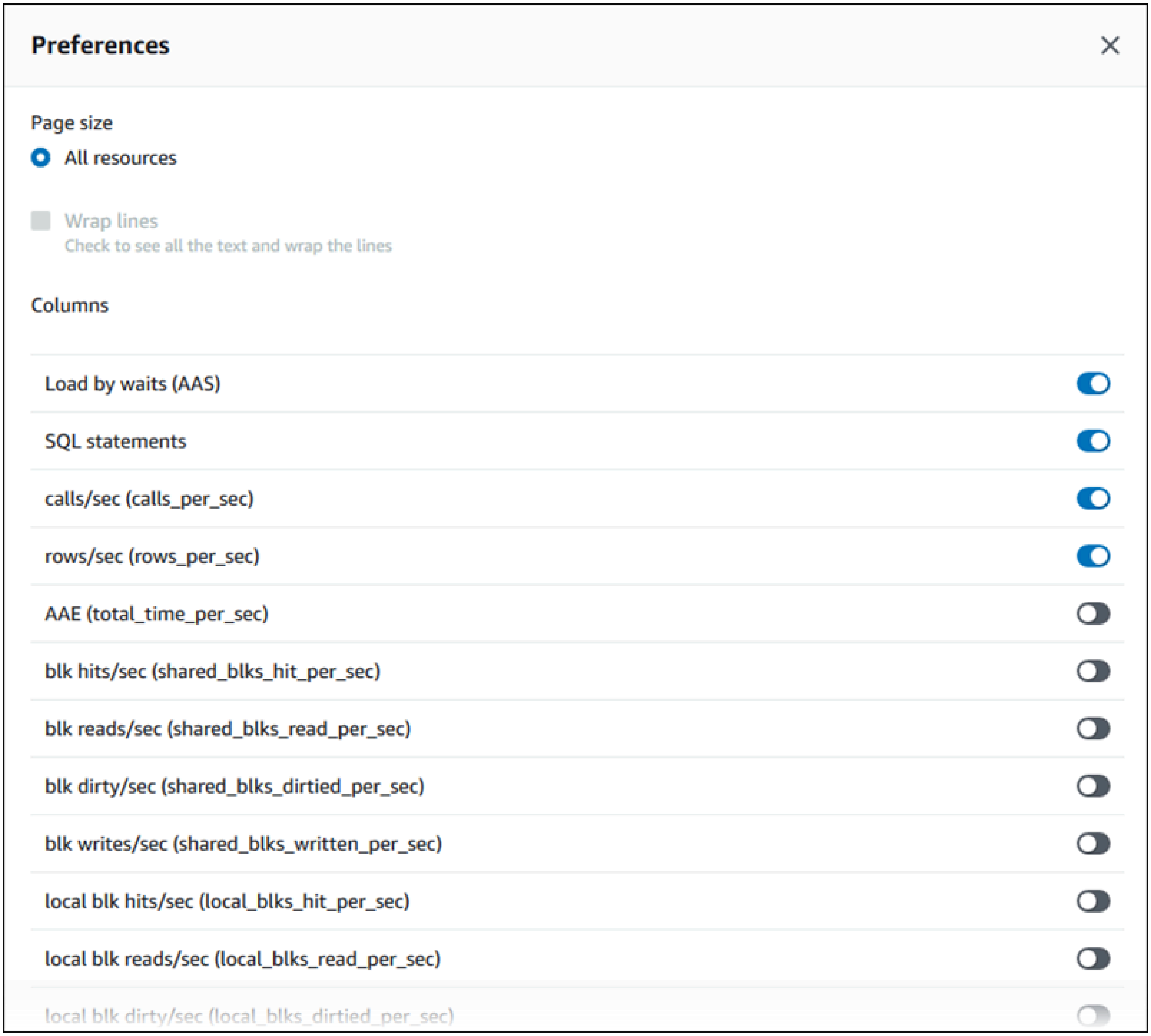
- Support Digest ID(다이제스트 ID 지원) – 다이제스트 ID의 해시 값입니다. 이 값은 AWS Support를 이용할 때 다이제스트 ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 다이제스트 ID 및 SQL 텍스트에 액세스할 수 있는 권한이 없습니다.

통계 기본 설정 선택

기본 설정(Preferences) 아이콘을 선택하여 상위 SQL(Top SQL) 탭에 표시되는 통계를 제어할 수 있습니다.



기본 설정 아이콘을 선택하면 기본 설정 창이 열립니다. 다음 스크린샷은 기본 설정 창의 예입니다.



상위 SQL(Top SQL) 탭에 표시하려는 통계를 활성화하고 마우스를 사용하여 창 아래쪽으로 스크롤한 다음 계속(Continue)을 선택합니다.

Aurora 엔진의 초당 또는 호출당 통계에 대한 자세한 내용은 성능 개선 도우미에 대한 SQL 통계의 엔진별 SQL 통계 섹션을 참조하십시오.

성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트에 액세스

기본적으로 상위 SQL(Top SQL) 테이블의 각 행에는 각 SQL 문에 대해 500바이트의 SQL 텍스트가 표시됩니다.

SQL 문이 500바이트를 초과하면 상위 SQL 테이블 아래의 SQL 텍스트 섹션에서 더 많은 텍스트를 볼 수 있습니다. 이 경우 SQL 텍스트에 표시된 쿼리의 최대 길이는 4KB입니다. 이 제한은 콘솔에 의해 도입되며 데이터베이스 엔진에서 설정한 제한에 따라 달라집니다. SQL 텍스트에 표시된 텍스트를 저장하려면 다운로드를 선택합니다.

주제

- Aurora MySQL 텍스트 크기 제한
- Aurora PostgreSQL DB 인스턴스에 대한 SQL 텍스트 한도 설정
- 성능 개선 도우미 대시보드에서 SQL 텍스트 보기 및 다운로드

Aurora MySQL 텍스트 크기 제한

SQL 텍스트를 다운로드할 때 데이터베이스 엔진은 최대 길이를 결정합니다. 다음의 엔진별 제한에 해당하는 SQL 텍스트를 다운로드할 수 있습니다.

DB 엔진	다운로드한 텍스트의 최대 길이
Aurora MySQL	길이는 4,096바이트로 고정됩니다.

성능 개선 도우미 콘솔의 SQL 텍스트 섹션은 엔진이 반환하는 최대값까지 표시합니다. 예를 들어, Aurora MySQL은 최대 1KB를 성능 개선 도우미에 반환하며, 원래 쿼리가 더 큰 경우에도 1KB만 수집하고 표시할 수 있습니다. 따라서 SQL 텍스트의 쿼리를 보거나 다운로드하면 성능 개선 도우미가 동일한 바이트 수를 반환합니다.

AWS CLI 또는 API, 성능 개선 도우미에는 콘솔이 적용하는 4KB 제한이 없으며, `DescribeDimensionKeys` 및 `GetResourceMetrics`로 최대 500바이트를 반환합니다.

> **Note**
>
> `GetDimensionKeyDetails`는 전체 쿼리를 반환하지만 크기에는 엔진 제한이 적용됩니다.

Aurora PostgreSQL DB 인스턴스에 대한 SQL 텍스트 한도 설정

Aurora PostgreSQL은 텍스트를 다르게 처리합니다. DB 인스턴스 파라미터 `track_activity_query_size`를 사용하여 텍스트 크기 제한을 설정할 수 있습니다. 이 파라미터에는 다음과 같은 특성이 있습니다.

기본 텍스트 크기

Aurora PostgreSQL 버전 9.6에서 `track_activity_query_size` 파라미터에 대한 기본 설정은 1,024바이트입니다. Aurora PostgreSQL 버전 10이상에서 기본 설정은 4,096바이트입니다.

최대 텍스트 크기

Aurora PostgreSQL 버전 12 이하에 대한 `track_activity_query_size` 제한은 102,400바이트입니다. 버전 13 이상에서는 최대 1MB입니다.

엔진이 성능 개선 도우미에 1MB를 반환하면 콘솔에는 처음 4KB만 표시됩니다. 쿼리를 다운로드하면 전체 1MB를 받을 수 있습니다. 이 경우 보기 및 다운로드하면 다른 바이트 수가 반환됩니다. `track_activity_query_size` DB 파라미터에 대한 자세한 내용은 PostgreSQL 설명서에서 [런타임 통계](#)를 참조하세요.

SQL 텍스트 크기를 늘리려면 `track_activity_query_size` 제한을 늘립니다. 파라미터를 수정하려면 Aurora PostgreSQL DB 인스턴스와 연결된 파라미터 그룹에서 파라미터 설정을 변경하세요.

인스턴스가 기본 파라미터 그룹을 사용할 때 설정 변경

1. 적절한 DB 엔진 및 DB 엔진 버전에 대해 새로운 DB 인스턴스 파라미터 그룹을 생성합니다.
2. 새 파라미터 그룹에 파라미터를 설정합니다.
3. 새 파라미터 그룹을 DB 인스턴스에 연결합니다.

DB 인스턴스 파라미터 설정에 대한 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하세요.

성능 개선 도우미 대시보드에서 SQL 텍스트 보기 및 다운로드

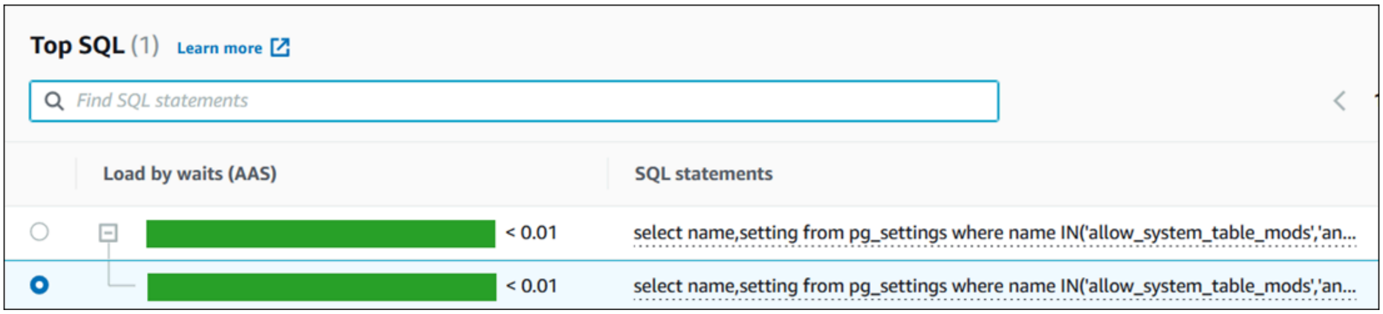
성능 개선 도우미 대시보드에서 SQL 텍스트를 보기 및 다운로드할 수 있습니다.

성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트를 보려면

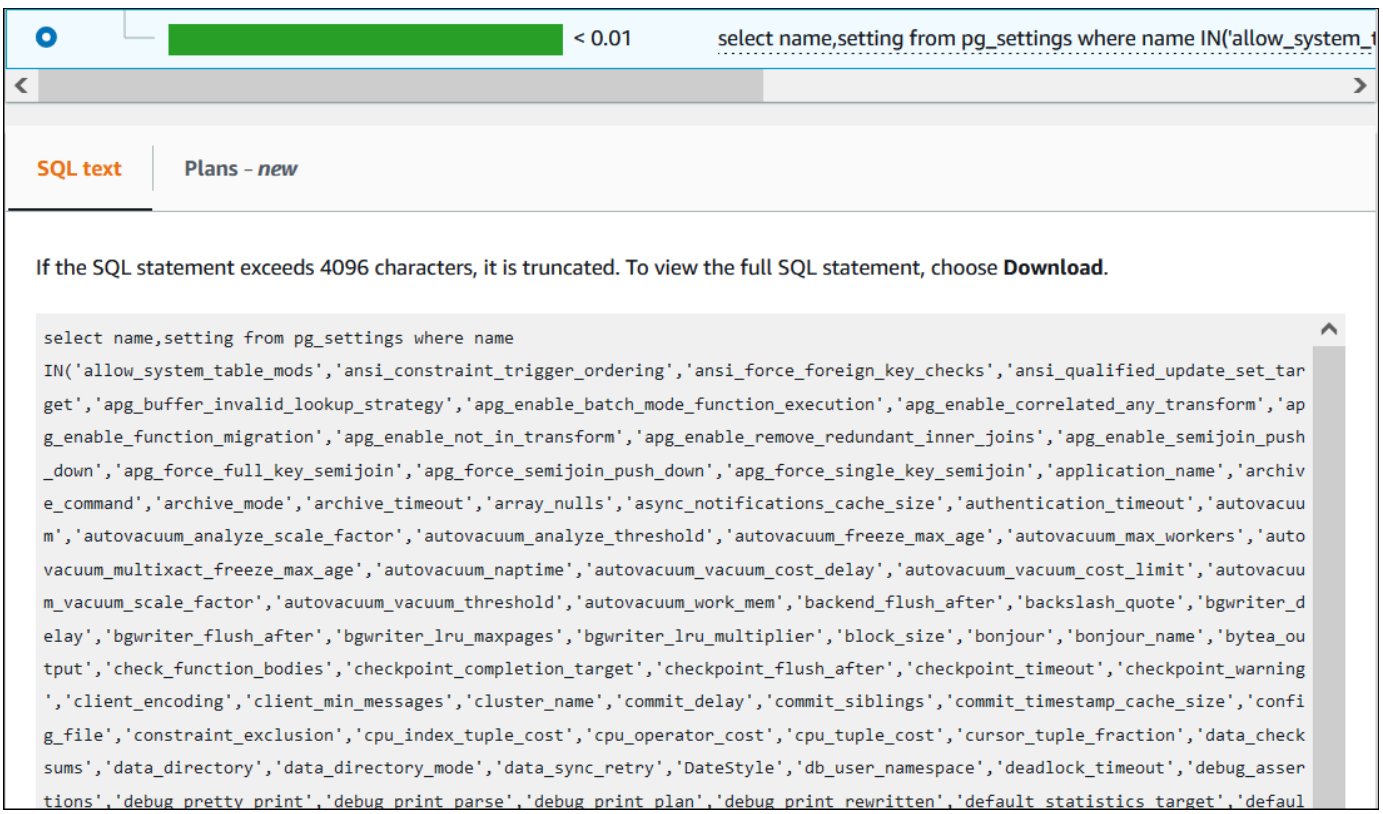
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 성능 개선 도우미 대시보드에서 아래로 스크롤하여 상위 SQL 탭으로 이동합니다.
5. 더하기(+) 기호를 선택하여 SQL 다이제스트를 펼치고 다이제스트의 하위 쿼리 중 하나를 선택합니다.

   500바이트 이상의 텍스트가 있는 SQL 문은 다음 이미지와 유사합니다.

6. SQL 텍스트 탭까지 아래로 스크롤합니다.

   성능 개선 도우미 대시보드는 각 SQL 문에 최대 4,096바이트를 표시할 수 있습니다.

7. (선택 사항) 복사를 선택하여 표시된 SQL 문을 복사하거나 다운로드를 선택하여 최대 DB 엔진 한도까지 SQL 텍스트를 볼 수 있는 SQL 문을 다운로드합니다.

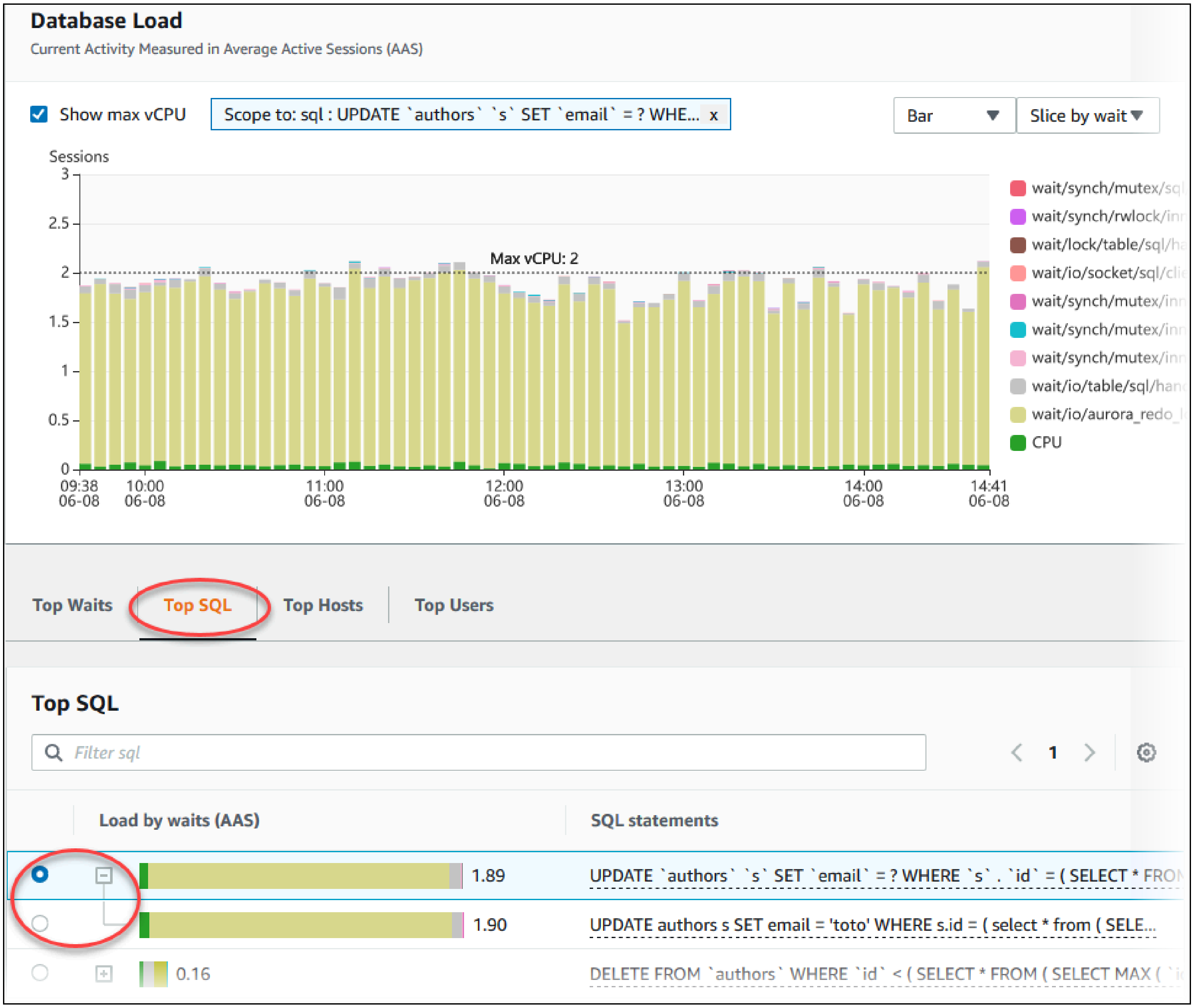
> **Note**
>
> SQL 문을 복사하거나 다운로드하려면 팝업 차단 기능을 비활성화하세요.

성능 개선 도우미 대시보드에서 SQL 통계 보기

성능 개선 도우미 대시보드에서 SQL 통계는 데이터베이스 로드(Database load) 차트의 상위 SQL(Top SQL) 탭에서 확인할 수 있습니다.

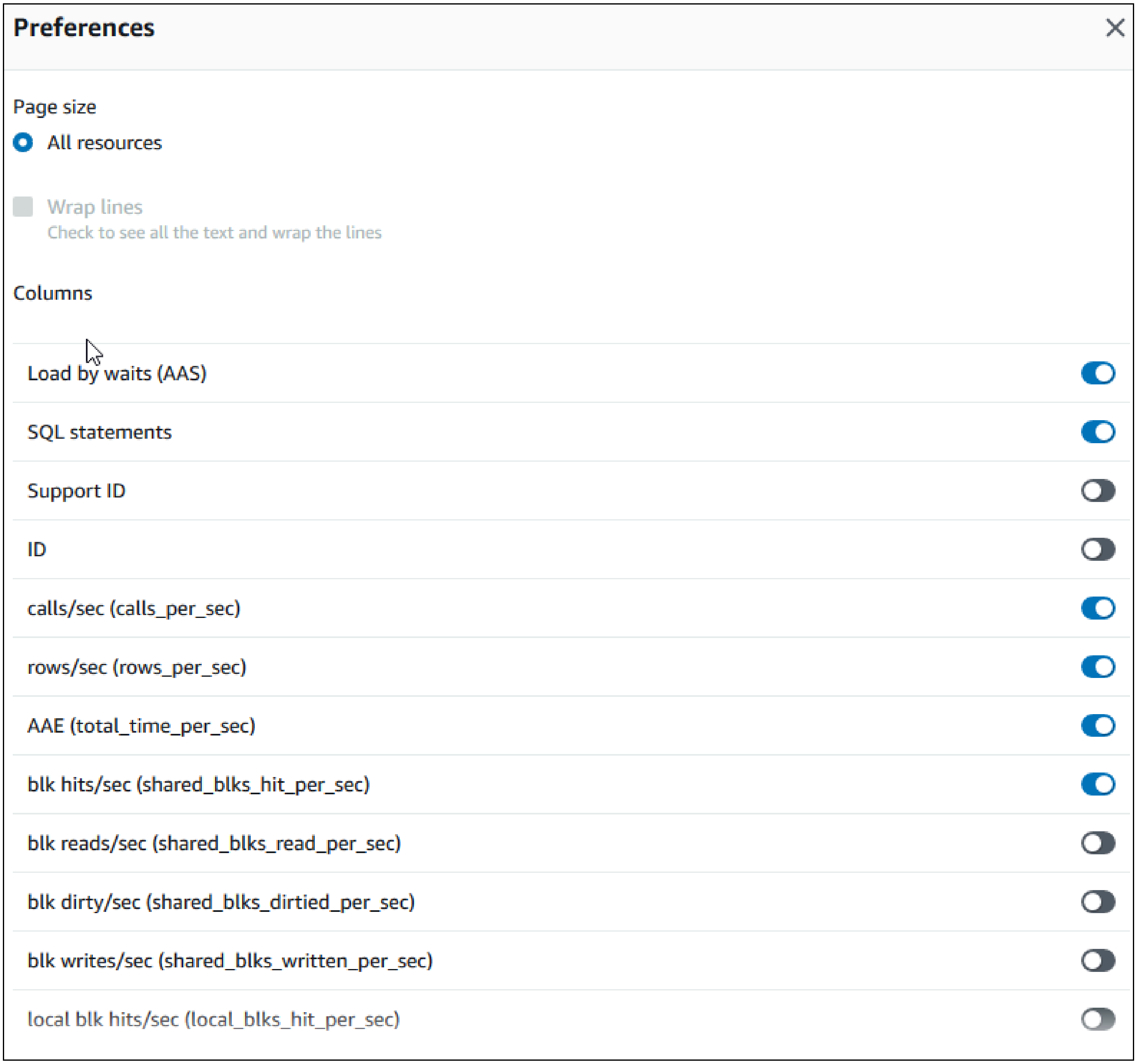
SQL 통계를 보는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 성능 개선 도우미(Performance Insights)를 선택합니다.
3. 페이지 상단에서 SQL 통계를 확인하려는 데이터베이스를 선택합니다.
4. 페이지의 하단으로 스크롤하고 상위 SQL(Top SQL)을 선택합니다.
5. 개별 문(Aurora MySQL만 해당) 또는 다이제스트 쿼리를 선택합니다.

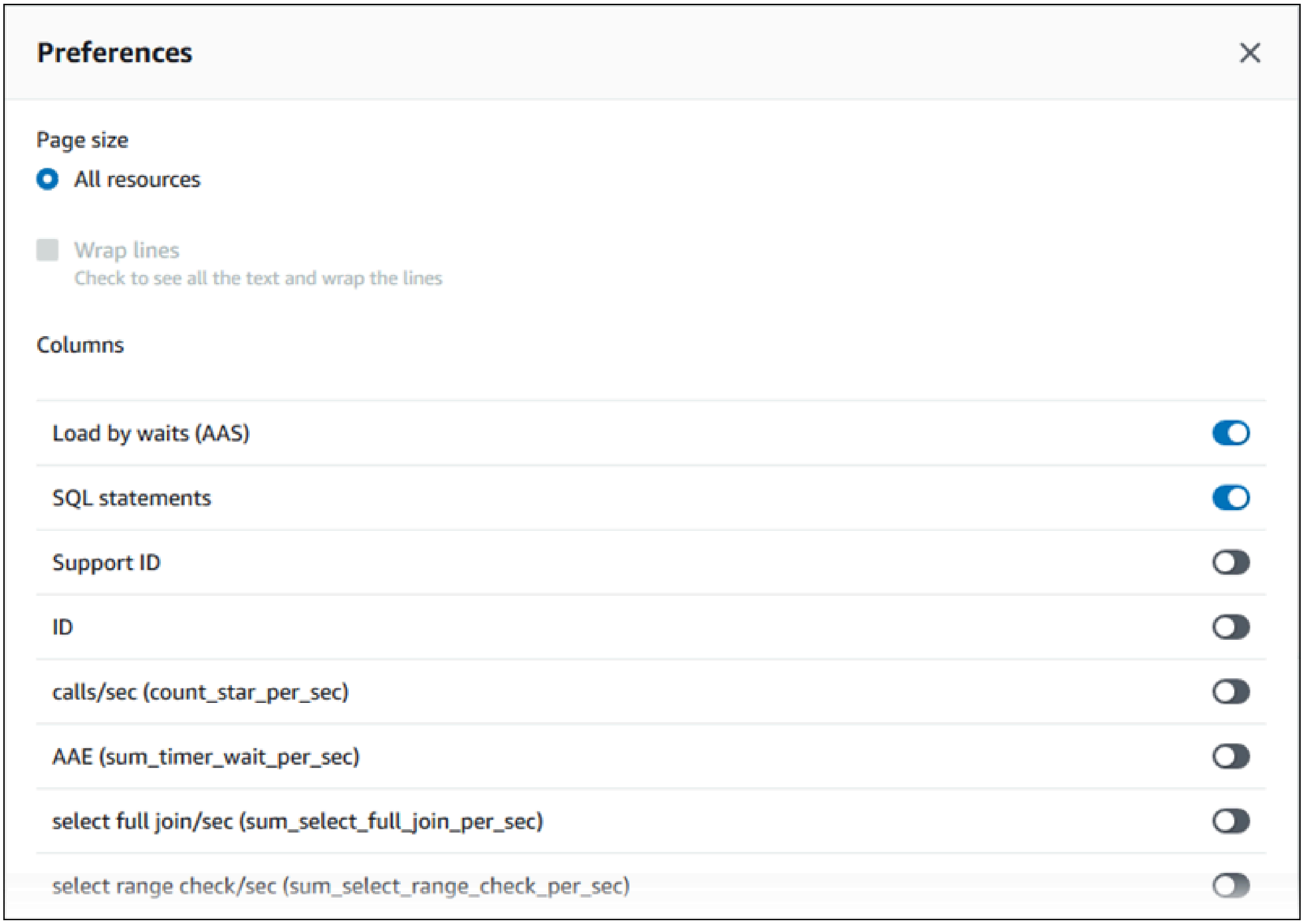


6. 차트 오른쪽 상단 모서리에 있는 기어 모양 아이콘을 선택하여 표시할 통계를 선택하십시오. Amazon RDSAurora엔진의 SQL 통계에 대한 설명을 보려면 [성능 개선 도우미에 대한 SQL 통계](#) 섹션을 참조하세요.

   다음 예는 Aurora PostgreSQL의 기본 설정을 보여줍니다.



다음 예는 Aurora My SQL DB 인스턴스의 기본 설정을 보여줍니다.



7. 저장(Save)을 선택하여 기본 설정을 저장합니다.

   상위 SQL(Top SQL) 테이블이 새로 고쳐집니다.

성능 개선 도우미 사전 권장 사항 보기

Amazon RDS 성능 개선 도우미는 특정 지표를 모니터링하고 특정 리소스에 대해 문제가 될 수 있다고 간주되는 수준을 분석하여 임계값을 자동으로 생성합니다. 지정된 기간 동안 새 지표 값이 사전 정의된 임계값을 초과하면 성능 개선 도우미는 사전 예방적 권장 사항을 생성합니다. 이 권장 사항은 향후 데이터베이스 성능에 미치는 영향을 방지하는 데 도움이 됩니다. 이러한 사전 예방적 권장 사항을 받으려면 유료 등급 보존 기간과 함께 성능 개선 도우미를 활성화해야 합니다.

성능 개선 도우미 켜기에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 섹션을 참조하세요. 성능 개선 도우미의 요금 및 데이터 보존에 대한 자세한 내용은 성능 개선 도우미의 요금 및 데이터 보존 섹션을 참조하세요.

사전 예방적 권장 사항이 지원되는 리전, DB 엔진 및 인스턴스 클래스를 알아보려면 성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원 섹션을 참조하세요.

권장 사항 세부 정보 페이지에서 사전 권장 사항에 대한 자세한 분석 및 권장 조사를 볼 수 있습니다.

권장 사항에 대한 자세한 내용은 Amazon Aurora의 권장 사항 단원을 참조하세요.

사전 예방적 권장 사항에 대한 자세한 분석을 보려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 다음 중 하나를 수행합니다.

   - 권장 사항을 선택합니다.

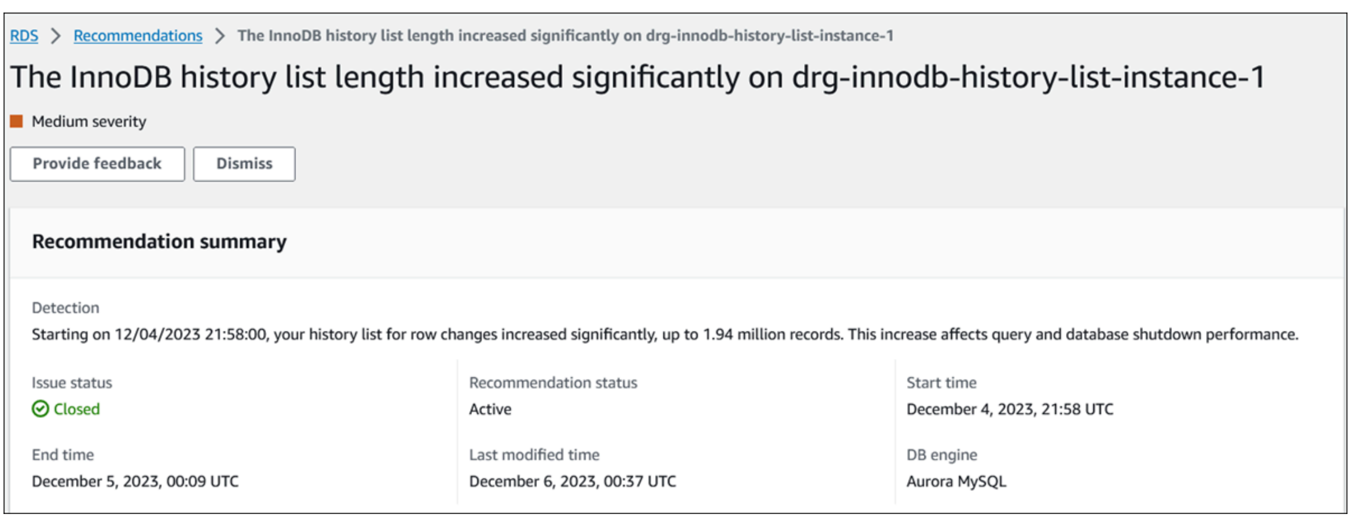
     권장 사항 페이지에는 계정 내 모든 리소스의 심각도별로 정렬된 권장 사항 목록이 표시됩니다.
   - 데이터베이스를 선택한 다음 데이터베이스 페이지에서 리소스에 대한 권장 사항을 선택합니다.

     권장 사항 탭에는 선택한 리소스에 대한 권장 사항 및 세부 정보가 표시됩니다.
3. 사전 예방적 권장 사항을 찾아 세부 정보 보기를 선택합니다.

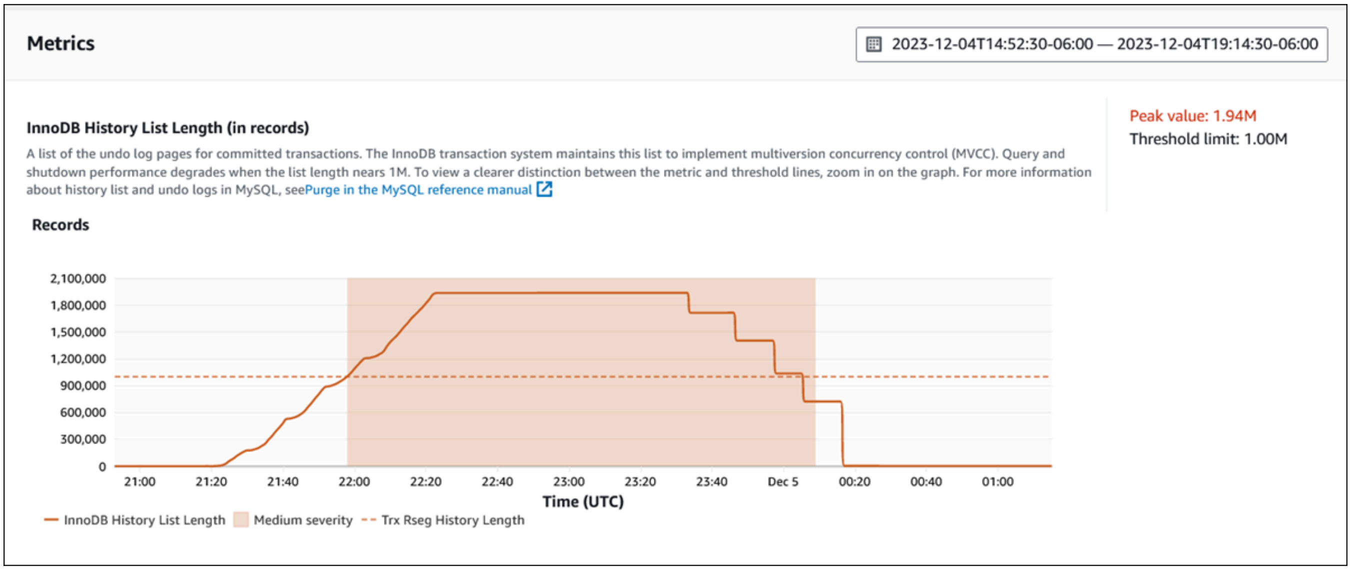
   권장 사항 세부 정보 페이지가 표시됩니다. 제목은 발견된 문제가 있는 리소스의 이름과 심각도를 제공합니다.

   권장 사항 세부 정보 페이지의 구성 요소는 다음과 같습니다.

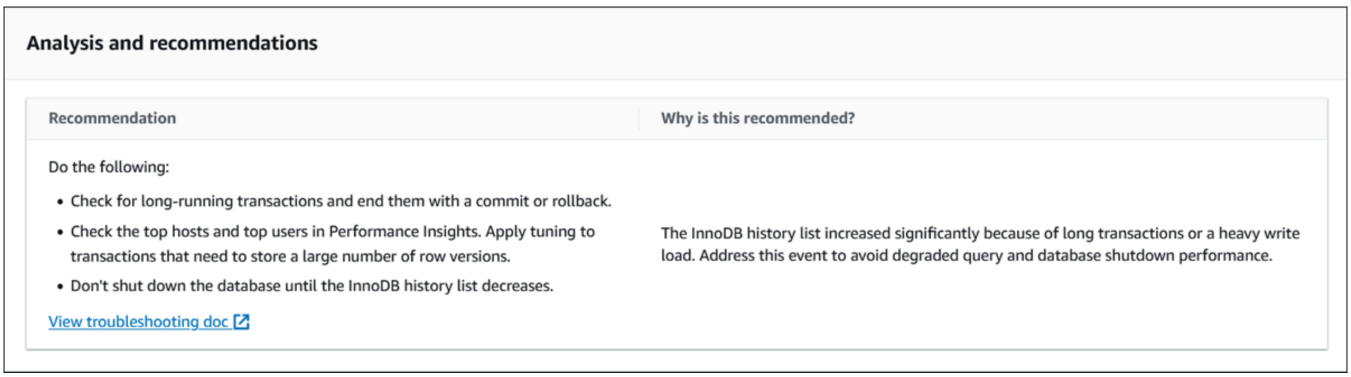
   - 권장 사항 요약 - 감지된 문제, 권장 사항 및 문제 상태, 문제 시작 및 종료 시간, 권장 사항 수정 시간, 엔진 유형입니다.



- 지표 - 감지된 문제의 그래프입니다. 각 그래프에는 리소스의 기준 동작에 따라 결정된 임계값과 이상 발생 시점부터 보고된 지표 데이터가 표시됩니다.



- 분석 및 권장 사항 - 권장 사항 및 해당 권장 사항을 제안한 이유입니다.



문제의 원인을 검토한 다음 제안된 권장 조치를 수행하여 문제를 해결하거나 오른쪽 상단의 삭제를 선택하여 권장 사항을 무시할 수 있습니다.

Aurora용 성능 개선 도우미 API를 사용하여 지표 검색

성능 개선 도우미를 켜면 API에서 인스턴스 성능에 대한 가시성을 제공합니다. Amazon CloudWatch Logs는 AWS 서비스의 벤딩 모니터링 지표에 대해 신뢰할 수 있는 소스를 제공합니다.

성능 개선 도우미는 평균 활성 세션(AAS) 수로 측정되는 데이터베이스 로드에 대한 도메인 별 보기를 제공합니다. 이 지표는 API 소비자에게 2차원 시계열 데이터 세트로 표시됩니다. 데이터의 시간 차원은 쿼리된 시간 범위 내 각 시점에 대한 DB 로드 데이터를 제공합니다. 각 시점에서는 요청된 차원에 관해 해당 시점에서 측정되는 전체 부하를 분해합니다(예: `SQL`, `Wait-event`, `User` 또는 `Host`).

Amazon RDS Performance Insights는 데이터베이스 성능을 분석하고 관련 문제를 해결할 수 있도록 Amazon Aurora 클러스터를 모니터링합니다. 성능 개선 도우미 데이터를 볼 수 있는 한 가지 방법은 AWS Management Console에서 보는 것입니다. 또한 성능 개선 도우미는 사용자가 자신의 데이터를 쿼리할 수 있도록 퍼블릭 API도 제공합니다. API를 사용하여 다음을 수행할 수 있습니다:

- 데이터를 데이터베이스로 오프로드
- 기존 모니터링 대시보드에 성능 개선 도우미 데이터 추가
- 모니터링 도구 구축

성능 개선 도우미 API를 사용하려면 Amazon RDS DB 인스턴스 중 하나에서 성능 개선 도우미를 활성화하십시오. 성능 개선 도우미 활성화에 대한 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 단원을 참조하십시오. 성능 개선 도우미 API에 대한 자세한 내용은 Amazon RDS 성능 개선 도우미 API 참조를 참조하십시오.

성능 개선 도우미 API에서는 다음과 같은 작업을 제공합니다.

성능 개선 도우미 작업	AWS CLI 명령	설명
`CreatePerformanceAnalysisReport`	`aws pi create-performance-analysis-report`	DB 인스턴스의 특정 기간에 대한 성능 분석 보고서를 생성합니다. 결과는 보고서의 고유 식별자인 `AnalysisReportId` 입니다.
`DeletePerformanceAnalysisReport`	`aws pi delete-performance-analysis-report`	성능 분석 보고서를 삭제합니다.
`DescribeDimensionKeys`	`aws pi describe-dimension-keys`	특정 기간에 대해 지표의 상위 N개 차원 키를 검색합니다.
`GetDimensionKeyDetails`	`aws pi get-dimension-key-details`	DB 인스턴스 또는 데이터 소스에 지정된 차원 그룹의 속성을 검색합니다. 예를 들어 SQL ID를 지정하고 차원 세부 정보를 사용할 수 있는 경우 `GetDimensionKeyDetails` 는 이 ID에 연결된 차

성능 개선 도우미 작업	AWS CLI 명령	설명
		원 db.sql.statement 의 전체 텍스트를 검색합니다. GetResourceMetrics 및 DescribeDimensionKeys 는 대용량 SQL 문 텍스트 검색을 지원하지 않으므로 이 작업을 유용하게 사용할 수 있습니다.
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	보고서에 대한 인사이트가 포함된 보고서를 검색합니다. 결과에는 보고서 상태, 보고서 ID, 보고서 시간 세부 정보, 인사이트 및 권장 사항이 포함됩니다.
GetResourceMetadata	aws pi get-resource-metadata	다양한 기능에 대한 특성을 검색합니다. 예를 들어 메타데이터는 특정 DB 인스턴스에서 특성이 켜지거나 꺼짐을 나타낼 수 있습니다.
GetResourceMetrics	aws pi get-resource-metrics	일정 기간의 데이터 소스 집합에 대한 성능 개선 도우미 지표를 검색합니다. 특정 차원 그룹 및 차원을 제공하고 각 그룹에 집계 및 필터링 기준을 제공할 수 있습니다.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	지정된 인스턴스에서 지정된 각 지표 유형에 대해 쿼리할 수 있는 차원을 검색합니다.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	지정된 DB 인스턴스에 대해 쿼리할 수 있는 지정된 지표 유형의 사용 가능한 모든 지표를 검색합니다.

성능 개선 도우미 작업	AWS CLI 명령	설명
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	DB 인스턴스에 대해 사용할 수 있는 모든 분석 보고서를 검색합니다. 각 보고서의 시작 시간을 기준으로 보고서가 나열됩니다.
ListTagsForResource	aws pi list-tags-for-resource	리소스에 추가된 모든 메타데이터 태그를 나열합니다. 목록에는 태그의 이름과 값이 포함됩니다.
TagResource	aws pi tag-resource	메타데이터 태그를 Amazon RDS 리소스에 추가합니다. 태그에는 이름과 값이 포함됩니다.
UntagResource	aws pi untag-resource	리소스에서 메타데이터 태그를 제거합니다.

성능 개선 도우미의 AWS CLI 예제 및 시계열 지표 검색에 대한 자세한 내용은 다음 주제를 참조하세요.

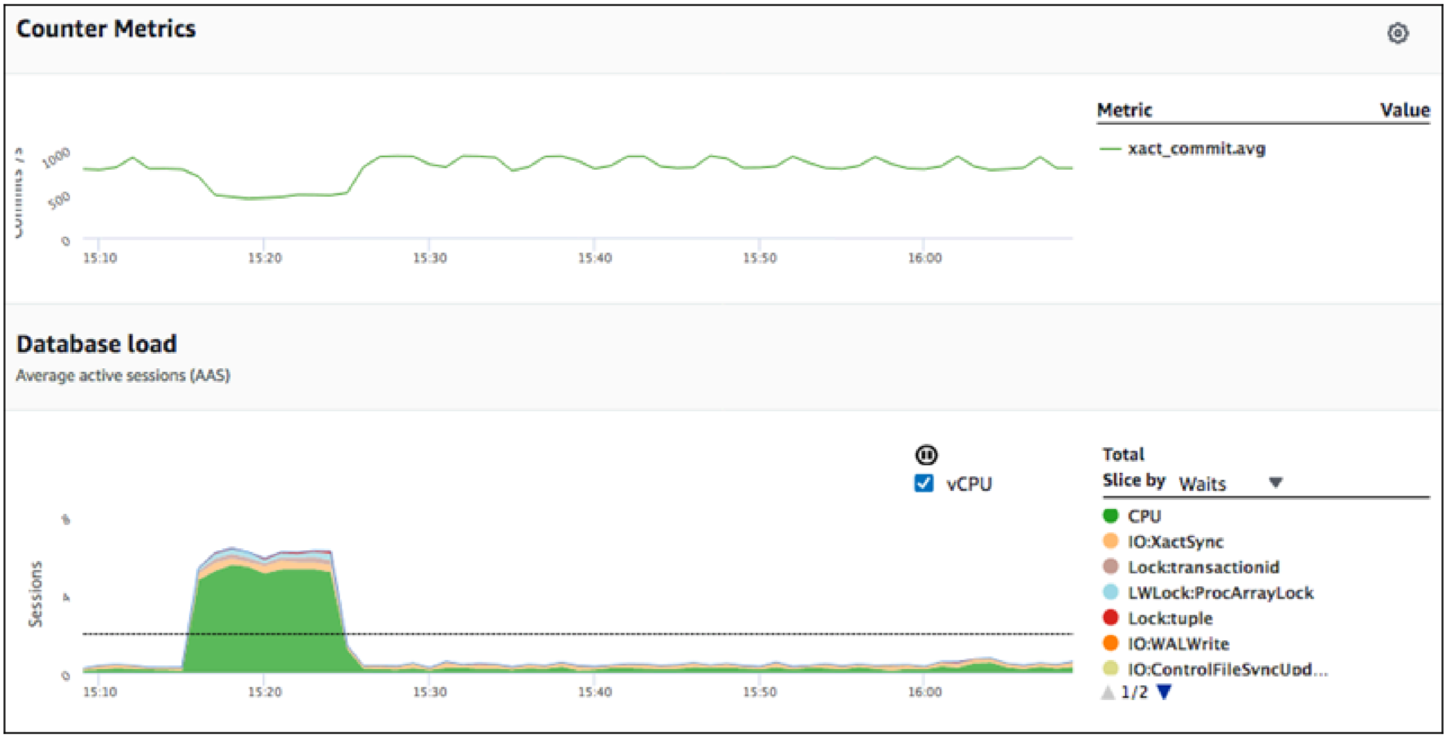
주제

- 성능 개선 도우미의 시계열 지표 조회
- 성능 개선 도우미에 대한 AWS CLI 예시

성능 개선 도우미의 시계열 지표 조회

`GetResourceMetrics` 연산은 성능 개선 도우미 데이터에서 시계열 지표를 하나 이상 조회합니다. `GetResourceMetrics`에는 지표 및 기간이 필요하고 데이터 포인트 목록이 포함된 응답을 반환합니다.

예를 들어 AWS Management Console에서 `GetResourceMetrics`는 다음 이미지에 표시된 것과 같이 [카운터 지표(Counter Metrics)] 차트와 [데이터베이스 로드(Database Load)] 차트를 채우는 데 사용됩니다.



GetResourceMetrics에서 반환하는 지표는 db.load를 제외하고 모두 표준 시계열 지표입니다. 이 지표는 Database Load(데이터베이스 로드) 차트에 표시됩니다. db.load 지표는 차원이라는 하위 구성 요소로 구분할 수 있다는 점에서 다른 시계열 지표와 다릅니다. 앞의 이미지에서 db.load는 db.load를 구성하는 대기 상태에 따라 구분되고 그룹화됩니다.

> **Note**
>
> GetResourceMetrics에서는 db.sampleload도 반환할 수 있지만 db.load 지표는 대부분의 경우 적절합니다.

GetResourceMetrics에서 반환하는 카운터 지표에 대한 자세한 내용은 [성능 개선 도우미 카운터](#)를 참조하십시오.

지표에 대해서는 다음 계산이 지원됩니다:

- 평균 – 일정 기간 동안 지표의 평균 값입니다. .avg를 지표 이름에 추가합니다.
- 최소 – 일정 기간 동안 지표의 최소 값입니다. .min를 지표 이름에 추가합니다.
- 최대 – 일정 기간 동안 지표의 최대 값입니다. .max를 지표 이름에 추가합니다.
- 합계 – 일정 기간 동안 지표 값의 합계입니다. .sum를 지표 이름에 추가합니다.
- 샘플 수 – 일정 기간 동안 지표가 수집된 횟수입니다. .sample_count를 지표 이름에 추가합니다.

예를 들어 지표를 300초 (5분) 동안 분당 1회씩 수집한다고 가정합시다. 각 분의 값은 1, 2, 3, 4, 5입니다. 이 경우 다음과 같은 계산 결과가 반환됩니다:

- 평균 – 3
- 최소 – 1
- 최대 – 5
- 합계 – 15
- 샘플 수 – 5

get-resource-metrics AWS CLI 명령 사용에 대한 자세한 내용은 get-resource-metrics 섹션을 참조하세요.

--metric-queries 옵션의 경우 결과를 얻고자 하는 쿼리를 한 개 이상 지정하십시오. 각 쿼리는 필수인 Metric과 선택 사항인 GroupBy 및 Filter 파라미터로 구성됩니다. 다음은 --metric-queries 옵션 사양을 보여주는 예입니다.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

성능 개선 도우미에 대한 AWS CLI 예시

다음 섹션에서 성능 개선 도우미의 AWS Command Line Interface(AWS CLI)에 대해 자세히 알아보고 AWS CLI 예제를 사용하세요.

주제

- 성능 개선 도우미에 대한 AWS CLI의 기본 제공 도움말
- 카운터 지표 검색
- 상위 대기 이벤트에 대한 DB 평균 로드 검색
- 상위 SQL에 대한 DB 평균 로드 검색
- SQL을 기준으로 필터링된 DB 평균 로드 검색

- SQL 문의 전체 텍스트 검색
- 특정 기간에 대한 성과 분석 보고서 생성
- 성능 분석 보고서 검색
- DB 인스턴스에 대한 모든 성능 분석 보고서 나열
- 성능 분석 보고서 삭제
- 성능 분석 보고서에 태그 추가
- 성능 분석 보고서에 대한 모든 태그 나열
- 성능 분석 보고서에서 태그 삭제

성능 개선 도우미에 대한 AWS CLI의 기본 제공 도움말

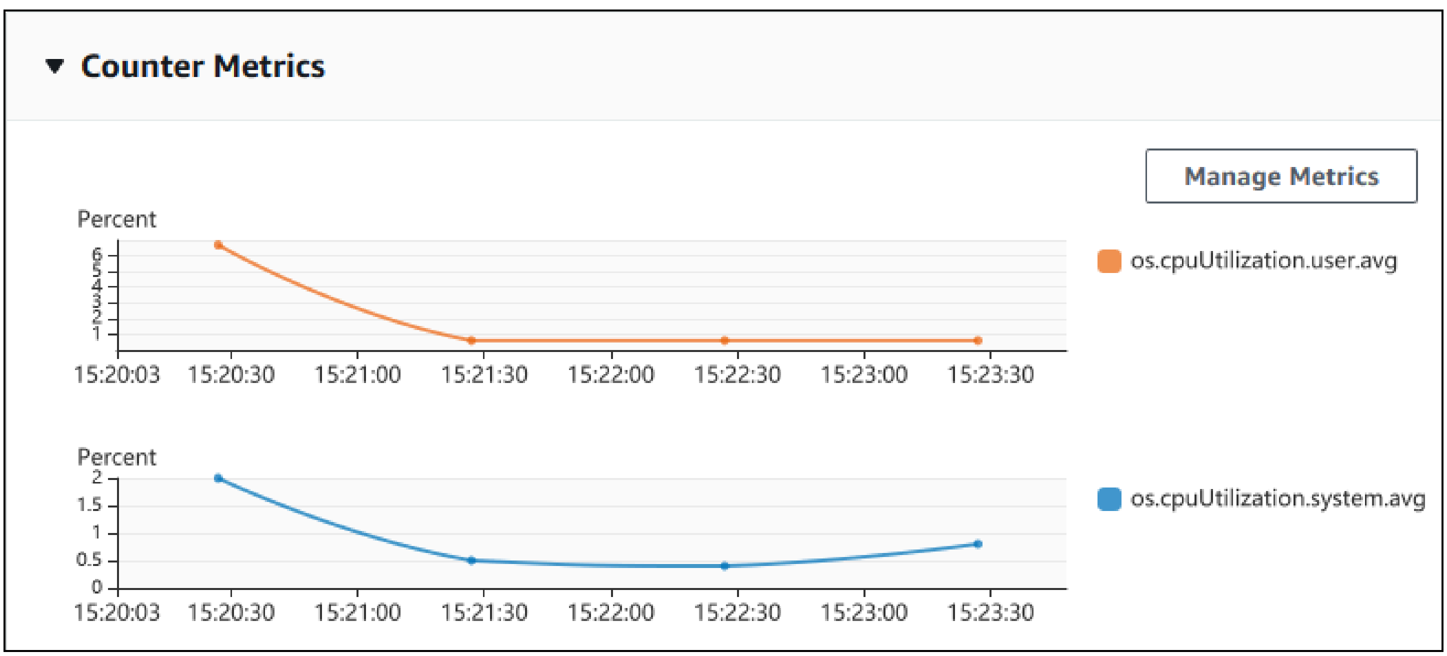
AWS CLI를 사용해 성능 개선 도우미 데이터를 볼 수 있습니다. 명령줄에 다음과 같이 입력하여 성능 개선 도우미용 AWS CLI 명령에 대한 도움말을 볼 수 있습니다.

```
aws pi help
```

AWS CLI가 설치되어 있지 않은 경우 설치에 대한 자세한 내용은 AWS CLI 사용 설명서의 AWS CLI 설치를 참조하세요.

카운터 지표 검색

다음 스크린샷은 AWS Management Console에 표시되는 카운터 지표 차트 2개를 나타낸 것입니다.



다음 예에서는 카운터 지표 차트 2개를 생성하기 위해 AWS Management Console이 사용하는 것과 동일한 데이터를 수집하는 방법을 보여줍니다.

대상 LinuxmacOS, 또는Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

`--metrics-query` 옵션에 대해 파일을 지정하면 명령이 더 쉽게 읽히도록 할 수 있습니다. 다음 예에서는 옵션에 대해 query.json이라는 파일을 사용합니다. 이 파일의 콘텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

다음 명령을 실행하여 파일을 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws pi get-resource-metrics \
```

```
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

앞의 예에서는 옵션에 다음 값을 지정합니다.

- `--service-type` – Amazon RDS용 RDS
- `--identifier` – DB 인스턴스에 대한 리소스 ID입니다.
- `--start-time` 및 `--end-time` – 쿼리할 기간에 대한 ISO 8601 `DateTime` 값으로서, 지원되는 형식은 여러 가지입니다

다음과 같이 1시간 범위로 쿼리합니다:

- `--period-in-seconds` – 1분당 쿼리에 대한 `60`
- `--metric-queries` – 쿼리 2개의 배열, 각 쿼리는 지표 1개에만 해당됨.

  지표 이름에는 지표를 유용한 범주로 분류하기 위해 점이 사용되고, 마지막 요소는 함수입니다. 예시에서 함수는 각 쿼리에 대해 `avg`입니다. Amazon CloudWatch와 마찬가지로 지원되는 함수는 `min`, `max`, `total` 및 `avg`입니다.

응답은 다음과 비슷합니다.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
```

```
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

응답에는 Identifier, AlignedStartTime 및 AlignedEndTime이 있습니다. --period-in-seconds 값이 60인 경우 시작 및 종료 시간은 분 단위로 맞춰져 있습니다. --period-in-seconds 값이 3600인 경우 시작 및 종료 시간은 시간 단위로 맞춰져 있습니다.

응답의 MetricList에는 다수의 항목이 있는데, 각각 Key 및 DataPoints 항목이 포함되어 있습니다. 각 DataPoint에는 Timestamp 및 Value이 있습니다. 쿼리는 1시간에 걸친 분당 데이터에 대한 것이므로 각 Datapoints 목록에는 Timestamp1/Minute1, Timestamp2/Minute2 등에서 최대 Timestamp60/Minute60까지 60개의 데이터 포인트가 있습니다.

쿼리는 두 가지 카운터 지표에 대한 것이므로 MetricList 응답에는 두 개의 요소가 있습니다.

상위 대기 이벤트에 대한 DB 평균 로드 검색

다음 예는 AWS Management Console에서 누적 영역 선 그래프를 생성하는 데 사용하는 것과 동일한 쿼리입니다. 이 예에서는 최상위 7개 대기 이벤트에 따라 구분된 로드의 마지막 한 시간 db.load.avg를 검색합니다. 명령은 카운터 지표 검색의 명령과 동일합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

다음 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
```

```
    --identifier db-ID ^
    --start-time 2018-10-30T00:00:00Z ^
    --end-time   2018-10-30T01:00:00Z ^
    --period-in-seconds 60 ^
    --metric-queries file://query.json
```

이 예시에서는 최상위 7개 대기 이벤트의 `db.load.avg` 및 `GroupBy`에 대한 지표를 지정합니다. 이 예의 유효 값에 대한 자세한 내용은 성능 개선 도우미 API 참조의 DimensionGroup 단원을 참조하십시오.

응답은 다음과 비슷합니다.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
```

```
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

이 응답에는 MetricList에 항목이 8개 있습니다. 총 db.load.avg에는 항목이 1개 있고, 최상위 7개 대기 이벤트 중 하나에 따라 구분된 db.load.avg에 각각에 대해서는 항목이 7개 있습니다. 첫 번째 예시와 달리 그룹화 차원이 있었기 때문에 지표에 대한 각 그룹화에는 키가 1개 있어야 합니다. 기본 카운터 지표 사용 사례처럼 각 지표에 키가 한 개만 있을 수는 없습니다.

상위 SQL에 대한 DB 평균 로드 검색

다음 예에서는 최상위 10개 SQL 문을 기준으로 db.wait_events를 그룹화합니다. SQL 문에는 두 가지 그룹이 있습니다.

- db.sql – 와 같은 SQL 문select * from customers where customer_id = 123
- db.sql_tokenized – 와 같은 토큰화된 SQL 문select * from customers where customer_id = ?

데이터베이스 성능 분석 시 파라미터만 다른 SQL 문은 하나의 로직 항목으로 간주하는 것이 도움이 될 수 있습니다. 따라서 쿼리 시에는 db.sql_tokenized를 사용할 수 있습니다. 그러나 특히 설명 계획에 관심이 있는 경우에는 때로 파라미터가 있는 전체 SQL 문을 검토하고 db.sql로 그룹화를 쿼리하는 것이 유용합니다. 토큰화된 SQL과 전체 SQL 간에는 상위-하위 관계가 있는데, 여러 개의 전체 SQL(하위)은 토큰화된 동일한 SQL(상위) 아래에 그룹화됩니다.

이 예의 명령은 상위 대기 이벤트에 대한 DB 평균 로드 검색의 명령과 유사합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

다음 예에는 db.sql_tokenized가 사용됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

이 예에서는 24시간 동안 쿼리를 실행하는데 1시간은 초 단위로 구성됩니다.

이 예시에서는 최상위 7개 대기 이벤트의 db.load.avg 및 GroupBy에 대한 지표를 지정합니다. 이 예의 유효 값에 대한 자세한 내용은 성능 개선 도우미 API 참조의 DimensionGroup 단원을 참조하십시오.

응답은 다음과 비슷합니다.

```
{
    "AlignedStartTime": 1540771200.0,
```

```
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

이 응답은 `MetricList`에 11개의 항목이 있는데(전체 1개, 최상위 토큰화 SQL 10개) 각 항목에는 시간당 `DataPoints`가 24개입니다.

토큰화된 SQL의 경우 각 차원 목록에 3개의 항목이 있습니다.

- `db.sql_tokenized.statement` – 토큰화된 SQL 문입니다.
- `db.sql_tokenized.db_id` – SQL 참조에 사용되는 기본 데이터베이스 ID 또는 기본 데이터베이스 ID를 사용할 수 없는 경우 성능 개선 도우미가 생성하는 합성 ID입니다. 이 예에서는 `pi-2372568224` 합성 ID를 반환합니다.

- `db.sql_tokenized.id` – 성능 개선 도우미 내부의 쿼리에 대한 ID입니다.

  AWS Management Console에서는 이 ID를 지원 ID라고 합니다. ID는 AWS Support에서 데이터베이스 문제를 해결하기 위해 조사할 수 있는 데이터이기 때문에 이 이름이 지정됩니다. AWS는 데이터의 보안 및 개인 정보를 매우 중요하게 취급하며, 거의 모든 데이터는 AWS KMS 키로 암호화되어 저장됩니다. 그러므로 AWS 내부의 어느 누구도 이 데이터를 볼 수 없습니다. 앞의 예에서 `tokenized.statement`와 `tokenized.db_id` 모두 암호화되어 저장됩니다. 데이터베이스 관련 문제가 있는 경우 AWS Support가 지원 ID를 참조하여 도움을 드릴 수 있습니다.

쿼리 시 `Group`에서 `GroupBy`을 지정하면 편리할 수 있습니다. 그러나 반환되는 데이터에 대한 더 세분화된 제어를 위해서는 차원 목록을 지정하십시오. 예를 들어 `db.sql_tokenized.statement`만 필요한 경우에는 query.json file에 `Dimensions` 속성을 추가할 수 있습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

SQL을 기준으로 필터링된 DB 평균 로드 검색

앞의 이미지에서는 특정 쿼리가 선택되어 있고 상위 평균 활성 세션 누적 영역 선 그래프는 이 쿼리로 범위가 지정되어 있습니다. 쿼리가 여전히 최상위 7개 전체 대기 이벤트에 대한 것이라 하더라도 응답의 값은 필터링됩니다. 필터로 인해 특정 필터의 짝이 되는 세션만 고려합니다.

이 예에서 해당되는 API 쿼리는 [상위 SQL에 대한 DB 평균 로드 검색](#) 단원의 명령과 유사합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

대상 LinuxmacOS, 또는Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
```

```
    --start-time 2018-10-30T00:00:00Z \
    --end-time   2018-10-30T01:00:00Z \
    --period-in-seconds 60 \
    --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
    --service-type RDS ^
    --identifier db-ID ^
    --start-time 2018-10-30T00:00:00Z ^
    --end-time   2018-10-30T01:00:00Z ^
    --period-in-seconds 60 ^
    --metric-queries file://query.json
```

응답은 다음과 비슷합니다.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
```

```
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
```

```
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
```

```
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

이 응답에서 모든 값은 query.json 파일에 지정된 토큰화된 SQL AKIAIOSFODNN7EXAMPLE의 기여에 따라 필터링됩니다. 키는 필터링된 SQL에 영향을 미친 상위 5개 대기 이벤트이므로 필터가 없는 쿼리와는 다른 순서를 따를 수 있습니다.

SQL 문의 전체 텍스트 검색

다음 예제에서는 DB 인스턴스 `db-10BCD2EFGHIJ3KL4M5NO6PQRS5`에 대한 SQL 문의 전체 텍스트를 검색합니다. `--group`은 `db.sql`이고 `--group-identifier`는 `db.sql.id`입니다. 이 예제에서 *`my-sql-id`*는 `pi get-resource-metrics` 또는 `pi describe-dimension-keys`를 호출하여 검색된 SQL ID를 나타냅니다.

다음 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

Windows의 경우:

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

이 예제에서는 차원 세부 정보를 사용할 수 있습니다. 따라서 성능 개선 도우미는 잘리지 않은 SQL 문의 전체 텍스트를 검색합니다.

```
{
    "Dimensions":[
```

```
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

특정 기간에 대한 성과 분석 보고서 생성

다음 예시에서는 db-loadtest-0 데이터베이스에 대해 1682969503 시작 시간과 1682979503 종료 시간을 사용하여 성능 분석 보고서를 생성합니다.

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

응답은 보고서의 고유 식별자인 report-0234d3ed98e28fb17입니다.

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

성능 분석 보고서 검색

다음 예시에서는 report-0d99cc91c4422ee61 보고서에 대한 분석 보고서 세부 정보를 검색합니다.

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

응답은 보고서 상태, ID, 시간 세부 정보, 인사이트를 제공합니다.

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```


DB 인스턴스에 대한 모든 성능 분석 보고서 나열

다음 예시에서는 `db-loadtest-0` 데이터베이스에 대해 사용 가능한 모든 성능 분석 보고서를 나열합니다.

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

응답에는 보고서 ID, 상태 및 기간 세부 정보와 함께 모든 보고서가 나열됩니다.


```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
```

```
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

성능 분석 보고서 삭제

다음 예시에서는 `db-loadtest-0` 데이터베이스에 대한 분석 보고서를 삭제합니다.

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

성능 분석 보고서에 태그 추가

다음 예시에서는 `report-01ad15f9b88bcbd56` 보고서에 키 name 및 값 `test-tag`로 태그를 추가합니다.

```
aws pi tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--region us-west-2
```

성능 분석 보고서에 대한 모든 태그 나열

다음 예시에서는 `report-01ad15f9b88bcbd56` 보고서에 대한 태그를 모두 나열합니다.

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

응답에는 보고서에 추가된 모든 태그의 값과 키가 나열됩니다.

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

성능 분석 보고서에서 태그 삭제

다음 예시에서는 `report-01ad15f9b88bcbd56` 보고서에서 name 태그를 삭제합니다.

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

태그가 삭제된 후 `list-tags-for-resource` API를 호출하면 이 태그를 나열하지 않습니다.

AWS CloudTrail을 사용하여 Performance Insights 호출 로깅

성능 개선 도우미는 사용자, 역할 또는 성능 개선 도우미의 AWS 서비스에서 수행한 작업을 기록하는 서비스인 AWS CloudTrail에서 실행됩니다. CloudTrail은 성능 개선 도우미에 대한 모든 API 호출을 이벤트로 캡처합니다. 이 캡처에는 Amazon RDS 콘솔과 코드에서 성능 개선 도우미 API 작업으로의 호출이 포함됩니다.

추적을 생성하면 성능 개선 도우미를 포함해 CloudTrail 이벤트를 Amazon S3 버킷으로 지속적으로 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 데이터를 사용하여 특정 정보를 확인할 수 있습니다. 이 정보

에는 성능 개선 도우미에 대한 요청, 요청한 IP 주소, 요청 주체 및 요청한 시간이 포함됩니다. 또한 추가 세부 정보도 포함되어 있습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하세요.

CloudTrail의 성능 개선 도우미 정보 작업

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 성능 개선 도우미에서 활동이 수행되면 해당 활동은 [이벤트 기록(Event history)]에서 CloudTrail 콘솔의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하세요.

성능 개선 도우미의 이벤트를 포함해 AWS 계정의 이벤트를 계속 기록하려면 trail을 생선합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 AWS 리전의 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 작업하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 다음 주제를 참조하세요.

- 트레일 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

모든 성능 개선 도우미 작업은 CloudTrail이 기록하며 성능 개선 도우미 API 참조에 문서화됩니다. 예를 들어, `DescribeDimensionKeys` 및 `GetResourceMetrics` 작업에 대한 호출은 CloudTrail 로그 파일의 항목을 생성합니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 보안 인증 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

성능 개선 도우미 로그 파일 항목

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 원본의 단일 요청을 나타냅니다. 각 이벤트에는 요청된 작업에 대한 정보, 작업 날짜 및 시간, 요청 파라미터 등이 포함됩니다. CloudTrail 로그 파일은 퍼블릭 API 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예는 `GetResourceMetrics`작업을 보여주는 CloudTrail 로그 항목입니다.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
```

```
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

성능 개선 도우미 API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

AWS PrivateLink를 사용하여 VPC와 Amazon RDS 성능 개선 도우미 사이에 프라이빗 연결을 생성할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결을 사용하지 않고 VPC에 있는 것처럼 성능 개선 도우미에 액세스할 수 있습니다. VPC의 인스턴스에서 성능 개선 도우미에 액세스하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

AWS PrivateLink에서 제공되는 인터페이스 엔드포인트를 생성하여 이 프라이빗 연결을 설정합니다. 인터페이스 엔드포인트에 대해 사용 설정하는 각 서브넷에서 엔드포인트 네트워크 인터페이스를 생성합니다. 이는 성능 개선 도우미로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다.

자세한 내용은 AWS PrivateLink 가이드의 AWS PrivateLink를 통해 AWS 서비스에 액세스를 참조하세요.

성능 개선 도우미에 대한 고려사항

성능 개선 도우미에 대한 인터페이스 엔드포인트를 설정하려면 먼저 AWS PrivateLink 안내서의 고려사항을 검토합니다.

성능 개선 도우미는 인터페이스 엔드포인트를 통해 모든 API 작업에 대한 직접 호출을 지원합니다.

기본적으로 인터페이스 엔드포인트를 통해 성능 개선 도우미에 대한 전체 액세스가 허용됩니다. 보안 그룹을 엔드포인트 네트워크 인터페이스와 연결하면 인터페이스 엔드포인트를 통해 성능 개선 도우미로 향하는 트래픽을 제어할 수 있습니다.

가용성

성능 개선 도우미 API는 현재 성능 개선 도우미를 지원하는 AWS 리전의 VPC 엔드포인트를 지원합니다. 성능 개선 도우미의 가용성에 대한 자세한 내용은 성능 개선 도우미를 지원하는 리전 및 Aurora DB 엔진 단원을 참조하세요.

성능 개선 도우미에 대한 인터페이스 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 성능 개선 도우미에 대한 인터페이스 엔드포인트를 생성할 수 있습니다. 자세한 내용은 AWS PrivateLink 설명서의 인터페이스 엔드포인트 생성을 참조하십시오.

다음 서비스 이름을 사용하여 성능 개선 도우미에 대한 인터페이스 엔드포인트를 생성합니다.

인터페이스 엔드포인트에 프라이빗 DNS를 활성화하는 경우, 리전에 대한 기본 DNS 이름을 사용하여 성능 개선 도우미에 API 요청을 전송할 수 있습니다. 예: `pi.us-east-1.amazonaws.com`.

성능 개선 도우미 API에 대한 VPC 엔드포인트 정책 생성

엔드포인트 정책은 인터페이스 엔드포인트에 연결할 수 있는 IAM 리소스입니다. 기본 엔드포인트 정책을 사용하면 인터페이스 엔드포인트를 통해 성능 개선 도우미에 대한 전체 액세스가 허용됩니다. VPC에서 성능 개선 도우미에 허용되는 액세스를 제어하려면 사용자 지정 엔드포인트 정책을 인터페이스 엔드포인트에 연결합니다.

엔드포인트 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체 (AWS 계정, IAM 사용자, IAM 역할)
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 AWS PrivateLink 가이드의 엔드포인트 정책을 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예제: 성능 개선 도우미 작업에 대한 VPC 엔드포인트 정책

다음은 사용자 지정 엔드포인트 정책의 예입니다. 이 정책을 인터페이스 엔드포인트에 연결하면 나열된 성능 개선 도우미 작업에 대한 액세스 권한이 모든 리소스의 모든 보안 주체에 부여됩니다.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
```

```
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

예제: 지정된 AWS 계정의 모든 액세스를 거부하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 AWS 계정 123456789012가 엔드포인트를 사용하는 리소스에 대한 모든 액세스를 거부합니다. 이 정책은 다른 계정의 모든 작업을 허용합니다.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

성능 개선 도우미에 IP 주소 지정

IP 주소가 있으면 VPC 내의 리소스가 서로 통신하고, 인터넷을 통해 다른 리소스와 통신할 수 있습니다. 성능 개선 도우미는 IPv4와 IPv6 주소 지정 프로토콜을 모두 지원합니다. 기본적으로 성능 개선 도우미와 Amazon VPC는 IPv4 주소 지정 프로토콜을 사용합니다. 이 동작은 끌 수 없습니다. VPC를 생성할 때 VPC에 IPv4 CIDR 블록(프라이빗 IPv4 주소)를 지정해야 합니다.

선택적으로 IPv6 CIDR 블록을 VPC와 서브넷에 할당하고 그 블록에 속한 IPv6 주소를 서브넷의 RDS 리소스에 할당할 수도 있습니다. IPv6 프로토콜을 지원하면 지원되는 IP 주소 수가 늘어납니다. IPv6 프로토콜을 사용하면 향후에 인터넷이 성장할 때를 대비해 사용 가능한 주소를 충분히 확보할 수 있습니다. 신규 및 기존 RDS 리소스는 VPC 내에서 IPv4 및 IPv6 주소를 사용할 수 있습니다. 애플리케이션의 다른 부분에서 사용되는 두 프로토콜 간의 네트워크 트래픽을 구성, 보안 및 변환하면 운영 오버헤

드가 발생할 수 있습니다. Amazon RDS 리소스의 IPv6 프로토콜을 표준화하여 네트워크 구성을 간소화할 수 있습니다. 서비스 엔드포인트 및 할당량에 대한 자세한 내용은 Amazon Relational Database Service 엔드포인트 및 할당량을 참조하세요.

Aurora IP 주소 지정에 대한 자세한 내용은 Aurora IP 주소 지정을 참조하세요.

Amazon DevOps Guru for Amazon RDS로 Aurora 성능 이상 분석

Amazon DevOps Guru는 개발자와 운영자가 애플리케이션의 성능과 가용성을 개선하는 데 도움이 되는 완전관리형 운영 서비스입니다. DevOps Guru는 운영 문제 식별과 관련된 작업을 오프로드하므로 애플리케이션을 개선하기 위한 권장 사항을 신속하게 구현할 수 있습니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 Amazon DevOps Guru란 무엇인가요?를 참조하세요.

DevOps Guru는 모든 Amazon RDS DB 엔진의 기존 운영 문제를 감지, 분석 및 권장 사항을 제시합니다. DevOps Guru for RDS는 Amazon Aurora 데이터베이스의 성능 개선 도우미 지표에 기계 학습을 적용하여 이 기능을 확장합니다. 이러한 모니터링 기능을 통해 DevOps Guru for RDS는 성능 병목 현상을 감지 및 진단하고 특정 시정 조치를 권장할 수 있습니다. DevOps Guru for RDS는 Aurora databases 에서 문제가 발생하기 전에 문제를 감지할 수 있습니다.

이제 RDS 콘솔에서 이러한 권장 사항을 확인할 수 있습니다. 자세한 내용은 Amazon Aurora의 권장 사항 단원을 참조하십시오.

> **Important**
>
> Amazon DevOps Guru는 Aurora Serverless DB 클러스터에 사용할 수 없습니다.

다음 동영상은 RDS용 DevOps 전문가 개요입니다.

이 주제에 관해 자세히 알아보려면 내부에 있는 RDS용 Amazon DevOps Guru.를 참조하세요.

주제

DevOps Guru for RDS의 이점

Amazon Aurora 데이터베이스를 담당하는 경우 해당 데이터베이스에 영향을 미치는 이벤트 또는 회귀가 발생하고 있는 것을 알지 못할 수 있습니다. 이 문제에 대해 알아볼 때, 문제가 발생하는 이유나 어떻게 대응해야 할 지 모를 수도 있습니다. 데이터베이스 관리자(DBA)에게 도움을 요청하거나 타사 도구에 의존하는 대신 DevOps Guru for RDS의 권장 사항을 따를 수 있습니다.

DevOps Guru for RDS의 세부 분석을 통해 다음과 같은 이점을 얻을 수 있습니다.

빠른 진단

RDS용 DevOps Guru는 데이터베이스 원격 분석을 지속적으로 모니터링하고 분석합니다. 성능 개선 도우미, 향상된 모니터링 및 Amazon CloudWatch는 데이터베이스 클러스터에 대한 원격 분석 데이터를 수집합니다. DevOps Guru for RDS는 통계 및 기계 학습 기술을 사용하여 이 데이터를 마이닝하고 이상을 감지합니다. 원격 측정 데이터에 대한 자세한 내용은 Amazon Aurora 사용 설명서에서 성능 개선 도우미를 통한 Amazon Aurora 모니터링 및 Enhanced Monitoring을 사용하여 OS 지표 모니터링.

빠른 해결

각 이상 현상은 성능 문제를 식별하고 조사 또는 수정 작업 방법을 제안합니다. 예를 들어 RDS용 DevOps Guru는 특정 대기 이벤트를 조사하도록 권장할 수 있습니다. 또는 애플리케이션 풀 설정을 조정하여 데이터베이스 연결 수를 제한하도록 권장할 수도 있습니다. 이러한 권장 사항을 기반으로 수동으로 문제를 해결하는 것보다 성능 문제를 더 빨리 해결할 수 있습니다.

사전 예방 인사이트

DevOps Guru for RDS는 리소스의 지표를 사용하여 문제가 더 커지기 전에 잠재적으로 문제가 될 수 있는 동작을 탐지합니다. 예를 들어 데이터베이스에서 점점 더 많은 수의 온디스크 임시 테이블을 사용하여 성능에 영향을 미치기 시작할 가능성이 있는 경우, 이를 탐지할 수 있습니다. 그런 다음 DevOps Guru는 문제가 더 커지기 전에 문제를 해결하는 데 도움이 되는 권장 사항을 제공합니다.

Amazon 엔지니어의 깊이 있는 지식과 기계 학습

성능 문제를 감지하고 병목 현상을 해결하기 위해 DevOps Guru for RDS는 기계 학습(ML)과 고급 수학 공식을 사용합니다. Amazon 데이터베이스 엔지니어들은 수년간 수십만 개의 데이터베이스를 관리한 결과를 캡슐화하는 DevOps Guru for RDS의 개발에 기여했습니다. 이러한 집단 지식을 바탕으로 DevOps Guru for RDS를 통해 모범 사례를 가르칠 수 있습니다.

DevOps Guru for RDS 작동 방식

DevOps Guru for RDS는 Amazon RDS 성능 개선 도우미로부터 Aurora 데이터베이스에 대한 데이터를 수집합니다. 가장 중요한 지표는 `DBLoad`입니다. DevOps Guru for RDS는 성능 개선 도우미 지표를 사용하고, 기계 학습을 통해 분석하며, 대시보드에 인사이트를 게시합니다.

인사이트는 DevOps Guru가 탐지한 관련 이상의 모음입니다.

DevOps Guru for RDS에서, 이상은 Amazon Aurora 데이터베이스의 정상적인 성능으로 간주되는 것과 다른 패턴입니다.

사전 예방 인사이트

사전 예방 인사이트를 통해 문제가 발생하기 전에 문제를 일으킬 수 있는 행동을 파악할 수 있습니다. 여기에는 Amazon Aurora 데이터베이스에서 더 큰 문제가 발생하기 전에 문제를 해결하는 데 도움이 되는 권장 사항 및 관련 지표가 포함된 이상 항목이 포함되어 있습니다. 이러한 인사이트는 DevOps Guru 대시보드에 게시됩니다.

예를 들어 DevOps Guru는 Aurora PostgreSQL 데이터베이스에서 많은 온디스크 임시 테이블을 생성하고 있음을 감지할 수 있습니다. 이러한 추세를 해결하지 않으면 성능 문제가 발생할 수 있습니다. 각 사전 예방 인사이트에는 수정 조치에 대한 권장 사항 및 Amazon DevOps Guru의 사전 예방 인사이트를 활용하여 Aurora MySQL 조정 또는 Amazon DevOps Guru의 사전 예방 인사이트를 활용하여 Aurora PostgreSQL 튜닝의 관련 주제에 대한 링크가 포함되어 있습니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 DevOps Guru의 인사이트 활용을 참조하세요.

사후 대응 인사이트

사후 대응 인사이트는 비정상적인 동작이 발생하는 즉시 이를 식별합니다. DevOps Guru for RDS가 Amazon Aurora DB 인스턴스에서 성능 문제를 발견하면 DevOps Guru 대시보드에 사후 대응 인사이트를 게시합니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 DevOps Guru의 인사이트 활용을 참조하세요.

일반적인 이상

캐주얼 이상 항목은 사후 대응 인사이트 내에서 최상위 이상 항목입니다. 데이터베이스 로드(DB 로드)는 RDS용 DevOps 전문가의 인과 관계적 이상입니다.

이상은 심각도의 높음, 중간, 또는 낮음 수준을 할당하여 성능에 미치는 영향을 측정합니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 DevOps Guru for RDS의 주요 개념을 참조하세요.

DevOps Guru가 DB 인스턴스에서 진행 중인 이상 현상을 감지하면, RDS 콘솔의 데이터베이스 (Databases) 페이지에 알림이 표시됩니다. 또한 콘솔은 지난 24시간 동안 발생한 이상 현상을 알려줍니다. RDS 콘솔에서 이상 페이지로 이동하려면 경고 메시지에서 링크를 선택합니다. 또한 RDS 콘솔은 Amazon Aurora DB cluster 의 페이지에 알림을 표시합니다.

문맥적 이상

컨텍스트 이상 항목은 데이터베이스 로드(DB 로드) 내의 결과로, 사후 대응 인사이트와 관련이 있습니다. 각 문맥적 이상은 조사가 필요한 특정 Amazon Aurora 성능 문제를 설명합니다. 예를 들어 DevOps Guru for RDS는 CPU 용량을 늘리거나 DB 로드에 기여하는 대기 이벤트를 조사할 것을 권장할 수 있습니다.

> **Important**
>
> 프로덕션 인스턴스를 변경하기 전에 테스트 인스턴스에서 변경 사항을 테스트하는 것이 좋습니다. 이러한 방식으로 변경의 영향을 이해하게 됩니다.

자세한 내용은 Amazon DevOps Guru 사용 설명서의 [Amazon RDS의 이상 현상 분석](#)을 참조하세요.

DevOps Guru for RDS 설정

DevOps Guru for Amazon RDS가 Amazon Aurora 데이터베이스에 대한 인사이트를 게시할 수 있도록 하려면 다음 태스크를 완료합니다.

주제

- DevOps Guru for RDS에 사용되는 IAM 액세스 정책 구성
- Aurora DB 인스턴스의 성능 개선 도우미 활성화
- DevOps Guru 활성화 및 리소스 적용 범위 지정

DevOps Guru for RDS에 사용되는 IAM 액세스 정책 구성

RDS 콘솔에서 DevOps Guru의 알림을 보려면 AWS Identity and Access Management(IAM) 사용자 또는 역할에 다음 정책 중 하나가 있어야 합니다.

- AWS 관리형 정책 `AmazonDevOpsGuruConsoleFullAccess`
- AWS 관리형 정책 `AmazonDevOpsGuruConsoleReadOnlyAccess` 및 다음 정책 중 하나:
  - AWS 관리형 정책 `AmazonRDSFullAccess`
  - `pi:GetResourceMetrics` 및 `pi:DescribeDimensionKeys`를 포함하는 고객 관리형 정책

자세한 내용은 Performance Insights에 대한 액세스 정책 구성 단원을 참조하십시오.

Aurora DB 인스턴스의 성능 개선 도우미 활성화

DevOps Guru for RDS는 데이터를 위해 성능 개선 도우미를 사용합니다. 성능 개선 도우미가 없으면 DevOps Guru는 이상을 게시하지만 자세한 분석 및 권장 사항은 포함하지 않습니다.

Aurora DB 클러스터를 생성하거나 클러스터 인스턴스를 수정할 때 성능 개선 도우미를 활성화할 수 있습니다. 자세한 내용은 Aurora의 성능 개선 도우미 설정 및 해제 단원을 참조하십시오.

DevOps Guru 활성화 및 리소스 적용 범위 지정

다음 방법 중 하나로 Amazon Aurora 데이터베이스를 모니터링하도록 DevOps Guru를 활성화할 수 있습니다.

주제

- RDS 콘솔에서 DevOps Guru 활성화
- DevOps Guru 콘솔에 Aurora 리소스 추가
- AWS CloudFormation을 사용하여 Aurora 리소스 추가

RDS 콘솔에서 DevOps Guru 활성화

Amazon RDS 콘솔에서 여러 경로를 사용하여 DevOps Guru를 활성화할 수 있습니다.

주제

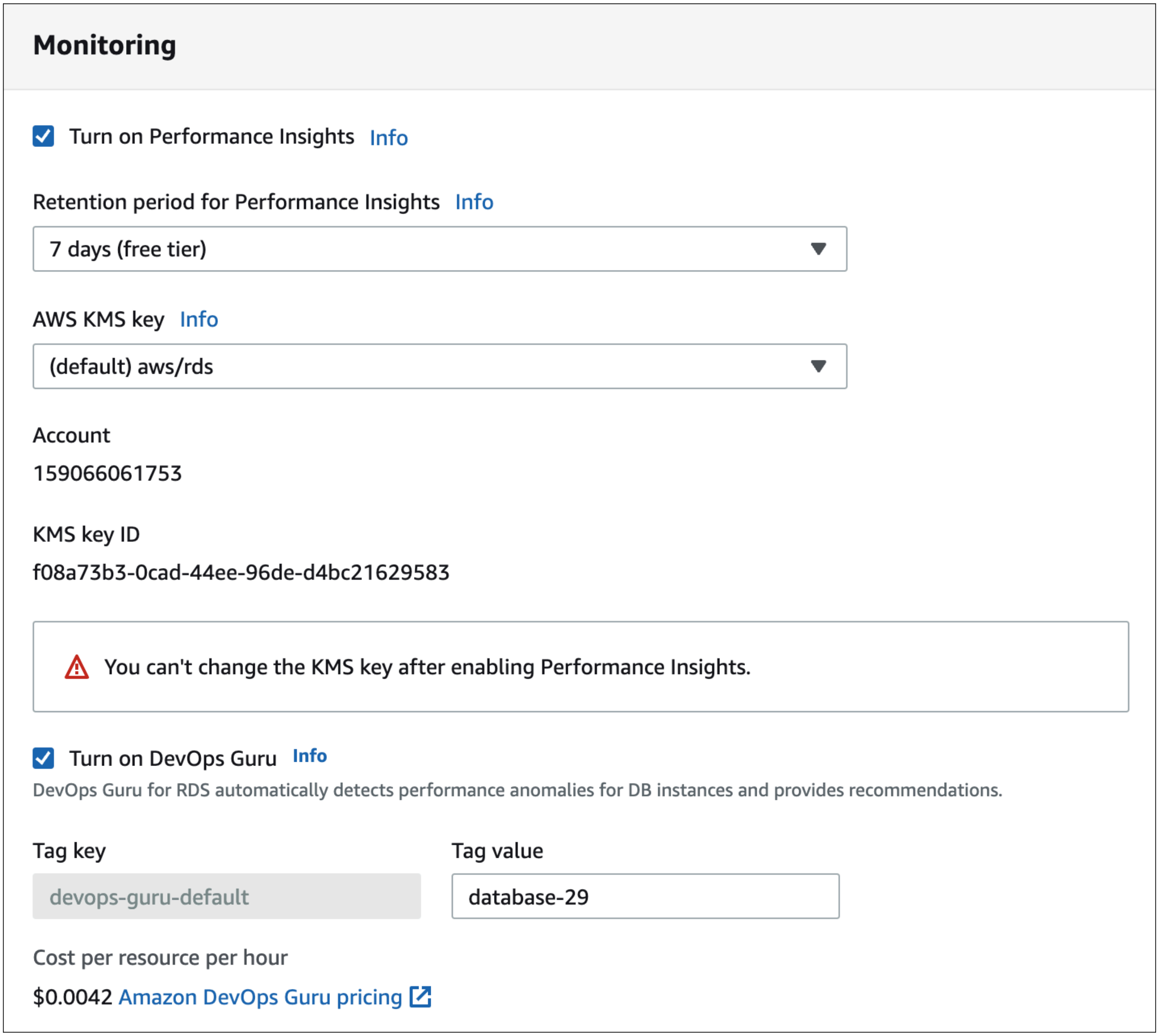
- Aurora 데이터베이스를 생성할 때 DevOps Guru 활성화
- 알림 배너에서 DevOps Guru 활성화
- DevOps Guru를 활성화할 때 권한 오류에 응답

Aurora 데이터베이스를 생성할 때 DevOps Guru 활성화

생성 워크플로에는 데이터베이스에 대한 DevOps Guru 적용 범위를 활성화하는 설정이 포함되어 있습니다. 이 설정은 프로덕션(Production) 템플릿을 선택할 때 기본적으로 지정됩니다.

Aurora 데이터베이스를 생성할 때 DevOps Guru를 활성화하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 모니터링 설정을 선택하는 단계를 제외하고 DB 클러스터 생성에 나와 있는 단계를 최대한 수행합니다.
3. 모니터링(Monitoring)에서 성능 개선 도우미 활성화(Turn on Performance Insights)를 선택합니다. DevOps Guru for RDS가 성능 이상에 대한 자세한 분석을 제공하려면 성능 개선 도우미를 활성화해야 합니다.
4. DevOps Guru 활성화(Turn on DevOps Guru)를 선택합니다.



5. DevOps Guru가 모니터링할 수 있도록 데이터베이스에 대한 태그를 생성합니다. 다음을 따릅니다.

   - 태그 키(Tag key) 텍스트 필드에서 **`Devops-Guru-`**로 시작하는 이름을 입력합니다.
   - 태그 값(Tag value) 텍스트 필드에서 원하는 값을 입력합니다. 예를 들어, Aurora 데이터베이스의 이름에 **`rds-database-1`**을 입력하는 경우 **`rds-database-1`**을 태그 값으로 입력할 수도 있습니다.

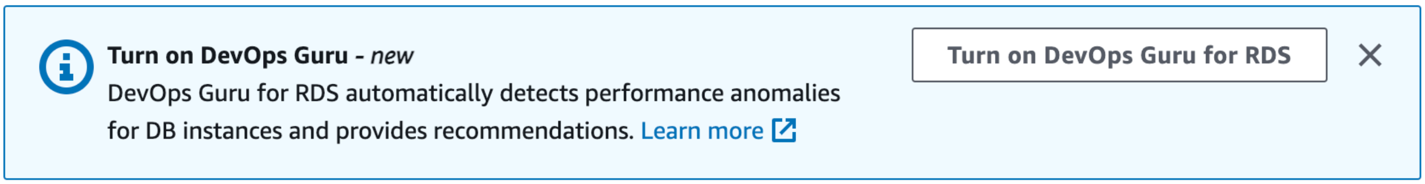
태그에 대한 자세한 내용은 Amazon DevOps Guru 사용 설명서의 '[태그를 사용하여 DevOps Guru 애플리케이션에서 리소스 식별](#)'을 참조하세요.

6. DB 클러스터 생성에 설명된 나머지 단계를 완료합니다.

알림 배너에서 DevOps Guru 활성화

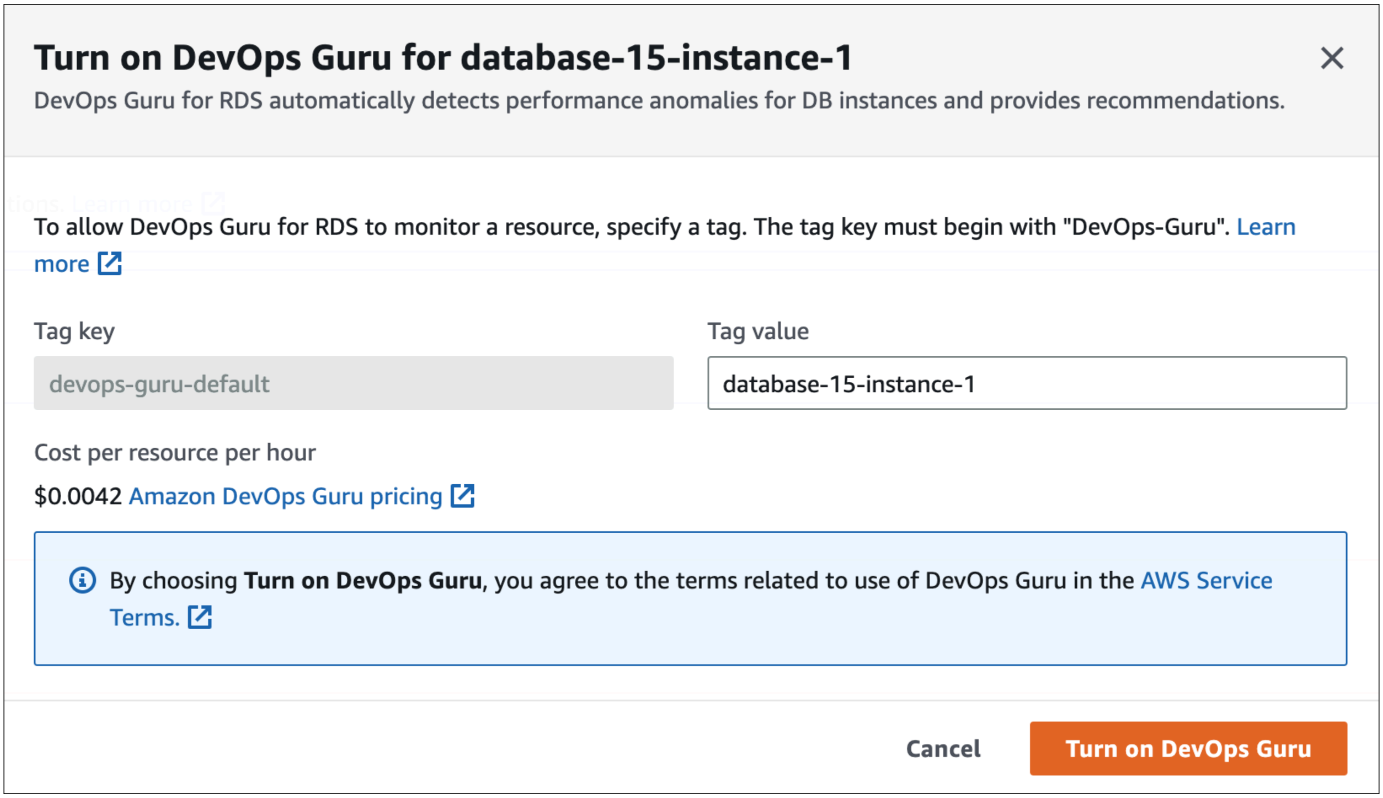
리소스가 DevOps Guru에 포함되지 않는 경우 Amazon RDS에서 다음 위치에 배너를 사용하여 알립니다.

- DB 클러스터 인스턴스의 모니터링(Monitoring) 탭
- 성능 개선 도우미 대시보드



Aurora 데이터베이스에 대해 DevOps Guru를 활성화하려면

1. 배너에서 DevOps Guru for RDS 활성화를 선택합니다.
2. 태그 키 이름 및 값을 입력합니다. 태그에 대한 자세한 내용은 Amazon DevOps Guru 사용 설명서의 '태그를 사용하여 DevOps Guru 애플리케이션에서 리소스 식별'을 참조하세요.



3. DevOps Guru 활성화를 선택합니다.

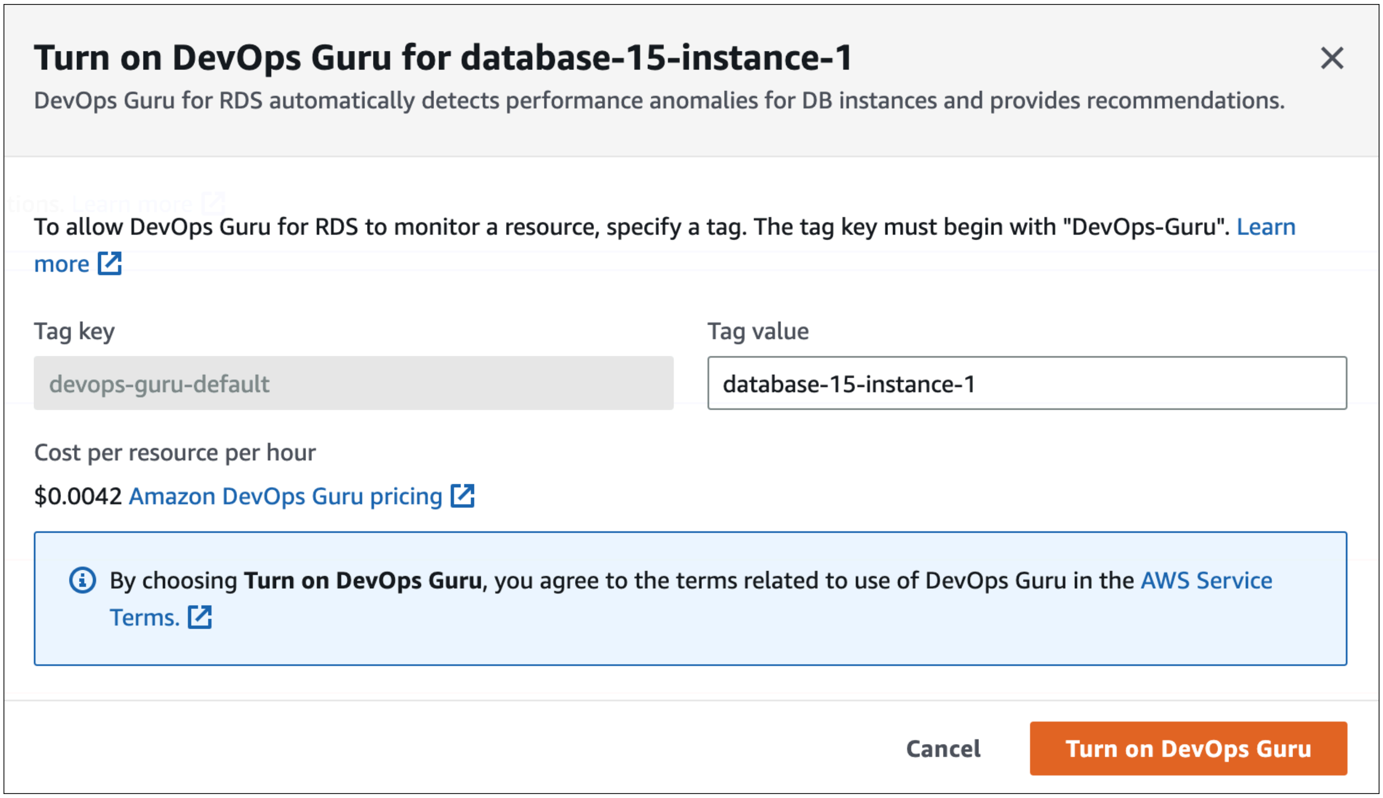
DevOps Guru를 활성화할 때 권한 오류에 응답

데이터베이스를 생성할 때 RDS 콘솔에서 DevOps Guru를 활성화하면 RDS에 권한 누락에 대해 다음 배너가 표시될 수 있습니다.

권한 오류에 응답하려면

1. IAM 사용자 또는 역할에 사용자 관리형 역할 `AmazonDevOpsGuruConsoleFullAccess`를 부여합니다. 자세한 내용은 DevOps Guru for RDS에 사용되는 IAM 액세스 정책 구성 단원을 참조하십시오.
2. RDS 콘솔을 엽니다.
3. 탐색 창에서 성능 개선 도우미를 선택합니다.
4. 방금 생성한 클러스터에서 DB 인스턴스를 선택합니다.
5. 스위치를 선택해 DevOps Guru for RDS 기능을 켭니다.

6. 태그 값을 선택합니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 '태그를 사용하여 DevOps Guru 애플리케이션에서 리소스 식별'을 참조하세요.



7. DevOps Guru 활성화를 선택합니다.

DevOps Guru 콘솔에 Aurora 리소스 추가

DevOps Guru 콘솔에서 DevOps Guru 리소스 적용 범위를 지정할 수 있습니다. Amazon DevOps Guru 사용 설명서의 DevOps Guru 리소스 적용 범위 지정에 나와 있는 단계를 따릅니다. 분석된 리소스를 편집할 때 다음 옵션 중 하나를 선택합니다.

- 모든 계정 리소스를 선택하여 AWS 계정 및 리전에서 Aurora 데이터베이스를 포함하여 지원되는 모든 리소스를 분석합니다.
- CloudFormation 스택을 선택하여 지정한 스택에 있는 Aurora 데이터베이스를 분석합니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 AWS CloudFormation 스택을 사용하여 DevOps Guru 애플리케이션에서 리소스 식별을 참조하세요.
- 태그를 선택하여 태그를 지정한 Aurora 데이터베이스를 분석합니다. 자세한 내용은 Amazon DevOps Guru 사용 설명서의 태그를 사용하여 DevOps Guru 애플리케이션에서 리소스 식별을 참조하세요.

자세한 내용은 Amazon DevOps Guru 사용 설명서의 DevOps Guru 활성화를 참조하세요.

AWS CloudFormation을 사용하여 Aurora 리소스 추가

태그를 사용하여 Aurora 리소스 적용 범위를 CloudFormation 템플릿에 추가할 수 있습니다. 다음 절차에서는 Aurora DB 인스턴스와 DevOps Guru 스택 모두에 대한 CloudFormation 템플릿이 있다고 가정합니다.

CloudFormation 태그를 사용하여 Aurora DB 인스턴스를 지정하는 방법

1. DB 인스턴스용 CloudFormation 템플릿에서 키/값 쌍을 사용하여 태그를 정의합니다.

   다음 예제에서는 Aurora DB 인스턴스용 `Devops-guru-cfn-default`에 `my-aurora-db-instance1` 값을 할당합니다.

   ```
   MyAuroraDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBClusterIdentifier: my-aurora-db-cluster
       DBInstanceIdentifier: my-aurora-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-aurora-db-instance1
   ```

2. DevOps Guru 스택용 CloudFormation 템플릿에서 리소스 컬렉션 필터에 동일한 태그를 지정합니다.

   다음 예제에서는 `my-aurora-db-instance1` 태그 값을 사용하여 리소스에 대한 적용 범위를 제공하도록 DevOps Guru를 구성합니다.

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
         Tags:
           - AppBoundaryKey: "Devops-guru-cfn-default"
             TagValues:
             - "devopsguru-my-aurora-db-instance1"
   ```

   다음 예제는 애플리케이션 `Devops-guru-cfn-default` 경계 내의 모든 리소스에 대한 적용 범위를 제공합니다.

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::DevOpsGuru::ResourceCollection과 AWS::RDS::DBInstance를 참조하세요.

Enhanced Monitoring을 사용하여 OS 지표 모니터링

Enhanced Monitoring을 통해 DB 인스턴스의 운영 체제를 실시간으로 모니터링할 수 있습니다. 다른 프로세스 또는 스레드에서 CPU를 사용하는 방법을 확인하려면 Enhanced Monitoring 지표가 유용합니다.

주제

Enhanced Monitoring 개요

Amazon RDS는 DB 인스턴스가 실행되는 운영 체제(OS)에 대한 측정치를 실시간으로 제공합니다. 콘솔에서 RDS DB 인스턴스에 대한 모든 시스템 지표 및 프로세스 정보를 볼 수 있습니다. 각 인스턴스에 대해 모니터링할 지표를 관리하고 요구 사항에 따라 대시보드를 사용자 지정할 수 있습니다. 향상된 모니터링 지표 설명은 향상된 모니터링의 OS 지표 섹션을 참조하세요.

RDS는 지표를 확장 모니터링에서 Amazon CloudWatch Logs 계정으로 전달합니다. CloudWatch Logs에서 CloudWatch에서 지표 필터를 생성하고 CloudWatch 대시보드에 그래프를 표시할 수 있습니다. 또한 선택한 모니터링 시스템에서 CloudWatch Logs의 Enhanced Monitoring JSON 출력을 사용할 수 있습니다. 자세한 내용은 Amazon RDS FAQ의 확장 모니터링을 참조하세요.

주제

CloudWatch 지표와 Enhanced Monitoring 지표의 차이점

하이퍼바이저는 VM(가상 머신)을 만들고 실행합니다. 하이퍼바이저를 사용하는 인스턴스는 메모리와 CPU를 가상으로 공유하여 여러 게스트 VM을 지원할 수 있습니다. CloudWatch는 DB 인스턴스의 하이퍼바이저에서 CPU 사용률에 대한 지표를 수집합니다. 반면, Enhanced Monitoring은 DB 인스턴스의 에이전트에서 지표를 수집합니다.

하이퍼바이저 계층에서는 소량의 작업만 수행하므로 CloudWatch와 Enhanced Monitoring 간의 차이점을 확인할 수 있습니다. DB 인스턴스가 더 작은 인스턴스 클래스를 사용하는 경우 차이가 커질 수 있습니다. 이 시나리오에서는 단일 물리적 인스턴스의 하이퍼바이저 계층에서 더 많은 VM(가상 머신)을 관리할 수 있습니다.

향상된 모니터링 지표 설명은 향상된 모니터링의 OS 지표 섹션을 참조하세요. CloudWatch 지표에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.

Enhanced Monitoring 지표 보존

기본적으로 Enhanced Monitoring 지표는 CloudWatch Logs에서 30일간 저장됩니다. 이 보존 기간은 일반적인 CloudWatch 지표와 다릅니다.

지표가 CloudWatch Logs에 저장되는 시간을 수정하려면 CloudWatch 콘솔에서 `RDSOSMetrics` 로그 그룹의 보존을 변경하십시오. 자세한 내용은 Amazon CloudWatch Logs User Guide의 CloudWatch에서 로그 데이터 보존 기간을 변경을 참조하십시오.

Enhanced Monitoring 비용

Enhanced Monitoring 지표는 CloudWatch 지표가 아닌 CloudWatch Logs에 저장됩니다. 확장 모니터링 비용은 다음 두 가지 요인에 따라 달라집니다.

- Amazon CloudWatch Logs가 제공하는 프리 티어를 초과하는 경우에만 확장 모니터링에 대해서만 비용이 청구됩니다. 요금은 CloudWatch Logs 데이터 전송 및 스토리지 요금을 기준으로 합니다.
- RDS 인스턴스에 대해 전송되는 정보의 양은 확장 모니터링 기능에 대해 정의된 세분성에 직접적으로 비례합니다. 모니터링 간격이 작을수록 OS 측정치가 더 자주 보고되고 모니터링 비용이 증가합니다. 비용을 관리하려면 계정의 여러 인스턴스에 대해 서로 다른 세부 단위를 설정합니다.
- Enhanced Monitoring 사용 비용은 Enhanced Monitoring을 활성화한 각 DB 인스턴스에 대해 적용됩니다. 모니터링하는 DB 인스턴스의 수가 많을수록 더 많은 비용이 청구됩니다.
- 컴퓨팅 집약적인 워크로드를 지원하는 DB 인스턴스는 많은 OS 프로세스 활동이 보고되고 Enhanced Monitoring에 대한 높은 비용이 청구됩니다.

요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

Enhanced Monitoring 설정 및 활성화

Enhanced Monitoring을 사용하려면 IAM 역할을 생성한 다음 Enhanced Monitoring을 활성화해야 합니다.

주제

- Enhanced Monitoring에 대한 IAM 역할 생성
- 향상된 모니터링 설정 및 해제
- 혼동된 대리자 문제로부터 보호

Enhanced Monitoring에 대한 IAM 역할 생성

Enhanced Monitoring은 사용자를 대신하여 CloudWatch Logs에 OS 측정치 정보를 보낼 수 있는 권한이 필요합니다. AWS Identity and Access Management(IAM) 역할을 사용하여 Enhanced Monitoring에 권한을 부여합니다. 향상된 모니터링을 사용 설정할 때 이 역할을 생성하거나 미리 생성할 수 있습니다.

주제

- Enhanced Monitoring을 활성화할 때 IAM 역할 생성
- Enhanced Monitoring을 활성화하기 전에 IAM 역할 생성

Enhanced Monitoring을 활성화할 때 IAM 역할 생성

RDS 콘솔에서 Enhanced Monitoring을 활성화하면, Amazon RDS가 필요한 IAM 역할을 생성할 수 있습니다. 이 역할의 이름은 `rds-monitoring-role`입니다. RDS는 지정된 DB 인스턴스, 읽기 전용 복제본 또는 다중 AZ DB 클러스터에 이 역할을 사용합니다.

Enhanced Monitoring을 활성화할 때 IAM 역할을 생성하려면

1. 향상된 모니터링 설정 및 해제 단원의 단계를 따르십시오.
2. 역할을 선택하는 단계에서 모니터링 역할(Monitoring Role)을 기본값(Default)으로 설정합니다.

Enhanced Monitoring을 활성화하기 전에 IAM 역할 생성

Enhanced Monitoring을 활성화하기 전에, 필요한 역할을 생성할 수 있습니다. Enhanced Monitoring을 활성화할 때 새 역할의 이름을 지정합니다. AWS CLI 또는 RDS API를 사용하여 Enhanced Monitoring을 활성화할 경우 이 필수 역할을 생성해야 합니다.

확장 모니터링을 활성화하는 사용자는 `PassRole` 권한을 부여받아야 합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 AWS 서비스에 역할을 전달할 수 있는 권한 부여에 있는 예제 2를 참조하십시오.

Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면

1. https://console.aws.amazon.com에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성(Create role)을 선택합니다.
4. AWS 서비스(service) 탭을 선택한 다음 서비스 목록에서 RDS를 선택합니다.
5. RDS - 확장 모니터링(RDS - Enhanced Monitoring)과 다음(Next)을 차례로 선택합니다.
6. 권한 정책(Permissions policies)에 AmazonRDSEnhancedMonitoringRole이 표시되었는지 확인하고 다음(Next)을 선택합니다.
7. 역할 이름에 역할의 이름을 입력합니다. 예를 들면 **`emaccess`**를 입력합니다.

   사용자 역할에 대한 신뢰할 수 있는 엔터티는 monitoring.rds.amazonaws.com AWS 서비스입니다.
8. 역할 생성을 선택합니다.

향상된 모니터링 설정 및 해제

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 향상된 모니터링을 관리할 수 있습니다. 각 DB 클러스터에서 지표 수집에 대해 서로 다른 세부 수준을 설정할 수 있습니다. 콘솔에서 기존 DB 클러스터에 대한 향상된 모니터링을 전환할 수도 있습니다.

콘솔

DB 클러스터 또는 읽기 전용 복제본을 생성할 때나 DB 클러스터를 수정할 때 향상된 모니터링을 켤 수 있습니다. 향상된 모니터링을 활성화하기 위해 DB 인스턴스 또는 클러스터를 수정하는 경우 DB 인스턴스를 재부팅하지 않아도 변경 내용이 적용됩니다.

데이터베이스 페이지에서 다음 작업 중 하나를 수행할 때 RDS 콘솔에서 향상된 모니터링을 사용 설정할 수 있습니다.

- DB 클러스터 생성 - 데이터베이스 생성(Create database)을 선택합니다.
- 읽기 전용 복제본 생성(Create a read replica) - 작업(Actions)을 선택한 다음 읽기 전용 복제본 생성(Create read replica)을 선택합니다.
- DB 인스턴스 또는 DB 클러스터 수정 – 수정을 선택합니다.

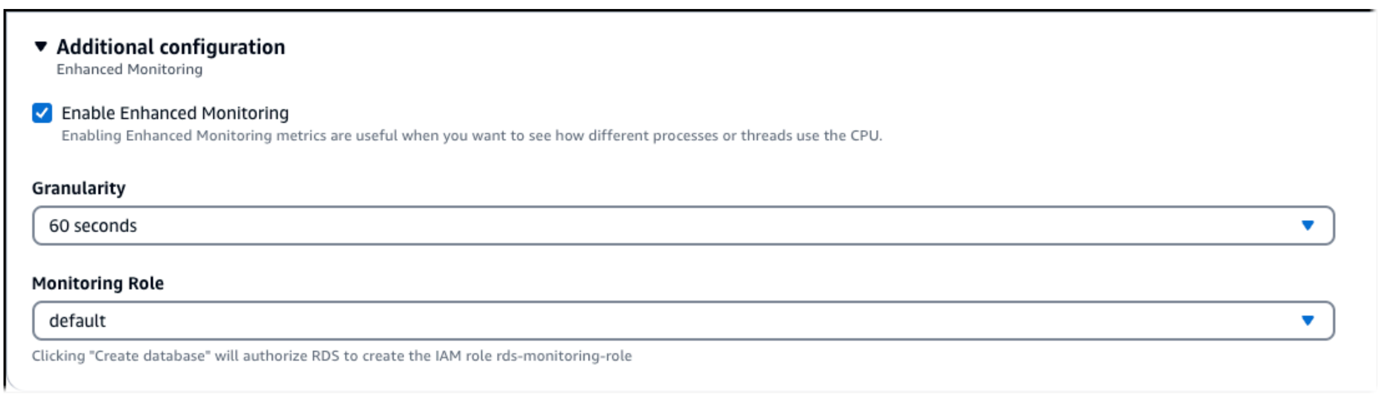
> **Note**
>
> DB 클러스터에 대해 향상된 모니터링을 활성화하면 클러스터 내의 개별 DB 인스턴스에 대해 향상된 모니터링을 관리할 수 없습니다.

DB 클러스터의 개별 DB 인스턴스에 대해 향상된 모니터링을 관리하고 있고 다른 인스턴스에 대해 세부 수준이 다른 값으로 설정된 경우, DB 클러스터는 향상된 모니터링과 관련하여 이질적입니다. 이러한 경우 클러스터 수준에서 향상된 모니터링을 관리하도록 DB 클러스터를 수정할 수 없습니다.

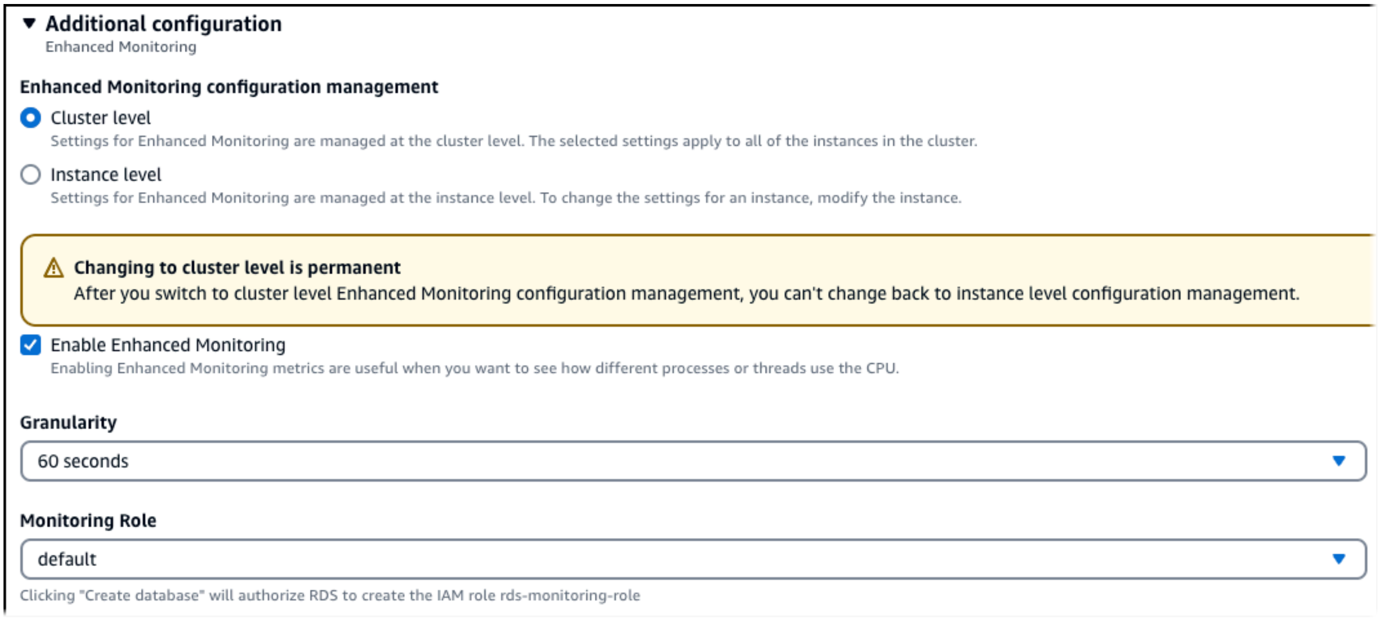
RDS 콘솔에서 향상된 모니터링 설정 또는 해제

1. 추가 구성(Additional configuration)으로 스크롤합니다.
2. 모니터링에서 DB 인스턴스 또는 읽기 전용 복제본에 대해 향상된 모니터링 활성화를 선택합니다. 클러스터 수준에서 향상된 모니터링을 활성화하면 클러스터 수준에서 향상된 모니터링 설정 및 옵션을 관리할 수 있습니다. 클러스터 수준 설정은 클러스터의 모든 DB 인스턴스에 적용됩니다. 클러스터 수준에서 향상된 모니터링을 비활성화하려면 옵션을 선택 취소합니다. 나중에 클러스터의 개별 DB 인스턴스에 대한 향상된 모니터링 설정을 수정할 수 있습니다.

   데이터베이스 생성 페이지에서 클러스터 수준에서 향상된 모니터링을 활성화하도록 선택할 수 있습니다.

   클러스터를 생성하는 동안 향상된 모니터링을 활성화하지 않으면 DB 클러스터 수정 페이지에서 클러스터를 수정할 수 있습니다.



> **Note**
>
> 클러스터 수준에서 관리되는 향상된 모니터링이 이미 있는 DB 클러스터의 개별 DB 인스턴스에 대해서는 향상된 모니터링을 관리할 수 없습니다.

3. 클러스터가 향상된 모니터링과 관련하여 이질적인 경우 클러스터 수준에서 향상된 모니터링을 관리하도록 선택할 수 없습니다. 클러스터 수준에서 향상된 모니터링을 관리하려면 각 인스턴스의 향상된 모니터링 설정이 일치하도록 변경합니다. 이제 클러스터를 수정할 때 클러스터 수준에서 향상된 모니터링을 관리하도록 선택할 수 있습니다.

4. Amazon RDS에 사용자를 대신하여 Amazon CloudWatch Logs와 통신하도록 허용하기 위해 생성한 IAM 역할에 대한 [Monitoring Role] 속성을 설정하거나, [Default]를 선택하여 RDS에서 `rds-monitoring-role` 역할을 자동으로 생성하도록 합니다.

5. 세부 수준 속성을 DB 인스턴스, DB 클러스터 또는 읽기 전용 복제본에 대한 지표가 수집되는 시점 간격(초)으로 설정합니다. [Granularity] 속성을 `1`, `5`, `10`, `15`, `30` 또는 `60` 값 중 하나로 설정할 수 있습니다.

   RDS 콘솔을 새로 고치는 최소 간격은 5초입니다. RDS 콘솔에서 단위를 1초로 설정한 경우에도 업데이트된 측정치는 5초마다 표시됩니다. CloudWatch Logs를 사용하여 1초 측정치 업데이트를 검색할 수 있습니다.

AWS CLI

AWS CLI를 사용하여 향상된 모니터링을 활성화하려면 다음 명령에서 `--monitoring-interval` 옵션을 `0` 이외의 값으로 설정하고 `--monitoring-role-arn` 옵션을 Enhanced Monitoring에 대한 IAM 역할 생성에서 생성된 역할로 설정합니다.

- create-db-cluster
- modify-db-cluster
- create-db-instance
- create-db-instance-read-replica
- modify-db-instance

`--monitoring-interval` 옵션은 확장 모니터링 지표가 수집되는 시점 간격(초)을 지정합니다. 이 옵션의 유효한 값은 `0`, `1`, `5`, `10`, `15`, `30` 및 `60`입니다.

AWS CLI를 사용하여 향상된 모니터링을 해제하려면 다음 명령에서 `--monitoring-interval` 옵션을 `0`으로 설정합니다.

Example

다음 예에서는 DB 인스턴스에 대해 향상된 모니터링을 설정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

다음 예에서는 DB 인스턴스에 대해 향상된 모니터링을 설정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

다음 예에서는 DB 인스턴스에 대해 향상된 모니터링을 설정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

RDS API를 사용하여 향상된 모니터링을 설정하려면 `MonitoringInterval` 파라미터를 `0` 이외의 값으로 설정하고 `MonitoringRoleArn` 파라미터를 Enhanced Monitoring에 대한 IAM 역할 생성에서 생성된 역할로 설정합니다. 다음 작업에서 이러한 파라미터를 설정합니다.

- CreateDBCluster
- ModifyDBCluster
- CreateDBInstance

- CreateDBInstanceReadReplica
- ModifyDBInstance

`MonitoringInterval` 파라미터는 확장 모니터링 지표가 수집되는 시점 간격(초)을 지정합니다. 유효 값은 `0`, `1`, `5`, `10`, `15`, `30`, `60`입니다.

RDS API를 사용하여 향상된 모니터링을 해제하려면 `MonitoringInterval`을 `0`으로 설정합니다.

혼동된 대리자 문제로부터 보호

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. AWS에서는 교차 서비스 가장으로 인해 혼동된 대리자 문제가 발생할 수 있습니다. 교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 직접적으로 호출할 때 발생할 수 있습니다. 직접적으로 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해 AWS에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다. 자세한 내용은 혼동된 대리자 문제를 참조하세요.

Amazon RDS가 다른 서비스에 제공할 수 있는 리소스에 대한 권한을 제한하려면 향상된 모니터링 역할에 대한 신뢰 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하는 것이 좋습니다. 두 전역 조건 컨텍스트 키를 모두 사용하는 경우 동일한 계정 ID를 사용해야 합니다.

혼동된 대리인 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다. Amazon RDS의 경우 `aws:SourceArn`을 `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*으로 설정합니다.

다음 예에서는 혼동된 대리인 문제를 방지하기 위해 신뢰 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용합니다.

JSON

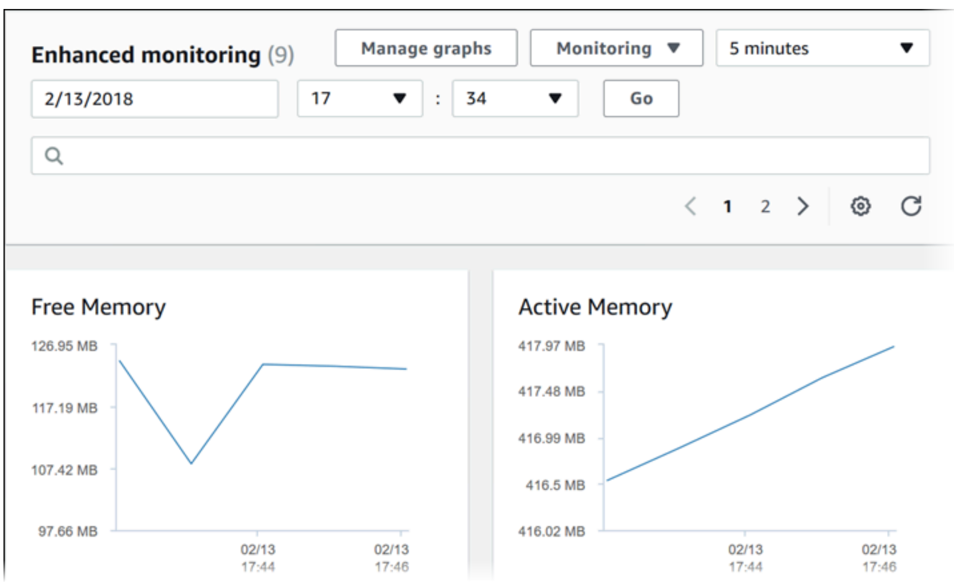
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
            "Service": "monitoring.rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole",
          "Condition": {
            "StringLike": {
              "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
            },
            "StringEquals": {
              "aws:SourceAccount": "my-account-id"
            }
          }
        }
      ]
    }
```

RDS 콘솔에서 OS 지표 보기

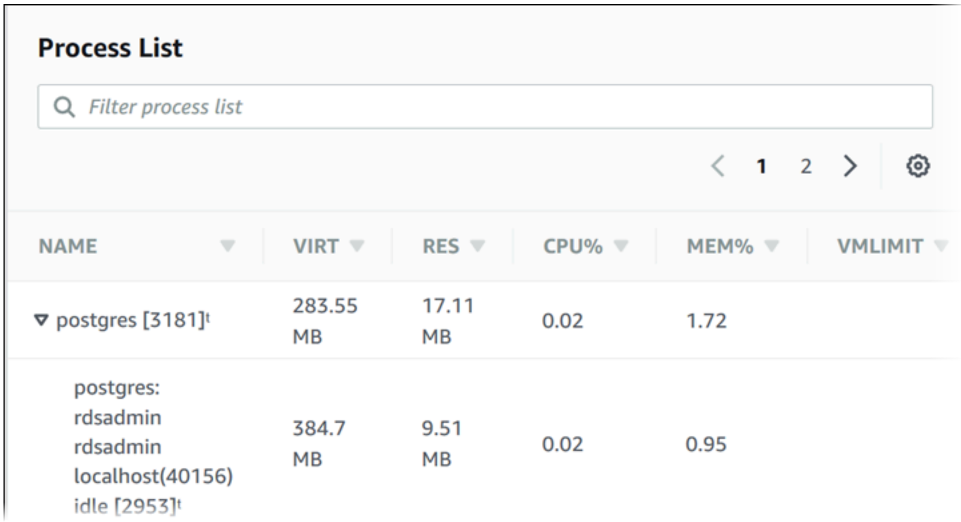
모니터링에서 확장 모니터링을 선택하면 RDS 콘솔에서 확장 모니터링이 보고하는 OS 측정치를 볼 수 있습니다.

다음 예에서는 향상된 모니터링 페이지를 보여줍니다. 향상된 모니터링 지표 설명은 향상된 모니터링의 OS 지표 섹션을 참조하세요.



DB 인스턴스에서 실행 중인 프로세스에 대한 자세한 정보를 보려면 [Monitoring]에 대해 [OS process list]를 선택합니다.

프로세스 목록 보기는 다음과 같이 표시됩니다.



프로세스 목록 보기에 표시되는 확장 모니터링 지표는 다음과 같이 구성됩니다.

- RDS child processes(RDS 하위 프로세스) – DB 인스턴스를 지원하는 RDS 프로세스(예: Amazon Aurora DB 클러스터의 경우 `aurora`,)를 요약하여 표시합니다. 프로세스 스레드는 상위 프로세스 아래에 중첩되어 표시됩니다. 프로세스 스레드에는 CPU 사용률만 표시됩니다. 다른 측정치는 프로세스의 모든 스레드에 대해 동일합니다. 콘솔에는 최대 100개의 프로세스와 스레드가 표시됩니다. 결과에는 CPU와 메모리를 소비하는 상위 프로세스 및 스레드가 함께 표시됩니다. 프로세스와 스레드가 각각 50개 이상씩 있는 경우 콘솔에는 각 범주의 상위 50개 소비자가 표시됩니다. 이 표시를 통해 성능에 가장 큰 영향을 미치고 있는 프로세스를 식별할 수 있습니다.
- RDS 프로세스 - RDS DB 인스턴스를 지원하는 데 필요한 RDS 관리 에이전트, 진단 모니터링 프로세스 및 기타 AWS 프로세스에서 사용되는 리소스를 요약하여 표시합니다.
- [OS processes] – 일반적으로 성능에 최소한의 영향만 미치는 커널 및 시스템 프로세스를 요약하여 표시합니다.

각 프로세스에 대해 나열되는 항목은 다음과 같습니다.

- VIRT – 프로세스의 가상 크기를 표시합니다.
- RES – 프로세스에서 사용 중인 실제 물리적 메모리를 표시합니다.
- CPU% – 프로세스에서 사용 중인 총 CPU 대역폭의 백분율을 표시합니다.
- MEM% – 프로세스에서 사용 중인 총 메모리의 백분율을 표시합니다.

RDS 콘솔에 표시되는 모니터링 데이터는 Amazon CloudWatch Logs으로부터 검색됩니다. CloudWatch Logs로부터 로그 스트림으로 DB 인스턴스용 측정치를 검색할 수도 있습니다. 자세한 내용은 CloudWatch Logs을 사용하여 OS 지표 보기 섹션을 참조하세요.

다음 기간 중에는 확장 모니터링 지표가 반환되지 않습니다.

- DB 인스턴스의 장애 조치 동안.
- DB 인스턴스의 인스턴스 클래스 변경(컴퓨팅 확장) 중.

Enhanced Monitoring 측정치는 데이터베이스 엔진이 재부팅되는 이유로만 DB 인스턴스의 재부팅 동안 반환됩니다. 운영 체제의 측정치는 계속 보고됩니다.

CloudWatch Logs을 사용하여 OS 지표 보기

DB 클러스터에 대한 향상된 모니터링을 활성화한 후 CloudWatch Logs를 사용하여 DB 인스턴스 또는 클러스터에 대한 측정치를 볼 수 있습니다. 각 로그 스트림에는 모니터링 중인 단일 DB 인스턴스가 표시됩니다. 로그 스트림 식별자는 DB 인스턴스 또는 DB 클러스터에 대한 리소스 식별자(`DbiResourceId`)입니다.

Enhanced Monitoring 로그 데이터를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 필요한 경우 DB cluster가 있는 AWS 리전을 선택합니다. 자세한 내용은 Amazon Web Services 일반 참조의 리전 및 엔드포인트를 참조하십시오.
3. 탐색 창에서 로그를 선택합니다.
4. 로그 그룹 목록에서 RDSOSMetrics를 선택합니다.
5. 로그 스트림 목록에서 보려는 로그 스트림을 선택합니다.

Amazon Aurora용 지표 참조

이 참조에서는 Amazon CloudWatch, Performance Insights 및 향상된 모니터링용 Amazon Aurora 지표에 대한 설명을 확인할 수 있습니다.

주제

Amazon Aurora에 대한 Amazon CloudWatch 지표

AWS/RDS 네임스페이스에는 Amazon Aurora에서 실행 중인 데이터베이스 개체에 적용되는 다음 지표가 포함되어 있습니다. 일부 지표는 Aurora MySQL, Aurora PostgreSQL 또는 둘 다에 적용됩니다. 또한 일부 지표는 DB 클러스터, 기본 DB 인스턴스, 복제본 DB 인스턴스 또는 모든 DB 인스턴스에 고유합니다.

Aurora Global Database 지표는 Aurora MySQL에서 쓰기 전달에 대한 Amazon CloudWatch 지표 및 Aurora PostgreSQL에서 쓰기 전달에 대한 Amazon CloudWatch 지표 섹션을 참조하세요. Aurora 병렬 쿼리 지표는 Aurora MySQL용 병렬 쿼리 모니터링 단원을 참조하세요.

주제

Amazon Aurora에 대한 클러스터 수준 지표

다음 테이블에서는 Aurora 클러스터에 특정한 지표에 대해 설명합니다.

지표	설명	적용 대상	단위
`AuroraGlobalDBDataTransferBytes`	Aurora Global Database에서 소스 AWS 리전에서 보조 AWS 리전으로 전송된 다시 실행 로그 데이터의 양입니다. **Note** 이 지표는 보조 AWS 리전에서만 사용할 수 있습니다.	Aurora MySQL 및 Aurora PostgreSQL	바이트
`AuroraGlobalDBProgressLag`	Aurora Global Database에서, 사용자 트랜잭션과 시스템 트랜잭션 모두에 있어서 보조 클러스터가 기본 클러스터보다 얼마나 뒤쳐져 있는지 측정합니다. **Note** 이 지표는 보조 AWS 리전에서만 사용할 수 있습니다.	Aurora MySQL 및 Aurora PostgreSQL	밀리초
`AuroraGlobalDBReplicatedWriteIO`	Aurora Global Database의 기본 AWS 리전에서 보조 AWS 리전의 클러스터 볼륨으로 복제된 쓰기 I/O 작업 수입니다. 글로벌 데이터베이스의 보조 AWS 리전에 대한 청구 계산은 `VolumeWriteIOPs` 를 사용하여 클러스터 내에서 수행된 쓰기를 설	Aurora MySQL 및 Aurora PostgreSQL	개수

지표	설명	적용 대상	단위
	명합니다. 글로벌 데이터베이스의 기본 AWS 리전에 대한 청구 계산은 `VolumeWriteIOPs` 를 사용하여 해당 클러스터 내의 쓰기 활동을 설명하고, `AuroraGlobalDBReplicatedWriteIO` 는 글로벌 데이터베이스 내에서 리전 간 복제를 설명합니다. Note 이 지표는 보조 AWS 리전에서만 사용할 수 있습니다.		
`AuroraGlobalDBReplicationLag`	Aurora Global Database의 경우 기본 AWS 리전에서 업데이트를 복제할 때의 지연 시간입니다. Note 이 지표는 보조 AWS 리전에서만 사용할 수 있습니다.	Aurora MySQL 및 Aurora PostgreSQL	밀리초

지표	설명	적용 대상	단위
`AuroraGlobalDBRPOLag`	Aurora Global Database에서는 복구 시점 목표(RPO) 지연 시간이 발생합니다. 이 지표는 사용자 트랜잭션에 있어서 보조 클러스터가 기본 클러스터보다 얼마나 뒤쳐져 있는지 측정합니다. **Note** 이 지표는 보조 AWS 리전에서만 사용할 수 있습니다.	Aurora MySQL 및 Aurora PostgreSQL	밀리초
`AuroraVolumeBytesLeftTotal`	클러스터 볼륨의 잔여 가용 공간입니다. 클러스터 볼륨이 확장됨에 따라 이 값은 감소합니다. 이 값이 0에 도달하면 클러스터에서 공간 부족 오류를 보고합니다. Aurora MySQL 클러스터가 128 tebibytes(TiB) 크기 제한에 근접하는지 여부를 감지하고 싶을 때 이 값은 `VolumeBytesUsed` 보다 더 쉽고 안정적으로 모니터링할 수 있습니다. `AuroraVolumeBytesLeftTotal` 에서는 내부 정리에 사용되는 스토리지와 스토리지 결제에 영향을 미치지 않는 기타 할당을 고려합니다.	Aurora MySQL	바이트

지표	설명	적용 대상	단위
BacktrackChangeRecordsCreationRate	DB 클러스터를 위해 5분간 생성된 역추적 변경 레코드의 수입니다.	Aurora MySQL	5분당 개수
BacktrackChangeRecordsStored	DB 클러스터에 의해 사용된 역추적 변경 레코드의 수입니다.	Aurora MySQL	개수
BackupRetentionPeriodStorageUsed	Aurora DB 클러스터의 백업 보존 기간 내 특정 시점 복원 기능을 지원하는 데 사용되는 총 백업 스토리지 양입니다. 이 양은 TotalBackupStorageBilled 지표에 의해 보고된 합계에 포함됩니다. 각 Aurora 클러스터마다 개별적으로 계산됩니다. 지침은 [Amazon Aurora 백업 스토리지 사용량 파악](#) 섹션을 참조하세요.	Aurora MySQL 및 Aurora PostgreSQL	바이트
ServerlessDatabaseCapacity	Aurora Serverless DB 클러스터의 현재 용량입니다.	Aurora MySQL 및 Aurora PostgreSQL	개수

지표	설명	적용 대상	단위
SnapshotStorageUsed	Aurora DB 클러스터에 대해 백업 보존 기간이 경과된 이후에 모든 Aurora 스냅샷에 사용된 총 백업 스토리지 양입니다. 이 양은 TotalBackupStorageBilled 지표에 의해 보고된 합계에 포함됩니다. 각 Aurora 클러스터마다 개별적으로 계산됩니다. 지침은 Amazon Aurora 백업 스토리지 사용량 파악 섹션을 참조하세요.	Aurora MySQL 및 Aurora PostgreSQL	바이트
TotalBackupStorageBilled	특정 Aurora DB 클러스터에 대한 총 백업 스토리지 양(단위: 바이트)입니다. 이 양에 대해 요금이 청구됩니다. 지표에는 BackupRetentionPeriodStorageUsed 및 SnapshotStorageUsed 지표로 측정되는 백업 스토리지가 포함됩니다. 이 지표는 각 Aurora 클러스터에 대해 별도로 계산됩니다. 지침은 Amazon Aurora 백업 스토리지 사용량 파악 섹션을 참조하세요.	Aurora MySQL 및 Aurora PostgreSQL	바이트

지표	설명	적용 대상	단위
`VolumeBytesUsed`	Aurora DB 클러스터에서 사용되는 스토리지 용량입니다. 이 값은 Aurora DB 클러스터의 요금에 영향을 미칩니다(요금에 대한 자세한 내용은 [Amazon RDS 요금 페이지](#) 참조). 이 값에는 스토리지 결제에 영향을 미치지 않는 일부 내부 스토리 할당이 반영되지 않습니다. Aurora MySQL의 경우 128TiB 스토리지 한도와 `VolumeBytesUsed` 를 비교하는 대신에 `AuroraVolumeBytesLeftTotal` 이 0에 근접하고 있는지 여부를 테스트하여 공간 부족 문제를 더 정확하게 예상할 수 있습니다. 복제본인 클러스터의 경우 이 지표 값은 복제본에서 추가되거나 변경된 데이터의 용량에 따라 달라집니다. 또한 원본 클러스터가 삭제되거나 새 복제본이 추가 또는 삭제될 때 지표가 증가하거나 감소할 수 있습니다. 자세한 내용은 [원본 클러스터 볼륨 삭제](#)을 참조하세요. Amazon RDS는 이 지표를 연속적으로 수집하지 않고	Aurora MySQL 및 Aurora PostgreSQL	바이트

지표	설명	적용 대상	단위
	일정 간격으로 수집하므로 작은 `--period` 값을 선택하는 것은 적절하지 않습니다.		

지표	설명	적용 대상	단위
VolumeReadIOPs	5분 간격 내에 클러스터 볼륨에서 요금이 부과된 읽기 I/O 작업의 수입니다. 요금이 청구된 읽기 작업은 클러스터 볼륨 수준에서 계산되며, Aurora DB 클러스터의 모든 인스턴스에 대해 집계된 후 5분 간격으로 보고됩니다. 이 값은 5분 동안의 읽기 작업 측정치 값을 사용하여 계산됩니다. 요금 부과된 읽기 작업 측정치의 값을 300로 나눠서 초당 요금 부과된 읽기 작업의 양을 확인할 수 있습니다. 예를 들어 요금 부과된 읽기 작업이 13,686을 반환할 경우 초당 요금 부과된 읽기 작업은 45(13,686 / 300 = 45.62)입니다. 버퍼 캐시에 없는 데이터베이스 페이지를 요청하는 쿼리에 대해서는 요금이 부과된 읽기 작업이 발생하므로 스토리지에서 로드해야 합니다. 쿼리 결과를 스토리지에서 읽은 후 버퍼 캐시로 로드하면 요금 부과된 작업이 급증할 수 있습니다. Amazon RDS는 이 지표를 연속적으로 수집하지 않고 일정 간격으로 수집하므로	Aurora MySQL 및 Aurora PostgreSQL	5분당 개수

지표	설명	적용 대상	단위
	작은 `--period` 값을 선택하는 것은 적절하지 않습니다. Tip Aurora MySQL 클러스터가 병렬 쿼리를 사용하는 경우 `VolumeReadIOPS` 값이 증가할 수 있습니다. 병렬 쿼리는 버퍼 풀을 사용하지 않습니다. 따라서 쿼리는 빠르지만 이렇게 최적화된 프로세싱은 읽기 작업 및 관련 비용을 증가시킬 수 있습니다.		
`VolumeWriteIOPs`	5분 간격으로 보고되는 클러스터 볼륨에 대한 평균 디스크 쓰기 I/O 작업 수입니다. 요금이 부과된 쓰기 작업 계산 방법에 대한 자세한 내용은 `VolumeReadIOPs` 단원을 참조하세요. Amazon RDS는 이 지표를 연속적으로 수집하지 않고 일정 간격으로 수집하므로 작은 `--period` 값을 선택하는 것은 적절하지 않습니다.	Aurora MySQL 및 Aurora PostgreSQL	5분당 개수

Amazon Aurora에 대한 인스턴스 수준 지표

다음 인스턴스 특정 Amazon CloudWatch 지표는 달리 명시되지 않는 한 모든 Aurora MySQL 및 Aurora PostgreSQL 인스턴스에 적용됩니다.

지표	설명	적용 대상	단위
`AbortedClients`	제대로 종료되지 않은 클라이언트 연결 수입니다.	Aurora MySQL	개수
`ActiveTransactions`	현재 Aurora 데이터베이스 인스턴스에서 1초마다 실행되고 있는 평균 트랜잭션 수입니다. 기본적으로 Aurora에서는 이 지표를 활성화하지 않습니다. 이 값의 측정을 시작하려면 특정 DB 인스턴스의 DB 파라미터 그룹에서 `innodb_monitor_enable='all'` 를 설정하세요.	Aurora MySQL	초당 개수
`ACUUtilization`	`ServerlessDatabaseCapacity` 지표의 값을 DB 클러스터의 최대 ACU 값으로 나눈 값입니다. 이 지표는 Aurora Serverless v2에만 적용됩니다.	Aurora MySQL 및 Aurora PostgreSQL	%
`AuroraBinlogReplicaLag`	이진수 로그 복제 소스를 기준으로 Aurora MySQL에서 실행 중인 이진수 로그 복제본 DB 클러스터의 지연 시간입니다. 지연이란 소스에서 복제본이 적용할 수 있는 레코드보다 빠르게 레코드를 생성하고 있음을 의미합니다.	Aurora MySQL의 기본	초

지표	설명	적용 대상	단위
	이 지표는 엔진 버전에 따라 다른 값을 보고합니다. Aurora MySQL 버전 2 MySQL `SHOW SLAVE STATUS`의 `Seconds_Behind_Master` 필드 Aurora MySQL 버전 3 `SHOW REPLICA STATUS` 이 지표를 사용하여 이진 로그 복제본 역할을 하는 클러스터의 오류 및 복제본 지연을 모니터링할 수 있습니다. 지표 값은 다음을 나타냅니다. 높은 값 복제본이 복제 소스를 지연시키고 있습니다. **`0`** 또는 **`0`**에 가까운 값 복제본 프로세스가 활성 상태입니다. **`-1`** Aurora가 지연 시간을 확인할 수 없는 경우는 복제본 설정 중 또는 복제본이 오류 상태에 있을 때 발생할 수 있습니다. 이진 로그 복제는 클러스터의 라이터 인스턴스에서만 발생		

지표	설명	적용 대상	단위
	하므로 WRITER 역할과 연결된 이 지표 버전을 사용하는 것이 좋습니다. 복제 관리에 대한 자세한 정보는 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 섹션을 참조하세요. 문제 해결에 대한 자세한 내용은 Amazon Aurora MySQL 복제 문제 단원을 참조하세요.		
`AuroraDMLRejectedMasterFull`	라이터 DB 인스턴스에서 세션이 꽉 차서 거부되는 전달된 쿼리 수입니다.	Aurora MySQL 버전 2의 기본	개수
`AuroraDMLRejectedWriterFull`	라이터 DB 인스턴스에서 세션이 꽉 차서 거부되는 전달된 쿼리 수입니다.	Aurora MySQL 버전 3의 기본	개수
`AuroraEstimatedSharedMemoryBytes`	마지막으로 구성된 폴링 간격 동안 자주 사용된 공유 버퍼 또는 버퍼 풀 메모리의 예상 양입니다.	Aurora PostgreSQL	바이트
`AuroraMemoryHealthState`	메모리 상태를 나타냅니다. `0` 값은 `NORMAL`과 같습니다. `10` 값이 `RESERVED`와 같으면 서버의 메모리 사용량이 임계 수준에 가까워지고 있음을 의미합니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.03.1 이상	Gauge

지표	설명	적용 대상	단위
`AuroraMemoryNumDeclinedSqlTotal`	메모리 부족(OOM) 회피로 인해 감소된 증분 쿼리 수입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.03.1 이상	개수
`AuroraMemoryNumKillConnTotal`	OOM 회피로 인해 종료된 증분 연결 수입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.03.1 이상	개수
`AuroraMemoryNumKillQueryTotal`	OOM 회피로 인해 종료된 증분 쿼리 수입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.03.1 이상	개수
`AuroraMillisecondsSpentInOomRecovery`	메모리 상태가 정상 상태 미만으로 떨어진 이후 경과한 시간입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.08.0 이상	밀리초
`AuroraNumOomRecoverySuccessful`	메모리 상태가 정상 상태로 복원된 횟수입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.08.0 이상	개수

지표	설명	적용 대상	단위
`AuroraNumOomRecoveryTriggered`	메모리 상태가 정상 상태 미만으로 떨어진 횟수입니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.	Aurora MySQL 버전 3.08.0 이상	개수
`AuroraOptimizedReadsCacheHitRatio`	최적화된 읽기 캐시에서 처리되는 요청의 백분율입니다. 값은 다음 공식을 사용하여 계산됩니다. `orcache_blks_hit/ (orcache_blks_hit + storage_blks_read)` `AuroraOptimizedReadsCacheHitRatio` 가 100%인 경우 최적화된 읽기 캐시에서 모든 페이지를 읽었음을 의미합니다. `AuroraOptimizedReadsCacheHitRatio` 가 0인 경우 최적화된 읽기 캐시에서 어떤 페이지도 읽지 않았음을 의미합니다.	Aurora PostgreSQL 의 기본	백분율
`AuroraReplicaLag`	Aurora 복제본의 경우, 기본 인스턴스에서 업데이트를 복제할 때의 지연 시간입니다.	Aurora MySQL 및 Aurora PostgreSQL 의 복제본	밀리초

지표	설명	적용 대상	단위
AuroraReplicaLagMaximum	기본 인스턴스와 DB 클러스터의 Aurora DB 인스턴스 사이에 발생하는 최대 지연 시간입니다. 읽기 복제본이 삭제되거나 이름이 변경되면 이전 리소스가 재활용 프로세스를 거치면서 복제 지연이 일시적으로 급증할 수 있습니다. 해당 기간 동안의 복제 지연을 정확하게 파악하려면 각 읽기 복제본 인스턴스에서 AuroraReplicaLag 지표를 모니터링하는 것이 좋습니다.	Aurora MySQL 및 Aurora PostgreSQL의 기본	밀리초
AuroraReplicaLagMinimum	기본 인스턴스와 DB 클러스터의 Aurora DB 인스턴스 사이에 발생하는 최소 지연 시간입니다.	Aurora MySQL 및 Aurora PostgreSQL의 기본	밀리초
AuroraSlowConnectionHandleCount	핸드셰이크를 시작하는 데 2초 이상 대기한 연결의 수입니다. 이 지표는 Aurora MySQL 버전 3에 적용됩니다.	Aurora MySQL	개수
AuroraSlowHandshakeCount	핸드셰이크를 시작하는 데 소요된 시간이 50밀리초 이상인 연결의 수입니다. 이 지표는 Aurora MySQL 버전 3에 적용됩니다.	Aurora MySQL	개수

지표	설명	적용 대상	단위
BacktrackWindowActual	대상 역추적 기간과 실제 역추적 기간 사이의 차이입니다.	Aurora MySQL의 기본	분
BacktrackWindowAlert	정해진 시간 동안 실제 역추적 기간이 대상 역추적 기간보다 작은 횟수.	Aurora MySQL의 기본	개수
BlockedTransactions	데이터베이스에서 1초마다 차단되는 평균 트랜잭션 수입니다.	Aurora MySQL	초당 개수
BufferCacheHitRatio	버퍼 캐시에서 처리하는 요청 비율.	Aurora MySQL 및 Aurora PostgreSQL	%
CommitLatency	엔진 및 스토리지가 커밋 작업을 완료하는 데 걸린 평균 시간입니다.	Aurora MySQL 및 Aurora PostgreSQL	밀리초
CommitThroughput	초당 커밋 작업의 평균 수.	Aurora MySQL 및 Aurora PostgreSQL	초당 개수
ConnectionAttempts	성공 여부에 관계없이 인스턴스에 연결하려는 시도 횟수입니다.	Aurora MySQL	개수

지표	설명	적용 대상	단위
`CPUCreditBalance`	5분 간격으로 보고되는 인스턴스가 누적된 CPU 크레딧 수입니다. 이 지표는 DB 인스턴스가 지정된 속도로 기준 성능 수준을 넘어 얼마나 오래 버스트할 수 있는지 결정하는 데 사용됩니다. 이 지표는 다음 인스턴스 클래스에만 적용됩니다. • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3` 및 `db.t4g` • Aurora PostgreSQL: `db.t3` 및 `db.t4g` Note T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 유형] 섹션을 참조하세요. 시작 크레딧은 Amazon RDS에서도 Amazon EC2에서와 동일한 방식으로 작동합니다. 자세한 내용은 Linux 인스턴스	Aurora MySQL 및 Aurora PostgreSQL	개수

지표	설명	적용 대상	단위
	용 Amazon Elastic Compute Cloud 사용 설명서의 시작 크레딧을 참조하세요.		
`CPUCreditUsage`	5분 간격으로 보고되는 지정된 기간 동안 소비된 CPU 크레딧 수입니다. 이 지표는 DB 인스턴스에 할당된 가상 CPU에서 처리 명령에 실제 CPU를 사용한 시간을 측정합니다. 이 지표는 다음 인스턴스 클래스에만 적용됩니다. • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3` 및 `db.t4g` • Aurora PostgreSQL: `db.t3` 및 `db.t4g` **Note** T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.	Aurora MySQL 및 Aurora PostgreSQL	개수

지표	설명	적용 대상	단위
`CPUSurplusCreditBalance`	`CPUCreditBalance` 값이 0일 때 무제한 인스턴스에서 소비된 잉여 크레딧의 수 입니다. 획득한 CPU 크레딧에 따라 `CPUSurplusCreditBalance` 값이 청산됩니다. 잉여 크레딧의 수가 인스턴스가 24시간 동안 획득할 수 있는 최대 크레딧 수를 초과한 경우 최대 값 이상으로 소비된 잉여 크레딧은 추가 요금으로 부과됩니다. CPU 크레딧 지표는 5분 간격으로만 제공됩니다.	Aurora MySQL 및 Aurora PostgreSQL	크레딧 (vCPU-분)

지표	설명	적용 대상	단위
`CPUSurplusCreditsCharged`	획득한 CPU 크레딧으로 청산되지 않는 소비 잉여 크레딧의 수로, 추가 요금으로 부과됩니다. 소비된 잉여 크레딧은 다음이 발생할 때 요금이 부과됩니다. • 소비한 잉여 크레딧이 인스턴스가 24시간 동안 획득할 수 있는 최대 크레딧 수를 초과하는 경우. 해당 시간이 끝날 때 최대 값 이상으로 소비한 잉여 크레딧에 요금이 부과됩니다. • 인스턴스가 중지 또는 종료된 경우. • 인스턴스가 `unlimited` 에서 `standard`로 전환됩니다. CPU 크레딧 지표는 5분 간격으로만 제공됩니다.	Aurora MySQL 및 Aurora PostgreSQL	크레딧 (vCPU-분)
`CPUUtilization`	Aurora DB 인스턴스의 CPU 사용률입니다.	Aurora MySQL 및 Aurora PostgreSQL	%

<table>
<tr><th>지표</th><th>설명</th><th>적용 대상</th><th>단위</th></tr>
<tr><td><code>DatabaseConnections</code></td><td>데이터베이스 인스턴스에 대한 클라이언트 네트워크 연결 수입니다.

지표 값에 다음이 포함되지 않기 때문에 데이터베이스 세션 수가 지표 값보다 클 수 있습니다.

• 더 이상 네트워크에 연결되어 있지 않지만 데이터베이스에 의해 정리되지 않은 세션
• 데이터베이스 엔진에 의해 자체 용도로 생성된 세션
• 데이터베이스 엔진의 병렬 실행 기능에 의해 생성된 세션
• 데이터베이스 엔진 작업 스케줄러에 의해 생성된 세션
• Amazon Aurora 연결</td><td>Aurora MySQL 및 Aurora PostgreSQL</td><td>개수</td></tr>
<tr><td><code>DDLLatency</code></td><td>생성 요청, 변경 요청, 삭제 요청 등의 평균 요청 소요 시간입니다.</td><td>Aurora MySQL</td><td>밀리초</td></tr>
<tr><td><code>DDLThroughput</code></td><td>초당 평균 DDL 요청 수</td><td>Aurora MySQL</td><td>초당 개수</td></tr>
<tr><td><code>Deadlocks</code></td><td>데이터베이스 1초마다 발생하는 평균 교착 수</td><td>Aurora MySQL 및 Aurora PostgreSQL</td><td>초당 개수</td></tr>
</table>

지표	설명	적용 대상	단위
`DeleteLatency`	삭제 작업의 평균 소요 시간입니다.	Aurora MySQL	밀리초
`DeleteThroughput`	초당 삭제 쿼리의 평균 수.	Aurora MySQL	초당 개수
`DiskQueueDepth`	디스크 액세스를 대기 중인 읽기/쓰기 요청 수.	Aurora MySQL 및 Aurora PostgreSQL	개수
`DMLLatency`	삽입, 업데이트 및 삭제의 평균 소요 시간입니다.	Aurora MySQL	밀리초
`DMLThroughput`	초당 평균 삽입, 업데이트 및 삭제 수.	Aurora MySQL	초당 개수
`EngineUptime`	인스턴스 실행 시간입니다.	Aurora MySQL 및 Aurora PostgreSQL	초
`FreeableMemory`	사용 가능한 RAM 크기입니다. Aurora MySQL 및 Aurora PostgreSQL 데이터베이스의 경우 이 지표는 `/proc/meminfo` 의 `MemAvailable` 필드 값을 보고합니다.	Aurora MySQL 및 Aurora PostgreSQL	바이트
`FreeEphemeralStorage`	사용 가능한 임시 NVMe 스토리지 크기입니다.	Aurora PostgreSQL	바이트

지표	설명	적용 대상	단위
`FreeLocalStorage`	사용 가능한 로컬 스토리지 양입니다. 다른 DB 엔진과 달리 Aurora DB 인스턴스의 경우, 이 지표는 각 DB 인스턴스에 사용 가능한 스토리지 크기를 보고합니다. 이 값은 DB 인스턴스 클래스에 좌우됩니다(요금에 대한 자세한 내용은 Amazon RDS 요금 페이지 참조). DB 인스턴스 클래스를 큰 것으로 선택하면 인스턴스의 여유 스토리지 공간을 늘릴 수 있습니다. (Aurora Serverless v2에는 적용되지 않습니다.)	Aurora MySQL 및 Aurora PostgreSQL	바이트
`InsertLatency`	삽입 작업의 평균 소요 시간입니다.	Aurora MySQL	밀리초
`InsertThroughput`	초당 평균 커밋 작업 수입니다.	Aurora MySQL	초당 개수
`LoginFailures`	초당 평균 로그인 실패 수	Aurora MySQL	초당 개수

지표	설명	적용 대상	단위
`MaximumUsedTransactionIDs`	가장 오랫동안 vacuum되지 않은 트랜잭션 ID의 경과 시간(트랜잭션 수). 이 값이 2,146,483,648(2^31 - 1,000,000)에 도달하면 트랜잭션 ID의 랩어라운드를 방지할 목적으로 데이터베이스가 읽기 전용 모드로 강제 전환됩니다. 자세한 내용은 PostgreSQL 설명서의 P트랜잭션 ID 랩어라운드 실패 방지를 참조하세요.	Aurora PostgreSQL	개수
`NetworkReceiveThroughput`	Aurora DB 클러스터의 각 인스턴스가 클라이언트에서 수신하는 네트워크 처리량입니다. 이 처리량에서 Aurora DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트 수(콘솔에 초당 메가바이트가 표시됨)
`NetworkThroughput`	Aurora DB 클러스터의 각 인스턴스가 클라이언트에서 수신하고 클라이언트로 전송하는 네트워크 처리량입니다. 이 처리량에서 Aurora DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트

지표	설명	적용 대상	단위
NetworkTransmitThroughput	Aurora DB 클러스터의 각 인스턴스가 클라이언트로 전송하는 네트워크 처리량입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트 수(콘솔에 초당 메가바이트가 표시됨)
NumBinaryLogFiles	생성된 binlog 파일 수입니다.	Aurora MySQL	개수
OldestReplicationSlotLag	수신된 WAL(Write-Ahead Log) 데이터를 기준으로 가장 지연된 복제본의 지연 크기.	Aurora PostgreSQL	바이트
PurgeBoundary	InnoDB 제거가 허용되는 최대 트랜잭션 수입니다. 이 지표가 장기간 발전하지 않으면 장기 실행 트랜잭션으로 인해 InnoDB 제거가 차단되었음을 나타내는 좋은 지표로 해석할 수 있습니다. 이를 조사하려면 Aurora MySQL DB 클러스터의 활성 트랜잭션을 확인합니다.	Aurora MySQL 버전 2, 버전 2.11 이상 Aurora MySQL 버전 3, 버전 3.08 이상	개수
PurgeFinishedPoint	InnoDB 제거가 수행되는 최대 트랜잭션 수입니다. 이 지표를 사용하면 InnoDB 제거가 얼마나 빠르게 진행되고 있는지 확인할 수 있습니다.	Aurora MySQL 버전 2, 버전 2.11 이상 Aurora MySQL 버전 3, 버전 3.08 이상	개수

지표	설명	적용 대상	단위
`Queries`	초당 실행 쿼리의 평균 수.	Aurora MySQL	초당 개수
`RDSToAuroraPostgreSQLReplicaLag`	기본 RDS PostgreSQL 인스턴스에서 클러스터의 다른 노드로 업데이트를 복제할 때 지연 시간입니다.	Aurora PostgreSQL 의 복제본	초
`ReadIOPS`	초당 평균 디스크 I/O 연산 수이지만 1분의 간격을 두고 읽기와 쓰기를 따로 보고합니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 개수
`ReadIOPSEphemeralStorage`	임시 NVMe 스토리지에 대한 평균 디스크 읽기 I/O 작업 수입니다.	Aurora PostgreSQL	초당 개수
`ReadLatency`	디스크 I/O 연산당 평균 처리 시간.	Aurora MySQL 및 Aurora PostgreSQL	초
`ReadLatencyEphemeralStorage`	임시 NVMe 스토리지의 디스크 읽기 I/O 작업당 소요된 평균 시간입니다.	Aurora PostgreSQL	밀리초
`ReadThroughput`	초당 디스크에서 읽은 평균 바이트 수입니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트
`ReadThroughputEphemeralStorage`	임시 NVMe 스토리지의 초당 디스크에서 읽은 평균 바이트 수입니다.	Aurora PostgreSQL	초당 바이트
`ReplicationSlotDiskUsage`	복제 슬롯 파일에 사용된 디스크 공간의 양입니다.	Aurora PostgreSQL	바이트

지표	설명	적용 대상	단위
`ResultSetCacheHitRatio`	Resultset 캐시에서 처리되는 평균 요청 수입니다.	Aurora MySQL 버전 2	백분율
`RollbackSegmentHistoryListLength`	삭제 표시 레코드로 커밋된 트랜잭션을 기록하는 실행 취소 로그입니다. 이러한 레코드는 InnoDB 제거 작업에 의해 처리되도록 예약됩니다.	Aurora MySQL	개수
`RowLockTime`	InnoDB 테이블에 대한 행 잠금을 획득하는 데 걸린 총 시간입니다.	Aurora MySQL	밀리초
`SelectLatency`	선택한 작업의 평균 소요 시간입니다.	Aurora MySQL	밀리초
`SelectThroughput`	초당 평균 select 쿼리 수	Aurora MySQL	초당 개수
`ServerlessDatabaseCapacity`	Aurora Serverless DB 클러스터의 현재 용량입니다.	Aurora MySQL 및 Aurora PostgreSQL	개수
`StorageNetworkReceiveThroughput`	DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신하는 네트워크 처리량입니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트
`StorageNetworkThroughput`	Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템과 송수신하는 네트워크 처리량입니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트

지표	설명	적용 대상	단위
StorageNetworkTransmitThroughput	Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송하는 네트워크 처리량입니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트
SumBinaryLogSize	binlog 파일의 총 크기입니다.	Aurora MySQL	바이트
SwapUsage	사용되는 스왑 공간 크기입니다. 이 지표는 다음 DB 인스턴스 클래스에 사용할 수 없습니다. • db.r3.*, db.r4.* 및 db.r7g.*(Aurora MySQL) • db.r7g.* (Aurora PostgreSQL)	Aurora MySQL 및 Aurora PostgreSQL	바이트
TempStorageIOPS	DB 인스턴스에 연결된 로컬 스토리지에 대한 읽기 및 쓰기 모두의 IOPS 수입니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. 이 지표는 Aurora Serverless v2에만 적용됩니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 개수
TempStorageThroughput	DB 인스턴스와 연결된 로컬 스토리지에서 전송되는 데이터의 양입니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. 이 지표는 Aurora Serverless v2에만 적용됩니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트

지표	설명	적용 대상	단위
`TransactionAgeMaximum`	가장 오랫동안 실행 중인 활성 트랜잭션의 연령입니다.	Aurora MySQL 버전 3, 버전 3.08 이상	초
`TransactionLogsDiskUsage`	Aurora PostgreSQL DB 인스턴스에서 트랜잭션 로그가 차지하는 디스크 공간입니다. 이 지표는 Aurora PostgreSQL이 논리적 복제 또는 AWS Database Migration Service 를 사용하는 경우에만 생성됩니다. 기본적으로 Aurora PostgreSQL에서는 트랜잭션 로그가 아닌 로그 레코드를 사용합니다. 트랜잭션 로그를 사용하지 않는 경우 이 지표의 값은 `-1`입니다.	Aurora PostgreSQL 의 기본	바이트
`TruncateFinishedPoint`	실행 취소 잘라내기가 수행될 때의 트랜잭션 식별자입니다.	Aurora MySQL 버전 2, 버전 2.11 이상 Aurora MySQL 버전 3, 버전 3.08 이상	개수
`UpdateLatency`	업데이트 작업의 평균 소요 시간입니다.	Aurora MySQL	밀리초
`UpdateThroughput`	초당 평균 업데이트 수입니다.	Aurora MySQL	초당 개수

지표	설명	적용 대상	단위
`WriteIOPS`	초당 생성된 Aurora 스토리지 쓰기 레코드 수입니다. 데이터베이스에 의해 생성된 로그 레코드 수보다 많거나 적습니다. 이는 8K 페이지 쓰기에 해당하지 않으며 전송된 네트워크 패킷에 해당하지 않습니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 개수
`WriteIOPSEphemeralStorage`	임시 NVMe 스토리지에 대한 평균 디스크 쓰기 I/O 작업 수입니다.	Aurora PostgreSQL	초당 개수
`WriteLatency`	디스크 I/O 연산당 평균 처리 시간.	Aurora MySQL 및 Aurora PostgreSQL	초
`WriteLatencyEphemeralStorage`	임시 NVMe 스토리지의 디스크 쓰기 I/O 작업당 소요된 평균 시간입니다.	Aurora PostgreSQL	밀리초
`WriteThroughput`	초당 영구 스토리지에 기록된 평균 바이트 수입니다.	Aurora MySQL 및 Aurora PostgreSQL	초당 바이트
`WriteThroughputEphemeralStorage`	임시 NVMe 스토리지에 디스크에 쓰여진 초당 평균 바이트 수입니다.	Aurora PostgreSQL	초당 바이트

Amazon Aurora에 대한 Amazon CloudWatch 사용량 지표

Amazon CloudWatch의 `AWS/Usage` 네임스페이스에는 Amazon RDS 서비스 할당량에 대한 계정 수준 사용량 지표가 포함됩니다. CloudWatch는 모든 AWS 리전의 사용량 지표를 자동으로 수집합니다.

자세한 내용은 Amazon CloudWatch 사용 설명서에서 CloudWatch 사용량 지표를 참조하세요. 할당량에 대한 자세한 내용은 Service Quotas 사용 설명서의 Amazon Aurora에 대한 할당량 및 제약 조건 및 할당량 증가 요청을 참조하세요.

지표	설명	단위*
AuthorizationsPerDBSecurityGroup	AWS 계정의 DB 보안 그룹당 수신 규칙 수입니다. 사용된 값은 계정의 DB 보안 그룹에서 가장 많은 수신 규칙 수입니다. 계정의 다른 DB 보안 그룹은 수신 규칙 수가 더 적을 수 있습니다.	개수
CustomEndpointsPerDBCluster	AWS 계정의 DB 클러스터당 사용자 지정 엔드포인트 수입니다. 사용된 값은 계정의 DB 클러스터에서 가장 많은 사용자 지정 엔드포인트 수입니다. 계정의 다른 DB 클러스터는 사용자 지정 엔드포인트 수가 더 적을 수 있습니다.	개수
DBClusterParameterGroups	AWS 계정에서 DB 클러스터 파라미터 그룹의 수입니다. 개수에는 기본 파라미터 그룹이 제외됩니다.	개수
DBClusterRoles	AWS Identity and Access Management의 DB 클러스터당 연결된 AWS 계정(IAM) 역할 수입니다. 사용된 값은 계정의 DB 클러스터에 연결된 IAM 역할이 가장 많은 수입니다. 계정의 다른 DB 클러스터는 연결된 IAM 역할 수가 더 적을 수 있습니다.	개수
DBClusters	AWS 계정에서 Amazon Aurora DB 클러스터의 수입니다.	개수
DBInstanceRoles	AWS Identity and Access Management의 DB 인스턴스당 연결된 AWS 계정(IAM) 역할 수입니다. 사용된 값은 계정의 DB 인스턴스에 연결된 IAM 역할이 가장 많은 수입니다. 계정의 다른 DB 인스턴스는 연결된 IAM 역할 수가 더 적을 수 있습니다.	개수
DBInstances	AWS 계정에서 DB 인스턴스의 수입니다.	개수
DBParameterGroups	AWS 계정에서 DB 파라미터 그룹의 수입니다. 개수에는 기본 DB 파라미터 그룹이 제외됩니다.	개수

지표	설명	단위*
DBSubnetGroups	AWS 계정에서 DB 서브넷 그룹의 수입니다. 개수에는 기본 서브넷 그룹이 제외됩니다.	개수
EventSubs criptions	AWS 계정의 이벤트 알림 구독 수입니다.	개수
Integrations	AWS 계정에 있는 Amazon Redshift와의 제로 ETL 통합 수입니다.	개수
ManualClu sterSnapshots	AWS 계정에서 수동으로 생성된 DB 클러스터 스냅샷의 수입니다. 개수에는 잘못된 스냅샷이 제외됩니다.	개수
OptionGroups	AWS 계정에서 옵션 그룹의 수입니다. 개수에는 기본 옵션 그룹이 제외됩니다.	개수
Proxies	AWS 계정의 RDS 프록시 수입니다.	개수
ReadRepli casPerMaster	계정의 DB 인스턴스당 읽기 전용 복제본 수입니다. 사용된 값은 계정의 DB 인스턴스에서 가장 많은 읽기 복제본 수입니다. 계정의 다른 DB 인스턴스는 읽기 전용 복제본 수가 더 적을 수 있습니다.	개수
ReservedD BInstances	AWS 계정에서 예약된 DB 인스턴스의 수입니다. 개수에는 수명 종료되거나 거절된 인스턴스가 제외합니다.	개수
SubnetsPe rDBSubnetGroup	AWS 계정의 DB 서브넷 그룹당 서브넷 개수입니다. 계정의 DB 서브넷 그룹에 가장 많은 서브넷 수입니다. 계정의 다른 DB 서브넷 그룹은 서브넷 수가 더 적을 수 있습니다.	개수

> **Note**
>
> Amazon RDS는 CloudWatch에 사용량 지표 단위를 게시하지 않습니다. 단위는 설명서에만 표시됩니다.

Aurora에 대한 Amazon CloudWatch 측정기준 목록

다음 표의 차원을 사용하여 Amazon RDSAurora 지표 데이터를 필터링할 수 있습니다.

차원	다음에 대해 요청된 데이터를 필터링합니다.
`DBInstanceIdentifier`	특정 DB 인스턴스
`DBClusterIdentifier`	특정 Aurora DB 클러스터
`DBClusterIdentifier, Role`	인스턴스 역할(WRITER/LEDERE)별로 지표를 집계하는 특정 Aurora DB 클러스터 예를 들어 클러스터에 속하는 모든 READER 인스턴스에 대한 지표를 집계할 수 있습니다.
`DbClusterIdentifier, EngineName`	특정 Aurora DB 클러스터와 엔진 이름의 조합입니다. 예를 들어 클러스터 `ams1` 및 엔진 `aurora`에 대한 `VolumeReadIOPs` 지표를 볼 수 있습니다.
`DatabaseClass`	데이터베이스 클래스의 모든 인스턴스 예를 들어 데이터베이스 클래스 `db.r5.large` 에 속하는 모든 인스턴스에 대한 지표를 집계할 수 있습니다.
`EngineName`	식별된 엔진 이름만 예를 들어 엔진 이름이 `aurora-postgresql` 인 모든 인스턴스에 대한 지표를 집계할 수 있습니다.
`SourceRegion`	지정된 리전만 예를 들어 `us-east-1` 리전의 모든 DB 인스턴스에 대한 지표를 집계할 수 있습니다.

Amazon RDS 콘솔의 Aurora 지표 가용성

Amazon Aurora가 제공하는 모든 지표를 Amazon RDS 콘솔에서 사용 가능하지는 않습니다. 그러나 AWS CLI 및 CloudWatch API와 같은 기타 도구를 이용하여 볼 수 있습니다. 또한 Amazon RDS 콘솔에서 제공하는 일부 측정치는 특정 인스턴스 클래스에서만 또는 다른 이름 및 측정 단위와 함께 표시됩니다.

주제

- 지난 시간 보기에서 사용 가능한 Aurora 지표
- 특정 사례에서 사용할 수 있는 Aurora 지표

- [사용할 수 없는 Aurora 지표]

지난 시간 보기에서 사용 가능한 Aurora 지표

Amazon RDS 콘솔의 최신 지표 보기에서 범주화된 Aurora 지표의 하위 세트를 확인할 수 있습니다. 다음 표에는 Aurora 인스턴스의 Amazon RDS 콘솔에 표시된 카테고리 및 관련 지표가 나열되어 있습니다.

Category	측정치

SQL	`ActiveTransactions`
	`BlockedTransactions`
	`BufferCacheHitRatio`
	`CommitLatency`
	`CommitThroughput`
	`DatabaseConnections`
	`DDLLatency`
	`DDLThroughput`
	`Deadlocks`
	`DMLLatency`
	`DMLThroughput`
	`LoginFailures`
	`ResultSetCacheHitRatio`
	`SelectLatency`
	`SelectThroughput`

시스템	`AuroraReplicaLag`

Category	측정치
	AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage (Aurora Serverless v2에는 적용되지 않습니다.) NetworkReceiveThroughput
배포	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

특정 사례에서 사용할 수 있는 Aurora 지표

또한 일부 Aurora 지표는 특정 인스턴스 클래스에서만, 또는 DB 인스턴스에서만, 또는 다른 이름 및 측정 단위와 함께 표시됩니다.

- `CPUCreditBalance` 및 `CPUCreditUsage` 메트릭은 Aurora MySQL `db.t2` 인스턴스 클래스 및 Aurora PostgreSQL `db.t3` 인스턴스 클래스에 한하여 표시됩니다.
- 다른 이름과 함께 표시되는 다음 지표는 다음과 같이 나열됩니다.

측정치	표시 이름
AuroraReplicaLagMaximum	최대 복제 지연 시간
AuroraReplicaLagMinimum	최소 복제 지연 시간

측정치	표시 이름
DDLThroughput	DDL
NetworkReceiveThroughput	네트워크 처리량
VolumeBytesUsed	[요금 부과됨] 사용한 볼륨 바이트
VolumeReadIOPs	[요금 부과됨] 볼륨 읽기 IOPS
VolumeWriteIOPs	[요금 부과됨] 볼륨 쓰기 IOPS

- 다음 지표는 전체 Aurora DB 클러스터에 적용되지만 Amazon RDS 콘솔에서 Aurora DB 클러스터의 DB 인스턴스를 확인할 때만 표시됩니다.
  - `VolumeBytesUsed`
  - `VolumeReadIOPs`
  - `VolumeWriteIOPs`
- 다음 측정치는Amazon RDS 콘솔에서 바이트가 아니라 MB 단위로 표시됩니다.
  - `FreeableMemory`
  - `FreeLocalStorage`
  - `NetworkReceiveThroughput`
  - `NetworkTransmitThroughput`
- 다음 지표는 Aurora 최적화된 읽기를 사용하는 Aurora PostgreSQL DB 클러스터에 적용됩니다.
  - `AuroraOptimizedReadsCacheHitRatio`
  - `FreeEphemeralStorage`
  - `ReadIOPSEphemeralStorage`
  - `ReadLatencyEphemeralStorage`
  - `ReadThroughputEphemeralStorage`
  - `WriteIOPSEphemeralStorage`
  - `WriteLatencyEphemeralStorage`
  - `WriteThroughputEphemeralStorage`

사용할 수 없는 Aurora 지표

다음 Aurora 지표는 Amazon RDS 콘솔에서 사용할 수 없습니다.

- `AuroraBinlogReplicaLag`
- `DeleteLatency`
- `DeleteThroughput`
- `EngineUptime`
- `InsertLatency`
- `InsertThroughput`
- `NetworkThroughput`
- `Queries`
- `UpdateLatency`
- `UpdateThroughput`

Amazon RDS 성능 개선 도우미를 위한 Amazon CloudWatch 지표

성능 개선 도우미는 Amazon CloudWatch에 일부 지표를 자동으로 게시합니다. 동일한 데이터는 성능 개선 도우미에서 쿼리할 수 있지만 CloudWatch에 지표가 있으면 CloudWatch 경보를 더 쉽게 추가할 수 있습니다. 또한 기존 CloudWatch 대시보드에 지표를 더 쉽게 추가할 수 있습니다.

측정치	설명
DBLoad	데이터베이스에 대한 활성 세션 수입니다. 일반적으로 사용자는 활성 세션의 평균 개수에 대한 데이터를 원합니다. 성능 개선 도우미에서 이 데이터는 `db.load.avg` 로 쿼리됩니다.
DBLoadCPU	대기 이벤트 유형이 CPU인 활성 세션 수입니다. 성능 개선 도우미에서 이 데이터는 `db.load.avg` 로 쿼리되며 대기 이벤트 유형인 CPU를 기준으로 필터링됩니다.
DBLoadNonCPU	대기 이벤트 유형이 CPU가 아닌 활성 세션 수입니다.
DBLoadRelativeToNumVCPUs	데이터베이스의 가상 CPU 수에 대한 DB 부하의 비율입니다.

> **Note**
>
> 이러한 메트릭은 DB 인스턴스에 로드가 있는 경우에만 CloudWatch에 게시됩니다.

CloudWatch 콘솔, AWS CLI 또는 CloudWatch API를 사용하여 이러한 지표를 검사할 수 있습니다. 특수 지표 수학 함수를 사용하여 다른 성능 개선 도우미 카운터 지표를 검사할 수도 있습니다. 자세한 내용은 CloudWatch에서 다른 성능 개선 도우미 카운터 지표 쿼리 섹션을 참조하세요.

예를 들어, get-metric-statistics 명령을 실행하여 DBLoad 지표에 대한 통계를 가져올 수 있습니다.

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

이 예에서는 다음과 비슷한 출력이 생성됩니다.

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
```

```
"Unit": "None",
"Average": 1.5
},
{
"Timestamp": "2021-07-19T21:32:00Z",
"Unit": "None",
"Average": 1.8
},
{
"Timestamp": "2021-07-19T21:29:00Z",
"Unit": "None",
"Average": 3.0
},
{
"Timestamp": "2021-07-19T21:33:00Z",
"Unit": "None",
"Average": 2.4
}
],
"Label": "DBLoad"
}
```

CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?를 참조하세요.

CloudWatch에서 다른 성능 개선 도우미 카운터 지표 쿼리

> **Note**
>
> Database Insights의 고급 모드를 사용 설정하면 Amazon RDS는 성능 개선 도우미 카운터 지표를 Amazon CloudWatch에 게시합니다. Database Insights에서는 DB_PERF_INSIGHTS 지표 수학 함수를 사용할 필요가 없습니다. CloudWatch Database Insights 대시보드를 사용하여 성능 개선 도우미 카운터 지표에 대한 경보를 검색, 쿼리 및 설정할 수 있습니다.

CloudWatch에서 RDS 성능 개선 도우미 지표를 쿼리하고, 경보를 표시하고, 그래프를 작성할 수 있습니다. CloudWatch의 DB_PERF_INSIGHTS 지표 수학 함수를 사용하여 DB 클러스터 에 대한 정보에 액세스할 수 있습니다. 이 함수를 사용하면 CloudWatch에 직접 보고되지 않는 성능 개선 도우미 지표를 사용하여 새 시계열을 생성할 수 있습니다.

CloudWatch 콘솔의 지표 선택 화면에서 수학 추가 드롭다운 메뉴를 클릭하여 새로운 지표 수학 함수를 사용할 수 있습니다. 이를 사용하여 성능 개선 도우미 지표 또는 CloudWatch와 성능 개선 도우미 지표의 조합에 대한 경보 및 그래프를 생성할 수 있습니다. 여기에는 1분 미만의 지표에 대한 고해상도 경보가 포함됩니다. `get-metric-data` 요청에 지표 수학 표현식을 포함하여 프로그래밍 방식으로 함수를 사용할 수도 있습니다. 자세한 내용은 지표 수학 구문 및 함수 및 AWS 데이터베이스의 성능 개선 도우미 카운터 지표에 대한 경보 생성을 참조하세요.

성능 개선 도우미 카운터

카운터 지표는 Performance Insights 대시보드의 운영 체제 및 데이터베이스 성능 지표 입니다. 이 정보와 데이터베이스 로드를 연관 지으면 성능 문제를 식별하고 분석하는 데 도움이 됩니다. 지표에 통계 함수를 추가해야 지표 값을 가져올 수 있습니다. 예를 들어 `os.memory.active` 지표에 지원되는 함수는 `.avg`, `.min`, `.max`, `.sum`, `.sample_count`입니다.

카운터 지표는 1분에 한 번씩 수집됩니다. OS 지표 수집은 향상된 모니터링을 켰는지, 껐는지에 따라 달라집니다. 향상된 모니터링이 꺼져 있는 경우 OS 지표는 1분마다 한 번씩 수집됩니다. 향상된 모니터링이 켜져 있는 경우 OS 지표는 선택한 기간 동안 수집됩니다. 향상된 모니터링에 대한 자세한 내용은 향상된 모니터링 설정 및 해제 섹션을 참조하세요.

주제

- 성능 개선 도우미 운영 체제 카운터
- Aurora MySQL용 성능 개선 도우미 카운터
- Aurora PostgreSQL용 성능 개선 도우미 카운터

성능 개선 도우미 운영 체제 카운터

다음 운영 체제 카운터는 `os`로 접두사가 붙으며, Aurora PostgreSQL 및 Aurora MySQL에 대한 성능 개선 도우미를 사용할 수 있습니다.

`ListAvailableResourceMetrics` API를 사용하여 DB 인스턴스에 대해 지원되는 카운터 지표 목록을 확인할 수 있습니다. 자세한 내용은 Amazon RDS 성능 개선 도우미 API 참조 안내서의 ListAvailableResourceMetrics를 참조하세요.

카운터	유형	Unit	지표	설명
활성	메모리	KB	os.memory.active	할당된 메모리의 양(KB)

카운터	유형	Unit	지표	설명
버퍼	메모리	KB	os.memory.buffers	스토리지 디바이스에 쓰기 이전에 I/O 요청을 버퍼링하는 데 사용되는 메모리의 양(KB)
캐시됨	메모리	KB	os.memory.cached	파일 시스템 기반 I/O를 캐시하는 데 사용되는 메모리의 양(킬로바이트 단위)입니다.
DB 캐시	메모리	바이트	os.memory.db.cache	tmpfs(shmem)를 포함한 데이터베이스 프로세스별 페이지 캐시에 사용되는 메모리의 양(바이트 단위)입니다.
DB 상주 세트 크기	메모리	바이트	os.memory.db.residentSetSize	tmpfs(shmem)를 제외한 데이터베이스 프로세스별 익명 및 스왑 캐시에 사용되는 메모리의 양(바이트 단위)입니다.
DB 스왑	메모리	바이트	os.memory.db.swap	데이터베이스 프로세스별 스왑에 사용되는 메모리의 양(바이트 단위)입니다.

카운터	유형	Unit	지표	설명
더티	메모리	KB	os.memory.dirty	수정되었지만 스토리지의 관련 데이터 블록에 기록되지 않은 RAM의 메모리 페이지 양(KB)
무료	메모리	KB	os.memory.free	할당되지 않은 메모리의 양(KB)
사용 가능한 방대한 페이지	메모리	Pages	os.memory .hugePagesFree	사용 가능한 방대한 페이지 수입니다. 방대한 페이지는 Linux 커널의 기능입니다.
예약된 방대한 페이지	메모리	Pages	os.memory .hugePagesRsvd	커밋된 방대한 페이지의 수
방대한 페이지 크기	메모리	KB	os.memory .hugePagesSize	각 방대한 페이지 단위의 크기(KB)
초과된 방대한 페이지	메모리	Pages	os.memory .hugePagesSurp	총계 대비 사용 가능한 초과 방대한 페이지 수
방대한 페이지 합계	메모리	Pages	os.memory .hugePagesTotal	방대한 페이지의 총 수입니다.
비활성	메모리	KB	os.memory .inactive	가장 적게 사용되는 메모리 페이지의 양(KB)

카운터	유형	Unit	지표	설명
매핑됨	메모리	KB	os.memory.mapped	프로세스 주소 공간 내에 메모리 매핑되는 총 파일 시스템 콘텐츠 양(KB)
메모리 부족 처리 수	메모리	종료	os.memory.outOfMemoryKillCount	마지막 수집 간격 동안 발생한 OOM 처리 수입니다.
페이지 테이블	메모리	KB	os.memory.pageTables	페이지 표에 사용된 메모리의 양(KB)
슬래브	메모리	KB	os.memory.slab	재사용 가능한 커널 데이터 구조의 양(KB)
합계	메모리	KB	os.memory.total	총 메모리 양(KB)
Writeback	메모리	KB	os.memory.writeback	RAM에서 지원 스토리지에 아직 기록 중인 더티 페이지의 양(KB)
게스트	CPU 사용률	백분율	os.cpuUtilization.guest	게스트 프로그램에서 사용 중인 CPU의 비율
유휴	CPU 사용률	백분율	os.cpuUtilization.idle	유휴 상태인 CPU의 비율
Irq	CPU 사용률	백분율	os.cpuUtilization.irq	소프트웨어 인터럽트에서 사용 중인 CPU의 비율

카운터	유형	Unit	지표	설명
Nice	CPU 사용률	백분율	os.cpuUtilization.nice	가장 낮은 우선순위로 실행 중인 프로그램에서 사용 중인 CPU의 비율
도용	CPU 사용률	백분율	os.cpuUtilization.steal	다른 가상 머신에서 사용 중인 CPU의 비율
시스템	CPU 사용률	백분율	os.cpuUtilization.system	커널에서 사용 중인 CPU의 비율
합계	CPU 사용률	백분율	os.cpuUtilization.total	사용 중인 CPU의 총 비율입니다. 이 값에는 nice 값이 포함됩니다.
User	CPU 사용률	백분율	os.cpuUtilization.user	사용자 프로그램에서 사용 중인 CPU의 비율
Wait	CPU 사용률	백분율	os.cpuUtilization.wait	I/O 액세스를 대기 중인 동안 사용되지 않은 CPU의 비율
Aurora 스토리지 Aurora 스토리지 바이트 Rx	디스크 I/O	초당 바이트	os.diskIO.auroraStorage.auroraStorageBytesRx	Aurora 스토리지에서 초당 수신된 바이트 수입니다.

카운터	유형	Unit	지표	설명
Aurora 스토리지 Aurora 스토리지 바이트 Tx	디스크 I/O	초당 바이트	os.diskIO.auroraStorage.auroraStorageBytesTx	Aurora 스토리지에 초당 업로드된 바이트 수입니다.
Aurora 스토리지 디스크 대기열 길이	디스크 I/O	요청	os.diskIO.auroraStorage.diskQueueDepth	Aurora 스토리지 디스크 대기열 길이입니다.
Aurora 스토리지 읽기 IO PS	디스크 I/O	초당 요청	os.diskIO.auroraStorage.readIOsPS	초당 읽기 작업 수
Aurora 스토리지 읽기 지연 시간	디스크 I/O	밀리초	os.diskIO.auroraStorage.readLatency	Aurora 스토리지에 대한 읽기 I/O 요청의 평균 지연 시간(밀리초)
Aurora 스토리지 읽기 처리량	디스크 I/O	초당 바이트	os.diskIO.auroraStorage.readThroughput	DB 클러스터에 요청할 때 사용되는 네트워크 처리량(초당 바이트)
Aurora 스토리지 쓰기 IO PS	디스크 I/O	초당 요청	os.diskIO.auroraStorage.writeIOsPS	초당 쓰기 작업 수
Aurora 스토리지 쓰기 지연 시간	디스크 I/O	밀리초	os.diskIO.auroraStorage.writeLatency	Aurora 스토리지에 대한 쓰기 I/O 요청의 평균 지연 시간(밀리초)

카운터	유형	Unit	지표	설명
Aurora 스토리지 쓰기 처리량	디스크 I/O	초당 바이트	os.diskIO .auroraSt orage.wri teThroughput	DB 클러스터의 응답에 사용되는 네트워크 처리량 (초당 바이트)
평균 대기열 길이 Rdstemp	디스크 I/O	요청	os.diskIO .rdstemp. avgQueueLen	I/O 디바이스의 대기열에서 대기 중인 요청 수입니다.
평균 요청 크기 Rdstemp	디스크 I/O	요청	os.diskIO .rdstemp. avgReqSz	I/O 디바이스의 대기열에서 대기 중인 요청 수입니다.
대기 Rdstemp	디스크 I/O	밀리초	os.diskIO .rdstemp.await	대기열 시간과 서비스 시간을 포함하여 요청에 응답하는 데 필요한 시간(밀리초)
읽기 IO PS Rdstemp	디스크 I/O	요청	os.diskIO .rdstemp. readIOsPS	초당 읽기 작업 수
읽기 KB Rdstemp	디스크 I/O	KB	os.diskIO .rdstemp.readKb	읽은 총 KB 수
읽기 KB PS Rdstemp	디스크 I/O	초당 킬로바이트	os.diskIO .rdstemp. readKbPS	초당 읽은 KB 수
Rdstemp Rrqm PS	디스크 I/O	초당 요청	os.diskIO .rdstemp.rrqmPS	초당 대기 중인 병합 읽기 요청 수

카운터	유형	Unit	지표	설명
Rdstemp TPS	디스크 I/O	초당 트랜잭션 수	os.diskIO .rdstemp.tps	초당 I/O 트랜잭션 수
유틸리티 Rdstemp	디스크 I/O	백분율	os.diskIO .rdstemp.util	요청이 발급된 CPU 시간의 비율
쓰기 IOPS Rdstemp	디스크 I/O	초당 요청	os.diskIO .rdstemp. writeIOsPS	초당 쓰기 작업 수
쓰기 KB Rdstemp	디스크 I/O	KB	os.diskIO .rdstemp.writeKb	기록한 총 KB 수
쓰기 KB PS Rdstemp	디스크 I/O	초당 킬로바이트	os.diskIO .rdstemp. writeKbPS	초당 기록한 KB 수
Rdstemp Wrqm PS	디스크 I/O	초당 요청	os.diskIO .rdstemp. wrqmPS	초당 대기 중인 병합 쓰기 요청 수
차단됨	업무	업무	os.tasks.blocked	차단되는 작업 수
실행 중	업무	업무	os.tasks.running	실행 중인 작업 수
Sleeping	업무	업무	os.tasks.sleeping	절전 상태인 작업 수
Stopped	업무	업무	os.tasks.stopped	중단된 작업 수
합계	업무	업무	os.tasks.total	총 작업 수
좀비	업무	업무	os.tasks.zombie	상위 작업은 활성화되었지만 비활성 상태인 하위 작업 수

카운터	유형	Unit	지표	설명
1개	로드 평균(분)	프로세스	os.loadAverageMinute.one	마지막 1분 동안 CPU 시간을 요청한 프로세스 수
15	로드 평균(분)	프로세스	os.loadAverageMinute.fifteen	마지막 15분 동안 CPU 시간을 요청한 프로세스 수
5	로드 평균(분)	프로세스	os.loadAverageMinute.five	마지막 5분 동안 CPU 시간을 요청한 프로세스 수
캐시됨	Swap	KB	os.swap.cached	캐시 메모리로 사용된 스왑 메모리의 양(KB)
무료	Swap	KB	os.swap.free	사용 가능한 스왑 메모리 양(KB)
있음	Swap	KB	os.swap.in	디스크에서 스왑된 메모리 양(KB)
Out	Swap	KB	os.swap.out	디스크로 스왑된 총 메모리 양(KB)
합계	Swap	KB	os.swap.total	사용 가능한 총 스왑 메모리의 양(킬로바이트 단위)입니다.
최대 파일	파일 시스템	파일	os.fileSys.maxFiles	파일 시스템에 대해 생성될 수 있는 최대 파일 수
사용된 파일	파일 시스템	파일	os.fileSys.usedFiles	파일 시스템의 파일 수

카운터	유형	Unit	지표	설명
사용된 파일(%)	파일 시스템	파일	os.fileSys.usedFilePercent	사용 중인 사용 가능한 파일의 비율
사용 비율	파일 시스템	백분율	os.fileSys.usedPercent	사용 중인 파일 시스템 디스크 공간의 비율
사용됨	파일 시스템	KB	os.fileSys.used	파일 시스템에서 파일에 사용된 디스크 공간의 양(KB)
합계	파일 시스템	KB	os.fileSys.total	파일 시스템에 사용 가능한 총 디스크 공간(KB)
Rx	Network	초당 바이트	os.network.rx	초당 수신된 바이트 수
Tx	Network	초당 바이트	os.network.tx	초당 업로드된 바이트 수
ACU 사용률	일반	백분율	os.general.acuUtilization	구성된 최대 용량에서 현재 용량의 백분율입니다.
최대 구성 ACU	일반	ACU	os.general.maxConfiguredAcu	사용자가 구성한 최대 용량으로, 단위는 Aurora 용량 단위(ACU)입니다.
최소 구성 ACU	일반	ACU	os.general.minConfiguredAcu	사용자가 구성한 최소 용량(ACU 단위)입니다.

카운터	유형	Unit	지표	설명
Num VCPU	일반	vCPU	os.general.numVCPUs	DB 인스턴스의 가상 CPU(vCPU) 수
서버리스 데이터베이스 용량	일반	ACU	os.general.serverlessDatabaseCapacity	인스턴스의 현재 용량(ACU 단위)입니다.

Aurora MySQL용 성능 개선 도우미 카운터

Aurora MySQL용 성능 개선 도우미에서는 다음 데이터베이스 카운터를 사용할 수 있습니다.

주제

- Aurora MySQL용 기본 카운터
- 기본이 아닌 Aurora MySQL용 카운터

Aurora MySQL용 기본 카운터

기본 지표는 Amazon Aurora가 아닌 데이터베이스 엔진에 의해 정의됩니다. MySQL 설명서의 서버 상태 변수에서 이러한 기본 지표에 대한 정의를 확인하실 수 있습니다.

카운터	유형	Unit	측정치
Com_analyze	SQL	초당 쿼리 수	db.SQL.Com_analyze
Com_optimize	SQL	초당 쿼리 수	db.SQL.Com_optimize
Com_select	SQL	초당 쿼리 수	db.SQL.Com_select
Innodb_rows_deleted	SQL	초당 행	db.SQL.Innodb_rows_deleted

카운터	유형	Unit	측정치
Innodb_rows_inserted	SQL	초당 행	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	초당 행	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	초당 행	db.SQL.Innodb_rows_updated
쿼리	SQL	초당 쿼리 수	db.SQL.Queries
질문	SQL	초당 쿼리 수	db.SQL.Questions
Select_full_join	SQL	초당 쿼리 수	db.SQL.Select_full_join
Select_full_range_join	SQL	초당 쿼리 수	db.SQL.Select_full_range_join
Select_range	SQL	초당 쿼리 수	db.SQL.Select_range
Select_range_check	SQL	초당 쿼리 수	db.SQL.Select_range_check
Select_scan	SQL	초당 쿼리 수	db.SQL.Select_scan
Slow_queries	SQL	초당 쿼리 수	db.SQL.Slow_queries
Sort_merge_passes	SQL	초당 쿼리 수	db.SQL.Sort_merge_passes
Sort_range	SQL	초당 쿼리 수	db.SQL.Sort_range
Sort_rows	SQL	초당 쿼리 수	db.SQL.Sort_rows

카운터	유형	Unit	측정치
Sort_scan	SQL	초당 쿼리 수	db.SQL.Sort_scan
Total_query_time	SQL	밀리초	db.SQL.Total_query_time
Table_locks_immediate	잠금	초당 요청	db.Locks.Table_locks_immediate
Table_locks_waited	잠금	초당 요청	db.Locks.Table_locks_waited
Innodb_row_lock_time	잠금	밀리초(평균)	db.Locks.Innodb_row_lock_time
Aborted_clients	Users	Connections	db.Users.Aborted_clients
Aborted_connects	Users	Connections	db.Users.Aborted_connects
연결	Users	연결	db.Users.Connections
External_threads_connected	Users	연결	db.Users.External_threads_connected
max_connections	Users	연결	db.Users.max_connections
Threads_connected	Users	연결	db.Users.Threads_connected
Threads_created	Users	Connections	db.Users.Threads_created
Threads_running	Users	Connections	db.Users.Threads_running
Created_tmp_disk_tables	Temp	초당 테이블	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	초당 테이블	db.Temp.Created_tmp_tables

카운터	유형	Unit	측정치
Innodb_buffer_pool_pages_data	Cache	페이지	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	페이지	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	Cache	초당 페이지	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	초당 페이지	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	테이블	db.Cache.Opened_tables
Opened_table_definitions	Cache	테이블	db.Cache.Opened_table_definitions
Qcache_hits	Cache	쿼리	db.Cache.Qcache_hits

기본이 아닌 Aurora MySQL용 카운터

기본이 아닌 카운터 지표는 Amazon RDS가 정의하는 카운터입니다. 기본이 아닌 지표는 특정 쿼리를 통해 얻는 지표일 수 있습니다. 기본이 아닌 지표는 파생 지표일 수 있습니다. 이 경우 비율, 적중률 또는 지연 시간에 대한 계산 시 2개 이상의 기본 카운터가 사용됩니다.

카운터	유형	Unit	지표	설명	정의
active_transactions	트랜잭션	db.Transactions.active_transaction	총 활성 트랜잭션입니다.	`SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX`	
innodb_buffer_pool_hit_rate	Cache	db.Cache.innoDB	InnoDB가 버퍼	`100 * innodb_buffer_pool`	

카운터	유형	Unit	지표	설명	정의
		ffer_pool_hit_rat	풀에서 InnoDB 가 충족할 수 있었던 읽기의 비율입니다.	`_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)`	
innodb_buffer_pool_hits	Cache	초당 페이지	db.Cache.innoDB_buffer_pool_hits	InnoDB가 버퍼 풀에서 충족할 수 있었던 읽기의 수입니다.	`innodb_buffer_pool_read_requests - innodb_buffer_pool_reads`

카운터	유형	Unit	지표	설명	정의
innodb_buffer_pool_usage	Cache	백분율	db.Cache.innoDB_buffer_pool_usage	데이터(페이지)를 포함하는 InnoDB 버퍼 풀의 비율입니다. **Note** 압축된 테이블을 사용하는 경우 이 값은 달라질 수 있습니다. 자세한 내용은 MySQL 설명서에서 [서버 상태 변수](https://dev.mysql.com/doc/refman/8.0/en/server-status-variables.html)의 `Innodb_buffer_pool_pages_data` 및 `Innodb_buffer_pool_pages_total` 에 대한 정보를 참조하세요.	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`

카운터	유형	Unit	지표	설명	정의
innodb_deadlocks	잠금	db.Lock innodb_ adlocks	교착 상태의 총 개수입니다.	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`	
innodb_lock_timeouts	잠금	db.Lock innodb_ ck_time ts	시간을 초과한 교착 상태의 총 개수입니다.	`SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'`	
innodb_row_lock_waits	잠금	db.Lock innodb_ w_lock_ its	대기의 원인이 된 행 잠금의 총 개수입니다.	`SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME ='lock_row_lock_waits'`	

카운터	유형	Unit	지표	설명	정의
innodb_rows_changed	SQL	db.SQL.Innodb_rows_changed	총 InnoDB 행 연산입니다.	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated	
query_cache_hit_rate	Cache	백분율	db.Cache.query_cache_hit_rate	MySQL 결과 집합 캐시(쿼리 캐시)의 적중률입니다.	Qcache_hits / (QCache_hits + Com_select) * 100
temp_disk_tables_percent	임시 파일	db.Temp.temp_disk_tables_percent	문을 실행할 때 서버가 디스크에 생성한 임시 테이블의 비율입니다.	(db.Temp.Created_tmp_disk_tables / db.Temp.Created_tmp_tables) * 100	

카운터	유형	Unit	지표	설명	정의
trx_rseg_history_len	트랜잭션	없음	db.Transactions.trx_rseg_history_len	다중 버전 동시성 제어를 구현하기 위해 InnoDB 트랜잭션 시스템에서 유지 관리하는 커밋된 트랜잭션의 실행 취소 로그 페이지 목록입니다. 실행 취소 로그 레코드에 대한 자세한 내용은 MySQL 설명서의 https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html 섹션을 참조하세요.	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`

Aurora PostgreSQL용 성능 개선 도우미 카운터

Aurora PostgreSQL용 성능 개선 도우미에서는 다음 데이터베이스 카운터를 사용할 수 있습니다.

주제

- Aurora PostgreSQL용 기본 카운터
- 기본이 아닌 Aurora PostgreSQL용 카운터

Aurora PostgreSQL용 기본 카운터

기본 지표는 Amazon Aurora가 아닌 데이터베이스 엔진에 의해 정의됩니다. PostgreSQL 설명서의 통계 보기에서 이러한 기본 지표에 대한 정의를 확인하실 수 있습니다.

카운터	유형	Unit	측정치
tup_deleted	SQL	초당 튜플 수	db.SQL.tup_deleted

카운터	유형	Unit	측정치
tup_fetched	SQL	초당 튜플 수	db.SQL.tup_fetched
tup_inserted	SQL	초당 튜플 수	db.SQL.tup_inserted
tup_returned	SQL	초당 튜플 수	db.SQL.tup_returned
tup_updated	SQL	초당 튜플 수	db.SQL.tup_updated
blks_hit	Cache	초당 블록 수	db.Cache.blks_hit
buffers_alloc	Cache	초당 블록 수	db.Cache.buffers_alloc
buffers_checkpoint	체크포인트	초당 블록 수	db.Checkpoint.buffers_checkpoint
checkpoints_req	체크포인트	분당 체크포인트	db.Checkpoint.checkpoints_req
checkpoint_sync_time	체크포인트	체크포인트당 밀리초	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	체크포인트	분당 체크포인트	db.Checkpoint.checkpoints_timed
checkpoint_write_time	체크포인트	체크포인트당 밀리초	db.Checkpoint.checkpoint_write_time
maxwritten_clean	체크포인트	분당 Bgwriter 클린 스톱	db.Checkpoint.maxwritten_clean
deadlocks	동시성	분당 교착 상태의 수	db.Concurrency.deadlocks
blk_read_time	I/O	밀리초	db.IO.blk_read_time
blks_read	I/O	초당 블록 수	db.IO.blks_read

카운터	유형	Unit	측정치
buffers_backend	I/O	초당 블록 수	db.IO.buffers_backend
buffers_backend_fsync	I/O	초당 블록 수	db.IO.buffers_backend_fsync
buffers_clean	I/O	초당 블록 수	db.IO.buffers_clean
temp_bytes	Temp	초당 바이트	db.Temp.temp_bytes
temp_files	Temp	분당 파일 수	db.Temp.temp_files
xact_commit	트랜잭션	초당 커밋 수	db.Transactions.xact_commit
xact_rollback	트랜잭션	초당 롤백 수	db.Transactions.xact_rollback
numbackends	User	Connections	db.User.numbackends
archived_count	WAL	분당 파일 수	db.WAL.archived_count

기본이 아닌 Aurora PostgreSQL용 카운터

기본이 아닌 카운터 지표는 Amazon Aurora가 정의하는 카운터입니다. 기본이 아닌 지표는 특정 쿼리를 통해 얻는 지표일 수 있습니다. 기본이 아닌 지표는 파생 지표일 수 있습니다. 이 경우 비율, 적중률 또는 지연 시간에 대한 계산 시 2개 이상의 기본 카운터가 사용됩니다.

카운터	유형	Unit	지표	설명	정의
checkpoint_sync_latency	체크포인트	밀리초	db.Checkpoint.checkpoint_sync_latency	파일이 디스크에 동기화되는 체크포인트 처리 중 일부에서 소요된 총 시간입니다.	`checkpoint_sync_time / (checkpoints_timed`

카운터	유형	Unit	지표	설명	정의
					+ checkpoints_req)
checkpoint_write_latency	체크포인트	밀리초	db.Checkpoint.checkpoint_write_latency	파일이 디스크에 쓰이는 체크포인트 처리 중 일부에서 소요된 총 시간입니다.	checkpoint_write_time / (checkpoints_timed + checkpoints_req)
local_blks_read	I/O	블록	db.IO.local_blks_read	읽은 총 로컬 블록 수입니다.	해당 사항 없음
local_blk_read_time	I/O	밀리초	db.IO.local_blk_read_time	`track_io_timing` 이 활성화된 경우 로컬 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.	해당 사항 없음
num_blocked_sessions	잠금	db.Locks.num_blocked_sessions	차단된 세션 수입니다.	–	
orcache_blks_hit	I/O	쿼리	db.IO.orcache_blks_hit	최적화된 읽기 캐시에서 발생한 공유 블록 히트의 총 수입니다.	해당 사항 없음

카운터	유형	Unit	지표	설명	정의
orcache_blk_read_time	I/O	밀리초	db.IO.orcache_blk_read_time	`track_io_timing` 이 활성화된 경우 최적화된 읽기 캐시에서 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.	해당 사항 없음
read_latency	I/O	밀리초	db.IO.read_latency	이 인스턴스의 백엔드에서 데이터 파일 블록을 읽는 데 소요된 시간입니다.	`blk_read_time / blks_read`
storage_blks_read	I/O	블록	db.IO.storage_blks_read	Aurora 스토리지에서 읽은 총 공유 블록 수입니다.	해당 사항 없음
storage_blk_read_time	I/O	밀리초	db.IO.storage_blk_read_time	`track_io_timing` 이 활성화된 경우 Aurora 스토리지에서 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.	해당 사항 없음

카운터	유형	Unit	지표	설명	정의
num_blocked_sessions	잠금	db.Locks.num_blocked_sessions	차단된 세션 수입니다.	–	
active_count	상태	세션	db.state.active_count	active 상태의 세션 수	해당 사항 없음
idle_count	상태	세션	db.state.idle_count	idle 상태의 세션 수	해당 사항 없음
idle_in_transaction_aborted_count	상태	세션	db.state.idle_in_transaction_aborted_count	idle in transaction (aborted) 상태의 세션 수	해당 사항 없음
idle_in_transaction_count	상태	세션	db.state.idle_in_transaction_count	idle in transaction 상태의 세션 수	해당 사항 없음
idle_in_transaction_max_time	State	초	db.state.idle_in_transaction_max_time	idle in transaction 상태에서 가장 오래 실행되는 트랜잭션의 지속 시간(초)	해당 사항 없음
logical_reads	SQL	블록	db.SQL.logical_reads	조회하고 읽은 총 블록 수입니다.	blks_hit + blks_read
queries_started	SQL	쿼리	db.SQL.queries	시작된 쿼리의 수	해당 사항 없음

카운터	유형	Unit	지표	설명	정의
queries_finished	SQL	쿼리	db.SQL.queries	완료된 쿼리 수	해당 사항 없음
total_query_time	SQL	밀리초	db.SQL.total_query_time	문을 실행하는 데 걸린 총 시간(밀리초)	해당 사항 없음
active_transactions	트랜잭션	트랜잭션	db.Transactions.active_transactions	활성 트랜잭션의 수	해당 사항 없음
blocked_transactions	트랜잭션	트랜잭션	db.Transactions.blocked_transactions	차단된 트랜잭션의 수	해당 사항 없음
commit_latency	트랜잭션	마이크로초	db.Transactions.commit_latency	커밋 작업의 평균 소요 시간입니다.	`db.Transactions.duration_commits / db.Transactions.xact_commit`
duration_commits	트랜잭션	밀리초	db.Transactions.duration_commits	마지막 1분간 소요된 총 트랜잭션 시간(밀리초)	해당 사항 없음
max_used_xact_ids	트랜잭션	트랜잭션	db.Transactions.max_used_xact_ids	vacuum되지 않은 트랜잭션의 수	해당 사항 없음

카운터	유형	Unit	지표	설명	정의
oldest_inactive_logical_replication_slot_xid_age	트랜잭션	길이	db.Transactions.oldest_inactive_logical_replication_slot_xid_age	비활성 논리 복제 슬롯에서 가장 오래된 트랜잭션의 기간입니다.	해당 사항 없음
oldest_active_logical_replication_slot_xid_age	트랜잭션	길이	db.Transactions.oldest_active_logical_replication_slot_xid_age	활성 논리 복제 슬롯에서 가장 오래된 트랜잭션의 기간입니다.	해당 사항 없음
oldest_reader_feedback_xid_age	트랜잭션	길이	db.transactions.oldest_reader_feedback_xid_age	Aurora 리더 인스턴스 또는 Aurora 글로벌 DB 리더 인스턴스에서 장기 실행 중인 트랜잭션 중 가장 오래된 트랜잭션의 기간입니다.	해당 사항 없음
oldest_prepared_transaction_xid_age	트랜잭션	길이	db.Transactions.oldest_prepared_transaction_xid_age	가장 오래된 준비된 트랜잭션의 기간입니다.	해당 사항 없음

카운터	유형	Unit	지표	설명	정의
oldest_running_transaction_xid_age	트랜잭션	길이	db.Transactions.oldest_running_transaction_xid_age	실행 중인 트랜잭션 중 가장 오래된 트랜잭션의 기간입니다.	해당 사항 없음
max_connections	사용자	Users	db.User.Max_Connections	`max_connections` 파라미터에 구성된 데이터베이스에 허용되는 최대 연결 수	해당 사항 없음
total_auth_attempts	사용자	Users	db.User.total_auth_attempts	이 인스턴스에 대한 연결 시도 수	해당 사항 없음
archive_failed_count	WAL	분당 파일 수	db.WAL.archive_failed_count	WAL 파일 보관 실패 횟수(분당 파일 수)	해당 사항 없음

성능 개선 도우미에 대한 SQL 통계

SQL 통계(SQL statistics)는 성능 개선 도우미에서 수집하는 SQL 쿼리에 대한 성능 관련 지표입니다. 성능 개선 도우미는 쿼리가 실행되는 초당 통계와 각 SQL 호출에 대한 통계를 수집합니다. SQL 통계는 선택한 시간 범위의 평균입니다.

SQL 다이제스트는 주어진 패턴을 갖지만 반드시 동일한 리터럴 값을 가질 필요는 없는 모든 쿼리의 합성입니다. 다이제스트는 리터럴 값을 물음표로 바꿉니다. 예: `SELECT * FROM emp WHERE lname= ?`. 이 다이제스트에는 다음 하위 쿼리가 구성될 수 있습니다.

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

모든 엔진은 다이제스트 쿼리에 대한 SQL 통계를 지원합니다.

이 기능에 대한 리전, DB 엔진 및 인스턴스 클래스 지원 정보는 성능 개선 도우미 기능에 대한 Amazon Aurora DB 엔진, 리전 및 인스턴스 클래스 지원 섹션을 참조하세요.

주제

- Aurora MySQL에 대한 SQL 통계
- Aurora PostgreSQL에 대한 SQL 통계

Aurora MySQL에 대한 SQL 통계

Aurora MySQL은 다이제스트 수준에서만 SQL 통계를 수집합니다. 명령문 수준에는 통계가 표시되지 않습니다.

주제

- Aurora MySQL에 대한 다이제스트 통계
- Aurora MySQL에 대한 초당 통계
- Aurora MySQL에 대한 호출당 통계
- Aurora MySQL에 대한 기본 통계

Aurora MySQL에 대한 다이제스트 통계

성능 개선 도우미는 `events_statements_summary_by_digest` 테이블에서 SQL 다이제스트 통계를 수집합니다. `events_statements_summary_by_digest` 테이블은 데이터베이스에 의해 관리됩니다.

다이제스트 테이블에는 제거 정책이 없습니다. 테이블이 가득 차면 AWS Management Console에 다음 메시지가 표시됩니다.

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

이 상황에서는 Aurora MySQL은 SQL 쿼리를 추적하지 않습니다. 이 문제를 해결하기 위해 성능 개선 도우미는 다음 조건이 모두 충족되면 자동으로 다이제스트 테이블을 자릅니다.

- 테이블이 꽉 찼습니다.
- 성능 개선 도우미가 성능 스키마를 자동으로 관리합니다.

자동으로 관리하려면 `performance_schema` 파라미터를 `0`으로 설정하고 [소스(Source)]를 `user` 이외의 값으로 설정해야 합니다. 성능 개선 도우미가 성능 스키마를 자동으로 관리하지 않는 경우 Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요 섹션을 참조하세요.

AWS CLI에서 describe-db-parameters 명령을 실행하여 파라미터 값의 소스를 확인합니다.

Aurora MySQL에 대한 초당 통계

Aurora MySQL DB 클러스터에 다음 SQL 통계를 사용할 수 있습니다.

측정치	Unit
db.sql_tokenized.stats.count_star_per_sec	초당 호출 수
db.sql_tokenized.stats.sum_timer_wait_per_sec	초당 평균 지연 시간(단위: ms)
db.sql_tokenized.stats.sum_select_full_join_per_sec	초당 전체 조인 선택
db.sql_tokenized.stats.sum_select_range_check_per_sec	초당 범위 검사 선택
db.sql_tokenized.stats.sum_select_scan_per_sec	초당 스캔 선택
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	초당 병합 패스 정렬
db.sql_tokenized.stats.sum_sort_scan_per_sec	초당 스캔 정렬
db.sql_tokenized.stats.sum_sort_range_per_sec	초당 범위 정렬
db.sql_tokenized.stats.sum_sort_rows_per_sec	초당 행 정렬
db.sql_tokenized.stats.sum_rows_affected_per_sec	초당 영향을 받는 행

측정치	Unit
db.sql_tokenized.stats.sum_rows_examined_per_sec	초당 검사된 행
db.sql_tokenized.stats.sum_rows_sent_per_sec	초당 전송된 행
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	초당 생성된 임시 디스크 테이블
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	초당 생성된 임시 테이블
db.sql_tokenized.stats.sum_lock_time_per_sec	초당 잠금 시간(ms)

Aurora MySQL에 대한 호출당 통계

다음 지표에서는 SQL 문의 호출당 통계를 제공합니다.

측정치	Unit
db.sql_tokenized.stats.sum_timer_wait_per_call	호출당 평균 지연 시간(단위: ms)
db.sql_tokenized.stats.sum_select_full_join_per_call	호출당 전체 조인 선택
db.sql_tokenized.stats.sum_select_range_check_per_call	호출당 범위 검사 선택
db.sql_tokenized.stats.sum_select_scan_per_call	호출당 스캔 선택
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	호출당 병합 패스 정렬
db.sql_tokenized.stats.sum_sort_scan_per_call	호출당 스캔 정렬
db.sql_tokenized.stats.sum_sort_range_per_call	호출당 범위 정렬

측정치	Unit
db.sql_tokenized.stats.sum_sort_rows_per_call	호출당 행 정렬
db.sql_tokenized.stats.sum_rows_affected_per_call	호출당 영향을 받는 행
db.sql_tokenized.stats.sum_rows_examined_per_call	호출당 검사된 행
db.sql_tokenized.stats.sum_rows_sent_per_call	호출당 전송된 행
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	호출당 생성된 임시 디스크 테이블
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	호출당 생성된 임시 테이블
db.sql_tokenized.stats.sum_lock_time_per_call	호출당 잠금 시간(ms)

Aurora MySQL에 대한 기본 통계

Aurora MySQL DB 클러스터에 다음 SQL 통계를 사용할 수 있습니다.

측정치	Unit
db.sql_tokenized.stats.count_star	호출
db.sql_tokenized.stats.sum_timer_wait	대기 시간(밀리초)
db.sql_tokenized.stats.sum_select_full_join	전체 조인 선택
db.sql_tokenized.stats.sum_select_range_check	범위 확인 선택
db.sql_tokenized.stats.sum_select_scan	스캔 선택
db.sql_tokenized.stats.sum_sort_merge_passes	병합 전달 정렬
db.sql_tokenized.stats.sum_sort_scan	스캔 정렬

측정치	Unit
db.sql_tokenized.stats.sum_sort_range	정렬 범위
db.sql_tokenized.stats.sum_sort_rows	행 정렬
db.sql_tokenized.stats.sum_rows_affected	영향을 받는 행
db.sql_tokenized.stats.sum_rows_examined	검사된 행
db.sql_tokenized.stats.sum_rows_sent	전송된 행
db.sql_tokenized.stats.sum_created_tmp_disk_tables	만들어진 임시 디스크 테이블
db.sql_tokenized.stats.sum_created_tmp_tables	만들어진 임시 테이블
db.sql_tokenized.stats.sum_lock_time	잠금 시간(밀리초)

Aurora PostgreSQL에 대한 SQL 통계

성능 개선 도우미가 SQL 호출과 쿼리가 실행되는 초당 통계가 SQL 통계를 수집합니다. 모든 Aurora 엔진은 다이제스트 수준에서만 통계를 수집합니다.

다음에서 Aurora PostgreSQL 다이제스트 수준 통계에 대한 정보를 찾을 수 있습니다.

주제

Aurora PostgreSQL에 대한 다이제스트 통계

SQL 다이제스트 통계를 보려면 `pg_stat_statements` 라이브러리를 로드해야 합니다. 이 라이브러리는 PostgreSQL 10과 호환되는 Aurora PostgreSQL DB 클러스터에 대해 기본적으로 로드됩니다. PostgreSQL 9.6과 호환되는 Aurora PostgreSQL DB 클러스터의 경우 이 라이브러리를 수동으로 활성화합니다. 이 라이브러리를 수동으로 활성화하려면 DB 인스턴스와 연결된 DB 파라미터 그룹의

pg_stat_statements에 shared_preload_libraries를 추가하세요. 그런 다음 DB 인스턴스를 재부팅합니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

> **Note**
>
> Performance Insights는 pg_stat_activity에서 잘리지 않은 쿼리에 대한 통계만 수집할 수 있습니다. 기본적으로 PostgreSQL 데이터베이스는 1,024바이트보다 긴 쿼리를 자릅니다. 쿼리 크기를 늘리려면 DB 인스턴스와 연결된 DB 파라미터 그룹에서 track_activity_query_size 파라미터를 변경합니다. 이 파라미터를 변경하면 DB 인스턴스를 재부팅해야 합니다.

Aurora PostgreSQL에 대한 초당 다이제스트 통계

다음 SQL 다이제스트 통계는 Aurora PostgreSQL DB 인스턴스에 제공됩니다.

측정치	Unit
db.sql_tokenized.stats.calls_per_sec	초당 호출 수
db.sql_tokenized.stats.rows_per_sec	초당 행
db.sql_tokenized.stats.total_time_per_sec	초당 평균 활성 실행(AAE)
db.sql_tokenized.stats.shared_blks_hit_per_sec	초당 블록 히트 수
db.sql_tokenized.stats.shared_blks_read_per_sec	초당 블록 읽기 수
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	초당 더티 블록 수
db.sql_tokenized.stats.shared_blks_written_per_sec	초당 블록 쓰기 수
db.sql_tokenized.stats.local_blks_hit_per_sec	초당 로컬 블록 히트 수
db.sql_tokenized.stats.local_blks_read_per_sec	초당 로컬 블록 읽기 수

측정치	Unit
db.sql_tokenized.stats.local_blks_dirtied_per_sec	초당 로컬 블록 더티 수
db.sql_tokenized.stats.local_blks_written_per_sec	초당 로컬 블록 쓰기 수
db.sql_tokenized.stats.temp_blks_written_per_sec	초당 임시 쓰기 수
db.sql_tokenized.stats.temp_blks_read_per_sec	초당 임시 읽기 수
db.sql_tokenized.stats.blk_read_time_per_sec	초당 평균 동시 읽기 수
db.sql_tokenized.stats.blk_write_time_per_sec	초당 평균 동시 쓰기 수

Aurora PostgreSQL에 대한 호출당 다이제스트 통계

다음 지표에서는 SQL 문의 호출당 통계를 제공합니다.

측정치	Unit
db.sql_tokenized.stats.rows_per_call	호출당 행 수
db.sql_tokenized.stats.avg_latency_per_call	호출당 평균 지연 시간(단위: ms)
db.sql_tokenized.stats.shared_blks_hit_per_call	호출당 블록 히트 수
db.sql_tokenized.stats.shared_blks_read_per_call	호출당 블록 읽기 수
db.sql_tokenized.stats.shared_blks_written_per_call	호출당 블록 쓰기 수
db.sql_tokenized.stats.shared_blks_dirtied_per_call	통화 당 더티 블록 수
db.sql_tokenized.stats.local_blks_hit_per_call	호출당 로컬 블록 히트 수

측정치	Unit
db.sql_tokenized.stats.local_blks_read_per_call	호출당 로컬 블록 읽기 수
db.sql_tokenized.stats.local_blks_dirtied_per_call	호출당 로컬 블록 더티 수
db.sql_tokenized.stats.local_blks_written_per_call	호출당 로컬 블록 쓰기 수
db.sql_tokenized.stats.temp_blks_written_per_call	호출당 임시 블록 쓰기 수
db.sql_tokenized.stats.temp_blks_read_per_call	호출당 임시 블록 읽기 수
db.sql_tokenized.stats.blk_read_time_per_call	호출당 읽기 시간(단위: ms)
db.sql_tokenized.stats.blk_write_time_per_call	호출당 쓰기 시간(단위: ms)

Aurora PostgreSQL에 대한 기본 통계

다음 SQL 통계는 Aurora PostgreSQL DB 인스턴스에서 사용 가능합니다.

지표	Unit
db.sql_tokenized.stats.calls	호출
db.sql_tokenized.stats.rows	행
db.sql_tokenized.stats.total_time	총 시간(밀리초)
db.sql_tokenized.stats.shared_blks_hit	블록 히트
db.sql_tokenized.stats.shared_blks_read	블록 읽기
db.sql_tokenized.stats.shared_blks_dirtied	더티 블록
db.sql_tokenized.stats.shared_blks_written	블록 쓰기
db.sql_tokenized.stats.local_blks_hit	로컬 블록 히트

지표	Unit
db.sql_tokenized.stats.local_blks_read	로컬 블록 읽기
db.sql_tokenized.stats.local_blks_dirtied	로컬 블록 더티
db.sql_tokenized.stats.local_blks_written	로컬 블록 쓰기
db.sql_tokenized.stats.temp_blks_written	임시 쓰기
db.sql_tokenized.stats.temp_blks_read	임시 읽기
db.sql_tokenized.stats.blk_read_time	평균 동시 읽기(밀리초)
db.sql_tokenized.stats.blk_write_time	평균 동시 쓰기(밀리초)

이러한 지표에 대한 자세한 내용은 PostgreSQL 설명서의 pg_stat_statements를 참조하세요.

향상된 모니터링의 OS 지표

Amazon Aurora는 DB 클러스터가 실행되는 운영 체제(OS)에 대한 측정치를 실시간으로 제공합니다. Aurora는 지표를 향상된 모니터링에서 Amazon CloudWatch Logs 계정으로 전달합니다. 다음 표에는 Amazon CloudWatch Logs에서 사용 가능한 OS 측정치가 나와 있습니다.

주제

- Aurora의 지표

Aurora의 지표

그룹	측정치	콘솔 이름	설명
`General`	`engine`	해당 사항 없음	DB 인스턴스에 대한 데이터베이스 엔진
	`instanceID`	해당 사항 없음	DB 인스턴스 식별자

그룹	측정치	콘솔 이름	설명
	instanceResourceID	해당 사항 없음	AWS 리전에 고유한 DB 인스턴스의 변경 불가능한 식별자로, 로그 스트림 식별자로도 사용됩니다.
	numVCPUs	해당 사항 없음	DB 인스턴스의 가상 CPU 수
	timestamp	해당 사항 없음	측정치를 가져온 시간
	uptime	해당 사항 없음	DB 인스턴스가 활성 상태로 유지된 시간
	version	해당 사항 없음	OS 측정치 스트림 JSON 형식의 버전
cpuUtilization	guest	CPU 게스트	게스트 프로그램에서 사용 중인 CPU의 비율
	idle	CPU 유휴	유휴 상태인 CPU의 비율
	irq	CPU IRQ	소프트웨어 인터럽트에서 사용 중인 CPU의 비율
	nice	CPU Nice	가장 낮은 우선순위로 실행 중인 프로그램에서 사용 중인 CPU의 비율
	steal	CPU Steal	다른 가상 머신에서 사용 중인 CPU의 비율
	system	CPU 시스템	커널에서 사용 중인 CPU의 비율
	total	CPU 합계	사용 중인 CPU의 총 비율입니다. 이 값에는 nice 값이 포함됩니다.
	user	CPU 사용자	사용자 프로그램에서 사용 중인 CPU의 비율

그룹	측정치	콘솔 이름	설명
	wait	CPU 대기	I/O 액세스를 대기 중인 동안 사용되지 않은 CPU의 비율
diskIO	avgQueueLen	평균 대기열 크기	I/O 디바이스의 대기열에서 대기 중인 요청 수입니다.
	avgReqSz	평균 요청 크기	평균 요청 크기(KB)
	await	디스크 I/O 대기	대기열 시간과 서비스 시간을 포함하여 요청에 응답하는 데 필요한 시간(밀리초)
	device	해당 사항 없음	사용 중인 디스크 디바이스의 식별자
	readIOsPS	읽기 IO/초	초당 읽기 작업 수
	readKb	읽기 합계	읽은 총 KB 수
	readKbPS	읽기 KB/초	초당 읽은 KB 수
	readLatency	읽기 지연 시간	읽기 I/O 요청을 제출하여 처리가 완료될 때까지 걸리는 시간(밀리초) 이 지표는 Amazon Aurora에만 사용할 수 없습니다.
	readThroughput	읽기 처리량	DB 클러스터에 요청할 때 사용되는 네트워크 처리량(초당 바이트) 이 지표는 Amazon Aurora에만 사용할 수 없습니다.
	rrqmPS	Rrqms	초당 대기 중인 병합 읽기 요청 수
	tps	TPS	초당 I/O 트랜잭션 수
	util	디스크 I/O 유틸리티	요청이 발급된 CPU 시간의 비율

그룹	측정치	콘솔 이름	설명
	writeIOsPS	쓰기 IO/초	초당 쓰기 작업 수
	writeKb	쓰기 합계	기록한 총 KB 수
	writeKbPS	쓰기 KB/초	초당 기록한 KB 수
	writeLatency	쓰기 지연 시간	쓰기 I/O 요청을 제출하여 처리가 완료될 때까지 걸리는 평균 시간(ms) 이 지표는 Amazon Aurora에만 사용할 수 없습니다.
	writeThroughput	쓰기 처리량	DB 클러스터의 응답에 사용되는 네트워크 처리량(초당 바이트) 이 지표는 Amazon Aurora에만 사용할 수 없습니다.
	wrqmPS	Wrqms	초당 대기 중인 병합 쓰기 요청 수
fileSys	maxFiles	최대 Inode	파일 시스템에 대해 생성될 수 있는 최대 파일 수
	mountPoint	해당 사항 없음	파일 시스템의 경로
	name	해당 사항 없음	파일 시스템의 이름
	total	총 파일 시스템	파일 시스템에 사용 가능한 총 디스크 공간(KB)
	used	사용된 파일 시스템	파일 시스템에서 파일에 사용된 디스크 공간의 양(KB)
	usedFilePercent	사용된 Inode	사용 중인 사용 가능한 파일의 비율

그룹	측정치	콘솔 이름	설명
	usedFiles	사용된 %	파일 시스템의 파일 수
	usedPercent	사용된 파일 시스템	사용 중인 파일 시스템 디스크 공간의 비율
loadAverageMinute	fifteen	평균 부하 15분	마지막 15분 동안 CPU 시간을 요청한 프로세스 수
	five	평균 로드 5분	마지막 5분 동안 CPU 시간을 요청한 프로세스 수
	one	평균 로드 1분	마지막 1분 동안 CPU 시간을 요청한 프로세스 수
memory	active	활성 메모리	할당된 메모리의 양(KB)
	buffers	버퍼링된 메모리	스토리지 디바이스에 쓰기 이전에 I/O 요청을 버퍼링하는 데 사용되는 메모리의 양(KB)
	cached	캐시된 메모리	파일 시스템 기반 I/O를 캐시하는 데 사용된 메모리의 양
	dirty	더티 메모리	수정되었지만 스토리지의 관련 데이터 블록에 기록되지 않은 RAM의 메모리 페이지 양(KB)
	free	여유 메모리	할당되지 않은 메모리의 양(KB)
	hugePagesFree	사용 가능한 방대한 페이지	사용 가능한 방대한 페이지 수입니다. 방대한 페이지는 Linux 커널의 기능입니다.
	hugePagesRsvd	예약된 방대한 페이지	커밋된 방대한 페이지의 수

그룹	측정치	콘솔 이름	설명
	hugePagesSize	방대한 페이지 크기	각 방대한 페이지 단위의 크기(KB)
	hugePagesSurp	초과된 방대한 페이지	총계 대비 사용 가능한 초과 방대한 페이지 수
	hugePagesTotal	방대한 페이지 합계	방대한 페이지의 총 수입니다.
	inactive	비활성 메모리	가장 적게 사용되는 메모리 페이지의 양(KB)
	mapped	매핑된 메모리	프로세스 주소 공간 내에 메모리 매핑되는 총 파일 시스템 콘텐츠 양(KB)
	pageTables	페이지 테이블	페이지 표에 사용된 메모리의 양(KB)
	slab	슬래브 메모리	재사용 가능한 커널 데이터 구조의 양(KB)
	total	메모리 합계	총 메모리 양(KB)
	writeback	다시 쓰기 메모리	RAM에서 지원 스토리지에 아직 기록 중인 더티 페이지의 양(KB)
network	interface	해당 사항 없음	DB 인스턴스에 대해 사용 중인 네트워크 인터페이스의 식별자
	rx	RX	초당 수신된 바이트 수
	tx	TX	초당 업로드된 바이트 수
processList	cpuUsedPc	CPU %	프로세스에서 사용된 CPU 비율

그룹	측정치	콘솔 이름	설명
	id	해당 사항 없음	프로세스의 식별자
	memoryUsedPc	MEM%	프로세스에서 사용된 메모리 비율
	name	해당 사항 없음	프로세스의 이름입니다.
	parentID	해당 사항 없음	프로세스의 상위 프로세스에 대한 프로세스 식별자
	rss	RES	프로세스에 할당된 RAM의 양(KB)
	tgid	해당 사항 없음	스레드 그룹 식별자이며, 스레드가 속한 프로세스 ID를 나타내는 숫자입니다. 이 식별자는 동일한 프로세스에서 스레드를 그룹화하는 데 사용됩니다.
	vss	VIRT	프로세스에 할당된 가상 메모리의 양(KB)
swap	total	Swap	사용 가능한 스왑 메모리 양(KB)
	in	스왑 인	디스크에서 스왑된 메모리 양(KB)
	out	스왑 아웃	디스크로 스왑된 총 메모리 양(KB)
	free	사용 가능한 스왑	사용 가능한 스왑 메모리 양(KB)
	cached	커밋된 스왑	캐시 메모리로 사용된 스왑 메모리의 양(KB)
tasks	blocked	차단되는 작업	차단되는 작업 수
	running	실행 중인 작업	실행 중인 작업 수

그룹	측정치	콘솔 이름	설명
	sleeping	절전 상태인 작업	절전 상태인 작업 수
	stopped	중단된 작업	중단된 작업 수
	total	총 작업	총 작업 수
	zombie	작업 좀비	상위 작업은 활성화되었지만 비활성 상태인 하위 작업 수

Amazon Aurora DB 클러스터에서 이벤트, 로그 및 스트림 모니터링

Amazon Aurora 데이터베이스 및 기타 AWS 솔루션을 모니터링할 때의 목표는 다음을 유지하는 것입니다.

- 신뢰성
- 가용성
- 성능
- 보안

Amazon Aurora 클러스터에서 지표 모니터링에서 지표를 사용하여 클러스터를 모니터링하는 방법을 설명합니다. 완벽한 솔루션은 데이터베이스 이벤트, 로그 파일 및 활동 스트림을 모니터링해야 합니다. AWS에서는 다음과 같은 모니터링 도구를 제공합니다.

- Amazon EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, 서비스형 소프트웨어(SaaS) 애플리케이션 및 AWS 서비스의 실시간 데이터 스트림을 제공하고 그 데이터를 AWS Lambda 등의 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 https://docs.aws.amazon.com/eventbridge/latest/userguide/Amazon EventBridge 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs는 Amazon Aurora 인스턴스, AWS CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스하는 방법을 제공합니다. Amazon CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임계값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하십시오.
- AWS CloudTrail는 AWS 계정에서 또는 이 계정을 대신하여 수행된 API 호출 및 관련 이벤트를 캡처합니다. CloudTrail은 지정된 Amazon S3 버킷에 CloudTrail 이벤트 로그 파일을 전송합니다. 어떤 사용자 및 계정이 AWS를 호출했는지 어떤 소스 IP 주소에 호출이 이루어졌는지 언제 호출이 발생했는지 확인할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.
- 데이터베이스 활동 스트림은 DB 클러스터에서 거의 실시간에 가까운 활동 스트림을 제공하는 Amazon Aurora 기능입니다. Amazon Aurora는 활동을 Amazon Kinesis Data Streams에 푸시합니다. Kinesis 스트림이 자동으로 생성됩니다. Kinesis에서 Amazon Data Firehose 및 AWS Lambda와 같은 AWS 서비스를 구성하여 스트림을 사용하고 데이터를 저장할 수 있습니다.

주제

Amazon RDS 콘솔에서 로그, 이벤트 및 스트림 보기

Amazon RDS는 RDS 콘솔에서 로그, 이벤트 및 데이터베이스 활동 스트림에 대한 정보를 표시하기 위해 AWS 서비스와 통합합니다.

Aurora DB 클러스터의 로그 및 이벤트(Logs & events) 탭에는 다음 정보가 나와 있습니다.

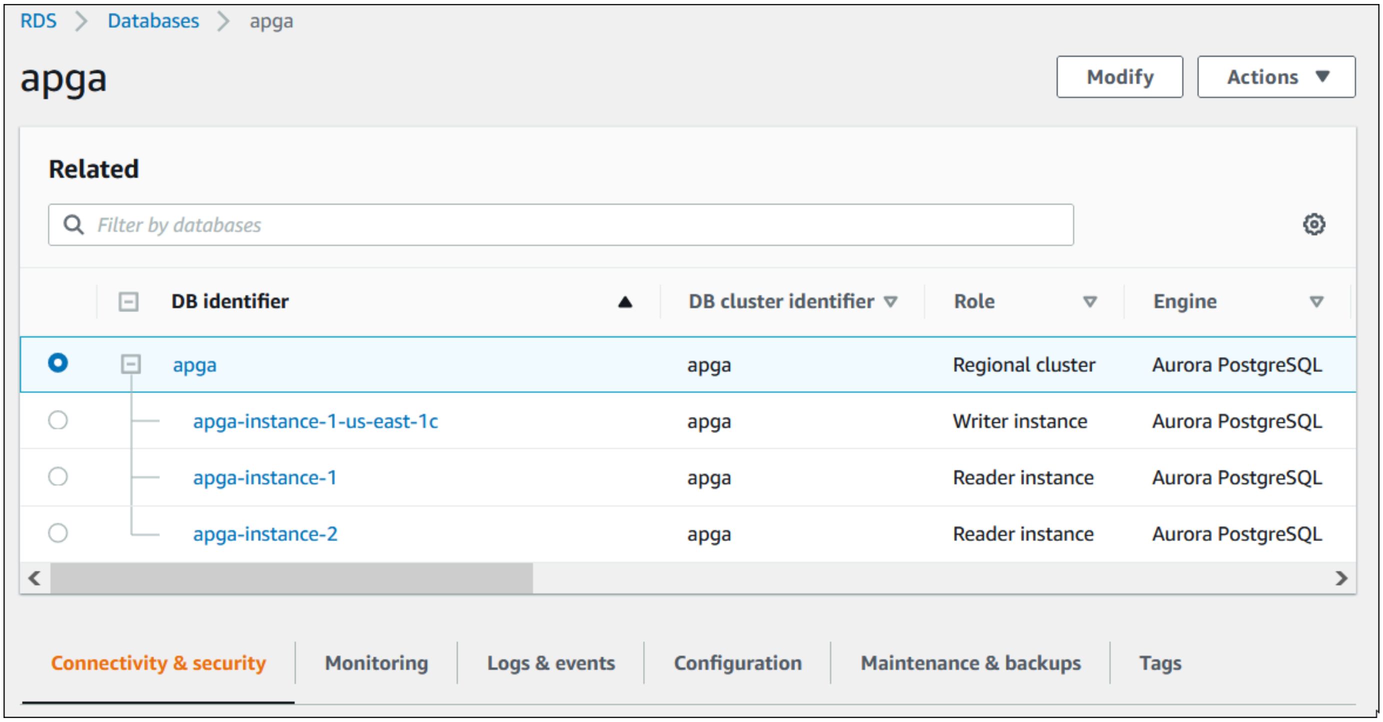
- Auto Scaling 정책 및 활동: Aurora Auto Scaling 기능 관련 정책 및 활동을 표시합니다. 이 정보는 클러스터 수준에서 로그 및 이벤트(Logs & events) 탭에만 표시됩니다.
- Amazon CloudWatch 경보: Aurora 클러스터에서 DB 인스턴스에 대해 구성한 모든 지표 경보를 표시합니다. 경보를 구성하지 않은 경우 RDS 콘솔에서 생성할 수 있습니다.
- 최근 이벤트(Recent events): Aurora DB 인스턴스 또는 클러스터에 대한 이벤트 요약(환경 변경 사항)을 보여줍니다. 자세한 내용은 Amazon RDS 이벤트 보기을 참조하세요.
- 로그(Logs): Aurora 클러스터에서 DB 인스턴스에서 생성된 데이터베이스 로그 파일을 표시합니다. 자세한 내용은 Amazon Aurora 로그 파일 모니터링을 참조하세요.

구성(Configuration) 탭은 데이터베이스 활동 스트림에 대한 정보를 표시합니다.

RDS 콘솔에서 Aurora DB 클러스터에 대한 로그, 이벤트 및 스트림 보기

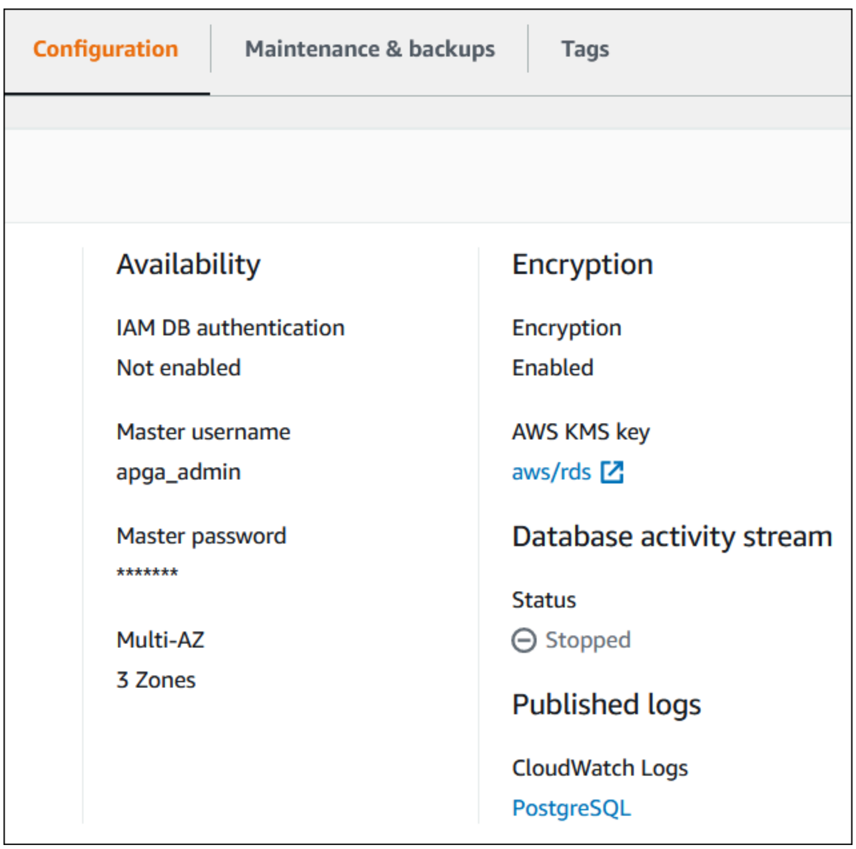
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 모니터링 하려는 Aurora DB 클러스터의 이름을 선택합니다.

   데이터베이스 페이지가 표시됩니다. 다음 예는 apga라는 Amazon Aurora PostgreSQL DB 클러스터의 을 보여줍니다.



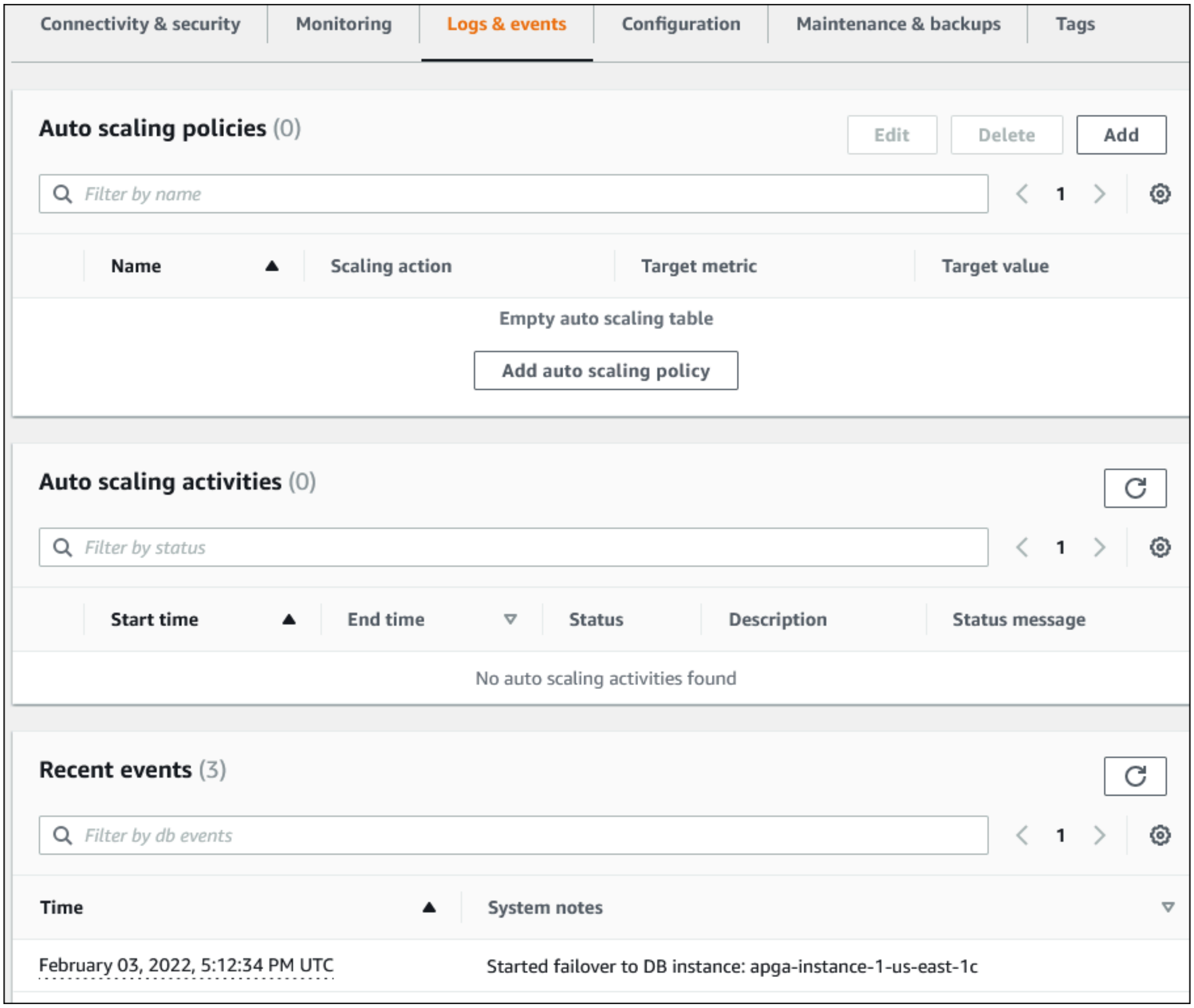
4. 아래로 스크롤하여 구성(Configuration)을 선택합니다.

   다음 예는 클러스터에 대한 데이터베이스 활동 스트림 상태를 보여줍니다.



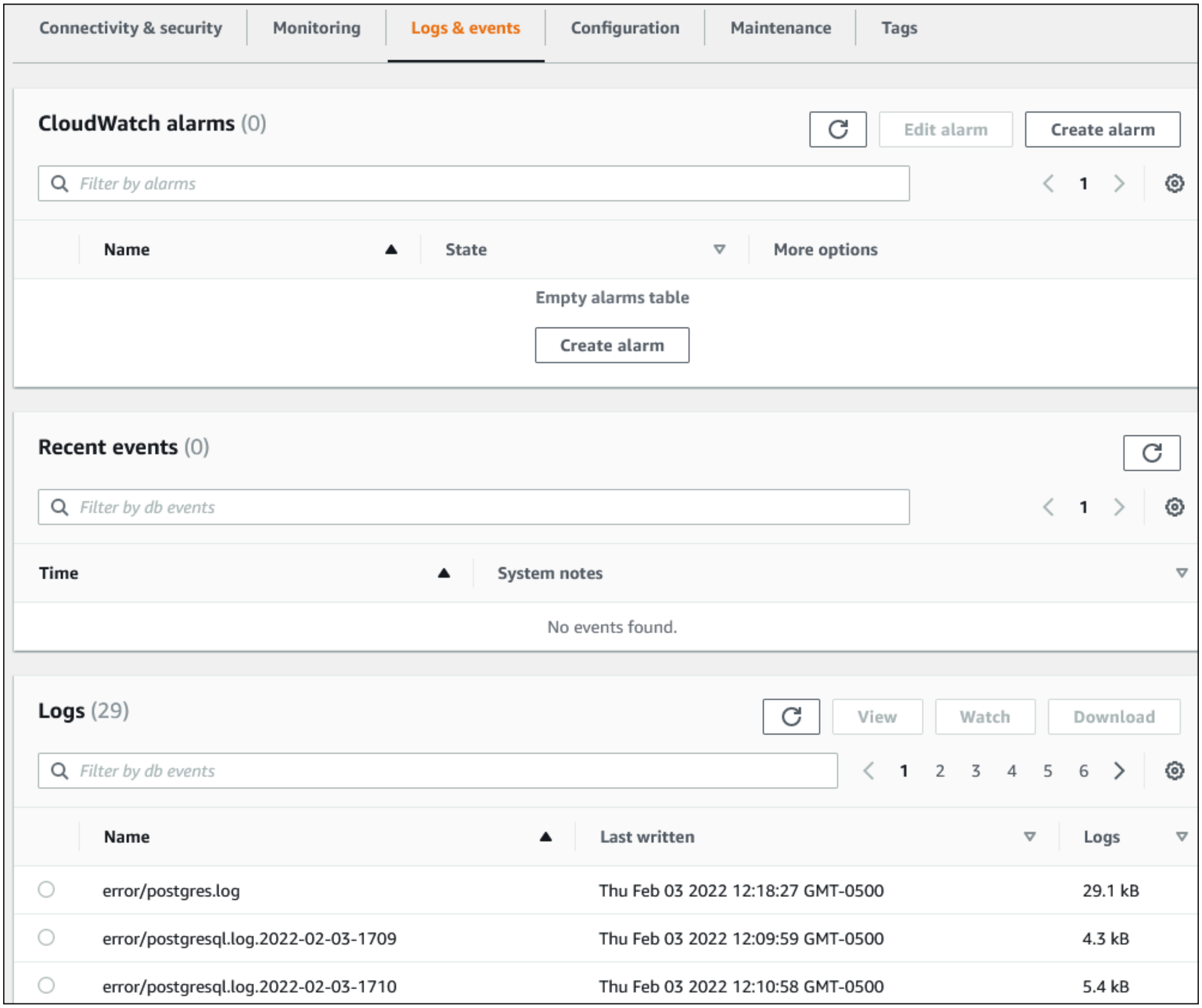
5. 로그 및 이벤트를 선택합니다.

   로그 및 이벤트 섹션이 표시됩니다.



6. Aurora 클러스터에서 DB 인스턴스를 선택한 다음 인스턴스에 대한 로그 및 이벤트(Logs & events)를 선택합니다.

   다음 예제에서는 DB 인스턴스 페이지와 DB 클러스터 페이지 간에 콘텐츠가 다르다는 것을 보여줍니다. DB 인스턴스 페이지에는 로그와 경보가 표시됩니다.


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

Amazon Aurora 이벤트 모니터링

이벤트는 환경의 변화를 나타냅니다. AWS 환경, 즉 SaaS 파트너 서비스나 애플리케이션 또는 사용자 지정 애플리케이션이나 서비스일 수 있습니다. Aurora 이벤트에 대한 설명을 보려면 Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지 섹션을 참조하세요.

주제

Aurora에 대한 이벤트 개요

RDS 이벤트는 Aurora 환경의 변경 사항을 나타내는 이벤트입니다. 예를 들어 Amazon Aurora는 DB 클러스터가 패치될 때 이벤트를 생성합니다. Amazon Aurora는 거의 실시간으로 EventBridge에 이벤트를 전송합니다.

> **Note**
>
> Amazon RDS는 최선의 작업을 기반으로 이벤트를 방출합니다. 알림 이벤트의 순서나 존재 여부에 따라 달라지는 프로그램은 순서가 잘못되거나 누락될 수 있으므로 작성하지 않는 것이 좋습니다.

Amazon RDS는 다음 리소스와 관련된 이벤트를 기록합니다.

- DB 클러스터

  클러스터 이벤트 목록은 DB 클러스터 이벤트 섹션을 참조하세요.
- DB 인스턴스

  DB 인스턴스 이벤트 목록은 DB 인스턴스 이벤트 섹션을 참조하세요.
- DB 파라미터 그룹

  DB 파라미터 그룹 이벤트 목록은 DB 파라미터 그룹 이벤트 섹션을 참조하세요.

- DB 보안 그룹

  DB 보안 그룹 이벤트 목록은 DB 보안 그룹 이벤트 섹션을 참조하세요.
- DB 클러스터 스냅샷

  DB 클러스터 스냅샷 이벤트 목록은 DB 클러스터 스냅샷 이벤트 섹션을 참조하세요.
- RDS 프록시 이벤트

  RDS 프록시 이벤트 목록은 RDS 프록시 이벤트 섹션을 참조하세요.
- 블루/그린 배포 이벤트

  블루/그린 배포 이벤트 목록은 블루/그린 배포 이벤트 섹션을 참조하세요.

이 정보에는 다음 사항이 포함됩니다.

- 이벤트의 날짜 및 시간
- 이벤트의 소스 이름 및 소스 유형
- 이벤트와 연결된 메시지
- 이벤트 알림에는 메시지가 전송된 시점의 태그가 포함되며 이벤트가 발생한 시점의 태그는 반영되지 않을 수 있습니다.

Amazon RDS 이벤트 보기

Amazon Aurora 리소스에 대해 다음 이벤트 정보를 검색할 수 있습니다.

- 리소스 이름
- 리소스 유형
- 이벤트 시간
- 이벤트 메시지 요약

AWS Management Console의 다음 부분에서 이벤트에 액세스할 수 있습니다.

- 이벤트 탭은 지난 24시간 동안의 이벤트를 표시합니다.
- 데이터베이스 탭의 로그 및 이벤트 섹션에 있는 최근 이벤트 테이블에는 최대 지난 2주 동안의 이벤트가 표시될 수 있습니다.

describe-events AWS CLI 명령 또는 DescribeEvents RDS API 작업을 사용하여 이벤트를 검색할 수도 있습니다. AWS CLI 또는 RDS API를 사용하여 이벤트를 볼 경우 최대 지난 14일 동안 발생한 이벤트를 검색할 수 있습니다.

> **Note**
>
> 더 오랜 기간 동안 이벤트를 저장해야 하는 경우 EventBridge에 Amazon RDS 이벤트를 보낼 수 있습니다. 자세한 내용은 Amazon Aurora 이벤트에서 트리거되는 규칙 생성 섹션을 참조하세요.

Amazon Aurora 이벤트에 대한 설명을 보려면 Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지 섹션을 참조하세요.

요청 파라미터를 포함하여 AWS CloudTrail를 사용하여 이벤트에 대한 세부 정보에 액세스하려면 CloudTrail 이벤트 단원을 참조하십시오.

콘솔

지난 24시간 동안 발생한 모든 Amazon RDS 이벤트를 보는 방법

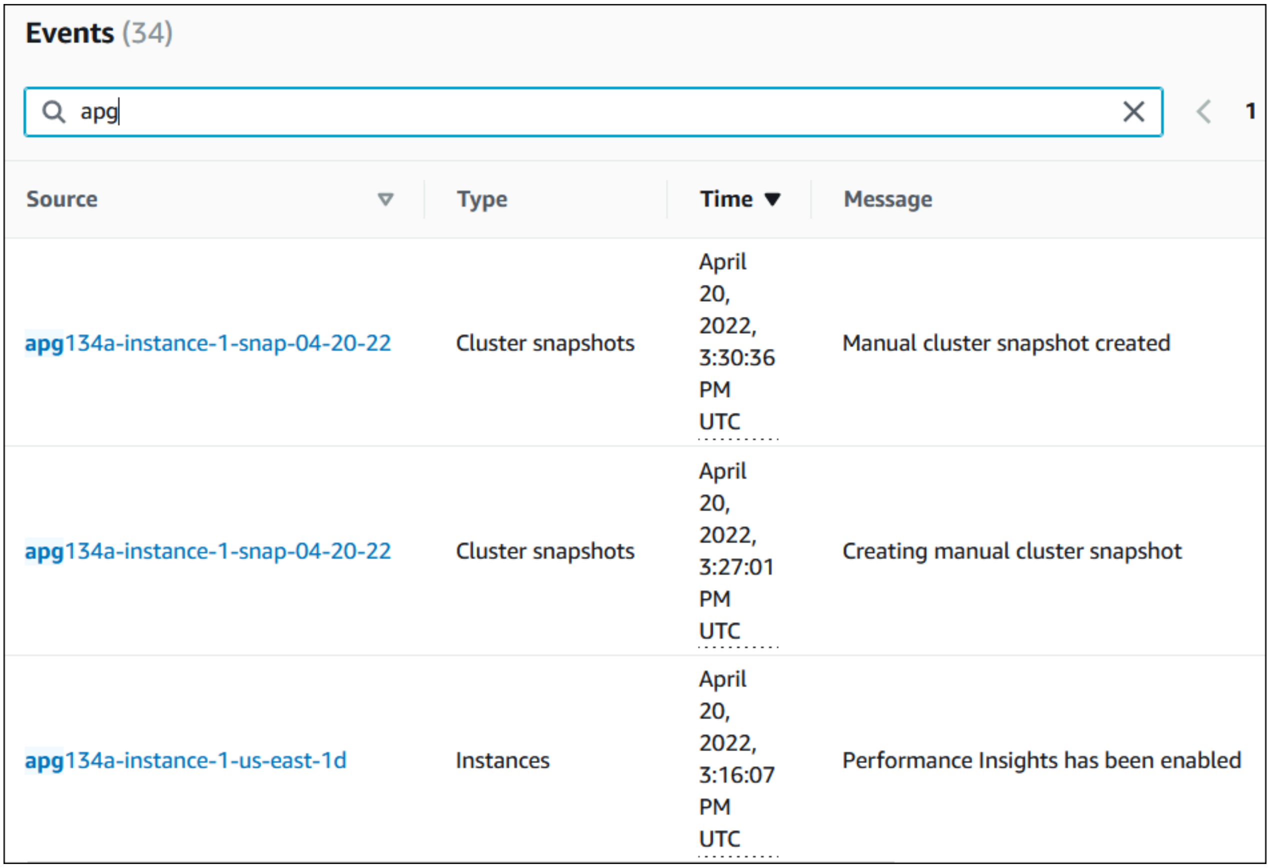
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 [Events]를 선택합니다.

   사용 가능한 이벤트가 목록에 표시됩니다.

3. (선택 사항) 검색어를 입력하여 결과를 필터링합니다.

   다음 예에서는 **apg** 문자로 필터링된 이벤트 목록을 보여줍니다.



AWS CLI

지난 한 시간 동안 생성된 모든 이벤트를 보려면 파라미터 없이 describe-events를 호출합니다.

```
aws rds describe-events
```

다음 샘플 출력은 DB 클러스터 인스턴스가 복구를 시작했음을 보여줍니다.

```
{
    "Events": [
        {
            "EventCategories": [
                "recovery"
```

```
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

지난 10,080분(7일) 동안의 모든 Amazon RDS 이벤트를 보려면 describe-events AWS CLI 명령을 호출하고 `--duration` 파라미터를 `10080`으로 설정합니다.

```
aws rds describe-events --duration 10080
```

다음 예에서는 DB 인스턴스 *test-instance*에 대해 지정된 시간 범위의 이벤트를 보여줍니다.

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

다음 샘플 출력은 백업 상태를 보여줍니다.

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
```

```
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
    ]
}
```

API

DescribeEvents RDS API 작업을 호출하고 `Duration` 파라미터를 20160으로 설정하여 지난 14일 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 볼 수 있습니다.

Amazon RDS 이벤트 알림 작업

Amazon RDS는 Amazon RDS 이벤트 발생 시 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알림 서비스를 제공합니다. 이 서비스는 AWS 리전에 따라 Amazon SNS가 지원하는 알림 메시지 형식에 따라 이메일, 문자 또는 HTTP 엔드포인트 호출 등이 될 수 있습니다.

주제

Amazon RDS 이벤트 알림 개요

Amazon RDS는 구독 가능한 카테고리로 이벤트를 그룹화합니다. 따라서 해당 카테고리의 이벤트가 발생했을 때 이에 대한 알림 메시지를 받을 수 있습니다.

주제

이벤트 구독에 적합한 RDS 리소스

Amazon Aurora의 경우 이벤트는 DB 클러스터와 DB 인스턴스 수준 모두에서 발생합니다. 다음 리소스에 대한 이벤트 범주에 구독할 수 있습니다.

- DB 인스턴스
- DB 클러스터
- DB 클러스터 스냅샷
- DB 파라미터 그룹
- DB 보안 그룹
- RDS 프록시
- 커스텀 엔진 버전

예를 들어 임의의 DB 인스턴스에 대한 백업 카테고리를 구독할 경우 백업 관련 이벤트가 발생하여 DB 인스턴스에 영향을 끼칠 때마다 알림 메시지가 수신됩니다. 혹은 DB 인스턴스의 구성 변경 카테고리를 구독하면 DB 인스턴스가 변경될 때마다 메시지가 수신됩니다. 또한 이벤트 알림 메시지 구독이 변경되어도 알림 메시지가 수신됩니다.

여러 가지 다른 구독을 만들 수 있습니다. 예를 들어, 전체 DB 인스턴스에 대한 모든 이벤트 알림을 수신하는 하나의 구독과 DB 인스턴스의 하위 집합에 대한 중요 이벤트만 포함하는 다른 구독을 만들 수 있습니다. 두 번째 구독의 경우 필터에 하나 이상의 DB 인스턴스를 지정합니다.

Amazon RDS 이벤트 알림을 구독하는 기본 프로세스

Amazon RDS 이벤트 알림 서비스를 구독하는 프로세스는 다음과 같습니다.

1. Amazon RDS 콘솔, AWS CLI 또는 API를 사용하여 Amazon RDS 이벤트 알림 서비스 구독을 생성합니다.

   Amazon RDS에서는 Amazon SNS 주제의 ARN을 사용하여 각 구독을 식별합니다. Amazon RDS 콘솔은 구독 생성 시 ARN을 생성합니다. Amazon SNS 콘솔인 AWS CLI 또는 Amazon SNS API를 사용하여 ARN을 생성합니다.
2. Amazon RDS가 구독 생성 시 제출한 이메일 주소로 승인 이메일 또는 SMS 메시지를 전송합니다.
3. 전송된 알림의 링크를 선택하여 구독 여부를 확인합니다.
4. Amazon RDS 콘솔은 구독 상태로 내 이벤트 구독 섹션을 업데이트합니다.
5. Amazon RDS는 구독을 생성할 때 제공한 주소로 알림을 보내기 시작합니다.

Amazon SNS 사용 시 자격 증명 및 액세스 관리에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 가이드의 [Amazon SNS의 Identity and Access Management](https://docs.aws.amazon.com/sns/latest/dg/sns-authentication-and-access-control.html)를 참조하세요.

AWS Lambda를 사용하여 DB 인스턴스의 이벤트 알림을 처리할 수 있습니다. 자세한 내용은 AWS Lambda 개발자 안내서의 Amazon RDS에서 AWS Lambda 사용을 참조하세요.

RDS 이벤트 알림 전송

Amazon RDS는 구독 생성 시 입력한 주소로 알림을 보냅니다. 알림에는 메시지에 대한 구조화된 메타데이터를 제공하는 메시지 속성이 포함될 수 있습니다. 메시지 속성에 대한 자세한 내용은 Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지 단원을 참조하세요.

이벤트 알림을 전달하는 데 최대 5분 정도 걸릴 수 있습니다.

> **Important**
>
> Amazon RDS는 이벤트 스트림에서 전송된 이벤트 순서를 보장하지 않습니다. 이벤트 순서는 변경될 수 있습니다.

Amazon SNS이 구독된 HTTP 또는 HTTPS 엔드포인트에 알림을 전송할 때 엔드포인트에 전송된 POST 메시지에는 JSON 문서를 포함하는 메시지 본문이 있습니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 메시지 및 JSON 형식을 참조하십시오.

문자 메시지로 알리도록 SNS를 구성할 수 있습니다. 자세한 내용은 Amazon Simple Notification Service 개발자 가이드의 모바일 문자 메시지(SMS)를 참조하세요.

구독을 삭제하지 않고 알림을 끄려면 Amazon RDS 콘솔에서 활성에 대해 아니요를 선택합니다. 또는 AWS CLI 또는 Amazon RDS API를 사용하여 `Enabled` 파라미터를 `false`(으)로 설정할 수 있습니다.

Amazon RDS 이벤트 알림 결제

Amazon RDS 이벤트 알림에 대한 결제는 Amazon SNS를 통해 이루어집니다. Amazon SNS 요금은 이벤트 알림 사용 시 적용됩니다. Amazon SNS 결제에 대한 자세한 내용은 Amazon Simple Notification Service 요금을 참조하세요.

Amazon EventBridge를 사용한 Aurora 이벤트의 예

다음 예에서는 JSON 형식으로 다양한 Aurora 이벤트 유형을 보여줍니다. JSON 형식으로 이벤트를 캡처하고 보는 방법을 보여 주는 자습서는 자습서: Amazon EventBridge를 사용하여 DB 인스턴스의 상태 변경 로깅 단원을 참조하세요.

주제

- DB 클러스터 이벤트의 예
- DB 파라미터 그룹 이벤트의 예
- DB 클러스터 스냅샷 이벤트의 예

DB 클러스터 이벤트의 예

다음은 JSON 형식의 DB 클러스터 이벤트에 대한 예제입니다. 이 이벤트는 이름이 `my-db-cluster`인 클러스터에 패치가 적용되었음을 보여줍니다. 이벤트 ID는 `RDS-EVENT-0173`입니다.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

DB 파라미터 그룹 이벤트의 예

다음은 JSON 형식의 DB 파라미터 그룹 이벤트에 대한 예제입니다. 이 이벤트는 `time_zone` 파라미터가 파라미터 그룹 `my-db-param-group`에서 업데이트되었음을 보여줍니다. 이벤트 ID는 RDS-EVENT-0037입니다.

```
{
  "version": "0",
```

```
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

DB 클러스터 스냅샷 이벤트의 예

다음은 JSON 형식의 DB 클러스터 스냅샷 이벤트에 대한 예제입니다. 이 이벤트는 `my-db-cluster-snapshot`이라는 스냅샷의 생성을 보여줍니다. 이벤트 ID는 RDS-EVENT-0074입니다.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
```

```
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

Amazon SNS 주제에 알림을 게시할 권한 부여

Amazon RDS 권한을 부여하여 알림을 Amazon Simple Notification Service(Amazon SNS) 주제에 게시하려면 대상 주제에 AWS Identity and Access Management(IAM) 정책을 연결하면 됩니다. 권한에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 [Amazon Simple Notification Service 액세스 제어 예제 사례]를 참조하세요.

기본적으로 Amazon SNS 주제에는 동일한 계정 내의 모든 Amazon RDS 리소스가 알림을 게시하도록 허용하는 정책이 있습니다. 교차 계정 알림을 허용하거나 특정 리소스에 대한 액세스를 제한하는 사용자 지정 정책을 연결할 수 있습니다.

다음은 대상 Amazon SNS 주제에 연결하는 IAM 정책의 예입니다. 지정된 접두사와 일치하는 이름을 가진 DB 인스턴스로 주제를 제한합니다. 이 정책을 사용하려면 다음 값을 지정합니다.

- `Resource` – Amazon SNS 주제에 대한 Amazon 리소스 이름(ARN)
- `SourceARN` - RDS 리소스 ARN
- `SourceAccount` – 사용자의 AWS 계정 ID

리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조에서 [Amazon RDS에서 정의한 리소스]를 참조하세요.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
```

```
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Amazon RDS 이벤트 알림 구독

가장 간단한 구독 생성 방법은 RDS 콘솔입니다. CLI 또는 API를 사용하여 이벤트 알림 구독을 생성하려면 먼저 Amazon Simple Notification Service 주제를 만든 후 Amazon SNS 콘솔이나 Amazon SNS API를 통해 해당 주제를 구독해야 합니다. 또한 CLI 명령이나 API 작업을 제출할 때도 사용되기 때문에 해당 주제의 Amazon 리소스 이름(ARN)을 잊어서는 안 됩니다. SNS 주제를 새로 만들어 구독하는 방법에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 시작하기 단원을 참조하십시오.

알림 메시지를 받고 싶은 소스 유형과 이벤트를 트리거링하는 Amazon RDS 소스를 지정할 수 있습니다.

소스 유형

소스 유형입니다. 예를 들어 소스 유형은 인스턴스일 수 있습니다. 소스 유형을 선택해야 합니다.

포함할 *###*

이벤트를 생성하는 Amazon RDS 리소스입니다. 예를 들어 특정 인스턴스 선택을 선택한 다음 myDBInstance1을 선택할 수 있습니다.

다음 테이블에서는 포함할 *###*를 지정하거나 지정하지 않을 때의 결과를 설명합니다

포함할 리소스	설명	예
지정됨	RDS는 지정된 리소스에 대한 모든 이벤트에 대해서만 알림을 보냅니다.	소스 유형이 인스턴스이고 리소스가 myDBInstance1인 경우 RDS는 myDBInstance1에 대한 모든 이벤트에 대해서만 알립니다.
지정되지 않음	RDS는 모든 Amazon RDS 리소스에 대해 지정된 소스 유형에 대한 이벤트를 알려줍니다.	소스 유형이 인스턴스인 경우 RDS는 계정의 모든 인스턴스 관련 이벤트에 대해 알려줍니다.

Amazon SNS 주제 구독자는 기본적으로 주제에 게시된 모든 메시지를 수신합니다. 메시지의 하위 집합만 수신하려면 구독자는 주제 구독에 필터 정책을 할당해야 합니다. SNS 메시지 필터링에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 메시지 필터링을 참조하세요.

콘솔

RDS 이벤트 알림 구독 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 이벤트 구독을 선택합니다.
3. 이벤트 구독 창에서 이벤트 구독 생성을 선택합니다.
4. 다음과 같이 구독 세부 정보를 입력합니다.

    a. 이름에서 이벤트 알림 구독 이름을 입력합니다.

    b. 알림 보내기 대상에게 다음 중 하나를 실시합니다.

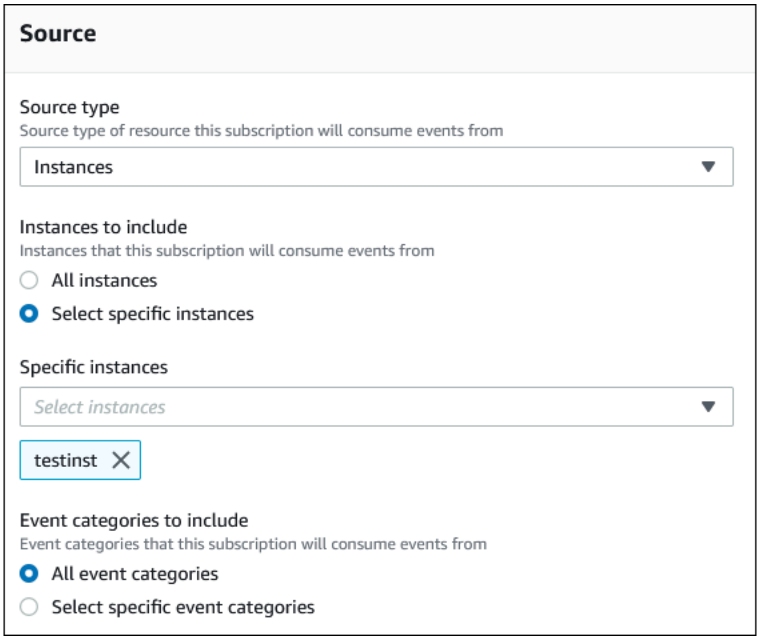
        - 새 이메일 주제를 선택합니다. 이메일 주제의 이름과 수신자 목록을 입력합니다. 기본 계정 연락처와 동일한 이메일 주소로 이벤트 구독을 구성하는 것이 좋습니다. 권장 사항, 서비스 이벤트 및 개인 건강 메시지는 다양한 채널을 사용하여 전송됩니다. 동일한 이메일 주소를 구독하면 모든 메시지가 한 위치에 통합됩니다.
        - Amazon 리소스 이름(ARN)을 선택합니다. 그런 다음 Amazon SNS 주제에 대해 기존 Amazon SNS ARN을 선택합니다.

          서버 측 암호화(SSE)에 대해 활성화된 주제를 사용하려면 AWS KMS key에 액세스하는 데 필요한 권한을 Amazon RDS에 부여하면 됩니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 AWS 서비스 및 암호화된 주제의 이벤트 소스 간 호환성 활성화를 참조하세요.

    c. [Source type]에서 원본 형식을 선택합니다. 예를 들어 클러스터 또는 클러스터 스냅샷을 선택합니다.

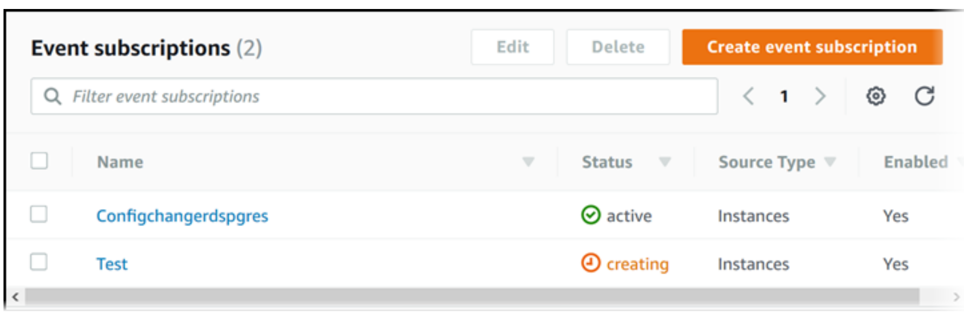
    d. 이벤트 알림을 받을 이벤트 카테고리 및 리소스를 선택합니다.

       다음 예에서는 `testinst`라는 DB 인스턴스에 대한 이벤트 알림을 구성합니다.



e. 생성을 선택합니다.

Amazon RDS 콘솔에 현재 구독 생성 중으로 나옵니다.



AWS CLI

RDS 이벤트 알림을 구독하려면 AWS CLI [`create-event-subscription`] 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`
- `--sns-topic-arn`

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-event-subscription \
```

```
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

Windows의 경우:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

Amazon RDS 이벤트 알림을 구독하려면 Amazon RDS API 함수 `CreateEventSubscription`을 호출합니다. 다음 필수 파라미터를 포함합니다.

- `SubscriptionName`
- `SnsTopicArn`

Amazon RDS 이벤트 알림 태그 및 속성

Amazon RDS에서 Amazon Simple Notification Service(SNS) 또는 Amazon EventBridge로 이벤트 알림을 전송할 경우, 알림에 메시지 속성과 이벤트 태그가 포함됩니다. RDS는 이벤트 태그가 메시지 본문에 있는 동안 메시지 속성을 메시지와 함께 별도로 전송합니다. 메시지 속성과 Amazon RDS 태그를 사용하여 메타데이터를 리소스에 추가할 수 있습니다. 이러한 태그를 DB 인스턴스, Aurora 클러스터 등에 대한 주석과 함께 수정할 수 있습니다. Amazon RDS 리소스 태그 지정에 대한 자세한 내용은 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 단원을 참조하세요.

기본적으로 Amazon SNS 및 Amazon EventBridge는 전송되는 모든 메시지를 수신합니다. SNS 및 EventBridge는 메시지를 필터링할 수 있으며 이메일, 문자 메시지 또는 HTTP 엔드포인트로 수신되는 통화 같은 기본 커뮤니케이션 모드로 알림을 전송할 수 있습니다.

> **Note**
> 이메일이나 문자 메시지로 전송된 알림에는 이벤트 태그가 없습니다.

다음 표에는 주제 구독자에게 전송되는 RDS 이벤트의 메시지 속성이 나와 있습니다.

Amazon RDS 이벤트 속성	설명
EventID	RDS 이벤트 메시지의 식별자는 예를 들어 RDS-EVENT -0006입니다.
리소스	이벤트를 발생시키는 리소스의 ARN 식별자(예: `arn:aws:rds:ap-southeast-2:123456789012:db:database-1`)입니다.

RDS 태그는 서비스 이벤트의 영향을 받은 리소스에 대한 데이터를 제공합니다. RDS는 알림이 SNS 또는 EventBridge로 전송될 때 메시지 본문에 태그의 현재 상태를 추가합니다.

SNS의 필터링 메시지 속성에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 메시지 필터링을 참조하세요.

EventBridge의 이벤트 태그 필터링에 대한 자세한 내용은 Amazon EventBridge 사용 설명서에서 Amazon EventBridge 이벤트 패턴에 사용할 콘텐츠 비교 연산자를 참조하세요.

SNS에 대한 페이로드 기반 태그 필터링에 대한 자세한 내용은 Introducing payload-based message filtering for Amazon SNS를 참조하세요.

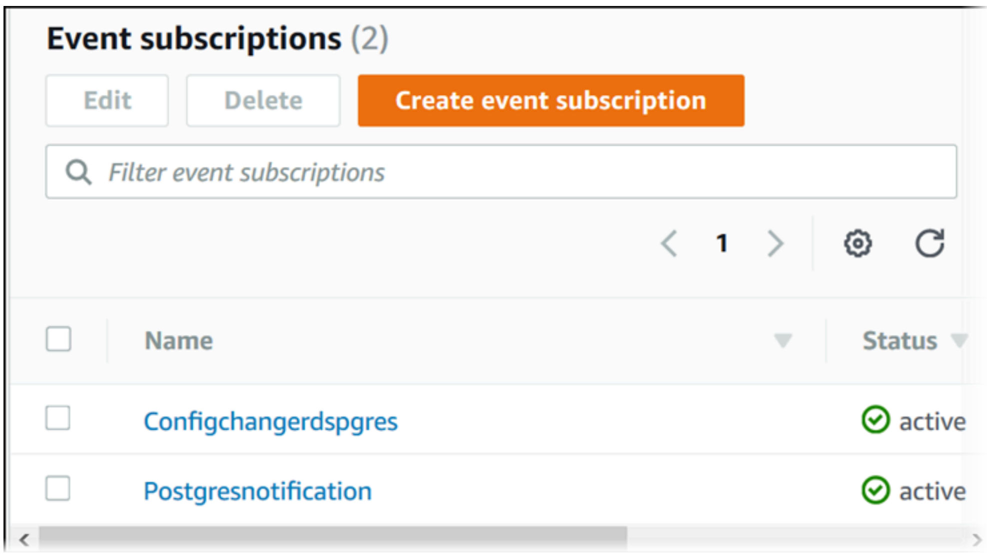
Amazon RDS 이벤트 알림 구독의 목록 표시

현재 Amazon RDS 이벤트 알림 구독을 목록으로 표시할 수 있습니다.

콘솔

현재 Amazon RDS 이벤트 알림 구독을 목록으로 표시하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Event subscriptions]를 선택합니다. [Event subscriptions] 창에 이벤트 알림 구독이 모두 표시됩니다.



AWS CLI

현재 Amazon RDS 이벤트 알림 구독을 표시하려면 AWS CLI `describe-event-subscriptions` 명령을 사용합니다.

Example

다음은 모든 이벤트 구독을 설명하는 예제입니다.

```
aws rds describe-event-subscriptions
```

다음은 `myfirsteventsubscription`을 설명하는 예제입니다.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

현재 Amazon RDS 이벤트 알림 구독을 표시하려면 Amazon RDS API `DescribeEventSubscriptions` 작업을 사용합니다.

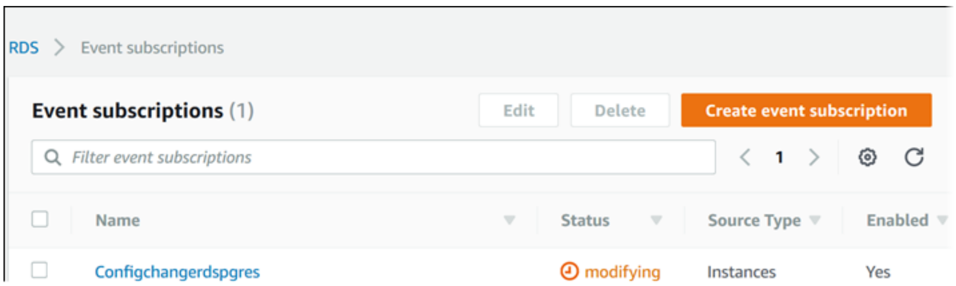
Amazon RDS 이벤트 알림 구독 수정

구독을 생성한 후에는 구독 이름, 소스 식별자, 카테고리 또는 주제 ARN을 변경할 수 있습니다.

콘솔

Amazon RDS 이벤트 알림 구독을 변경하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Event subscriptions]를 선택합니다.
3. [Event subscriptions] 창에서 수정할 구독을 선택한 다음 [Edit]를 선택합니다.
4. 대상 또는 원본 섹션에서 구독을 변경합니다.
5. 편집을 선택합니다. Amazon RDS 콘솔에 현재 구독 변경 중으로 나옵니다.



AWS CLI

Amazon RDS 이벤트 알림 구독을 수정하려면, AWS CLI `modify-event-subscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`

Example

다음 코드를 사용하여 `myeventsubscription`을 사용할 수 있습니다.

Linux, macOS 또는 Unix 대상:

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
    --enabled
```

Windows의 경우:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

Amazon RDS 이벤트를 수정하려면 Amazon RDS API 작업 ModifyEventSubscription을 호출합니다. 다음 필수 파라미터를 포함합니다.

- `SubscriptionName`

Amazon RDS 이벤트 알림 구독에 소스 식별자 추가

소스 식별자(이벤트를 발생시키는 Amazon RDS 소스)를 기존 구독에 추가할 수 있습니다.

콘솔

소스 식별자는 Amazon RDS 콘솔에서 구독 관련 설정을 변경하면서 선택 또는 선택 해제를 통해 쉽게 추가하거나 제거할 수 있습니다. 자세한 내용은 Amazon RDS 이벤트 알림 구독 수정 섹션을 참조하세요.

AWS CLI

Amazon RDS 이벤트 알림 구독에 소스 식별자를 추가하려면 AWS CLI `add-source-identifier-to-subscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`
- `--source-identifier`

Example

다음 예제에서는 `mysqldb` 구독에 소스 식별자 `myrdseventsubscription`를 추가합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Windows의 경우:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Amazon RDS 이벤트 알림 구독에 소스 식별자를 추가하려면 Amazon RDS API `AddSourceIdentifierToSubscription`을 호출합니다. 다음 필수 파라미터를 포함합니다.

- `SubscriptionName`

- `SourceIdentifier`

Amazon RDS 이벤트 알림 구독에서 소스 식별자 제거

임의 소스의 이벤트 알림 메시지를 더 이상 받고 싶지 않을 때는 소스 식별자(이벤트를 발생시키는 Amazon RDS 소스)를 구독에서 제거할 수 있습니다.

콘솔

소스 식별자는 Amazon RDS 콘솔에서 구독 관련 설정을 변경하면서 선택 또는 선택 해제를 통해 쉽게 추가하거나 제거할 수 있습니다. 자세한 내용은 Amazon RDS 이벤트 알림 구독 수정 섹션을 참조하세요.

AWS CLI

Amazon RDS 이벤트 알림 구독에서 소스 식별자를 제거하려면 AWS CLI `remove-source-identifier-from-subscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`
- `--source-identifier`

Example

다음 예제에서는 `mysqldb` 구독에서 소스 식별자 `myrdseventsubscription`를 제거합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Windows의 경우:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Amazon RDS 이벤트 알림 구독에서 소스 식별자를 제거하려면 Amazon RDS API `RemoveSourceIdentifierFromSubscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

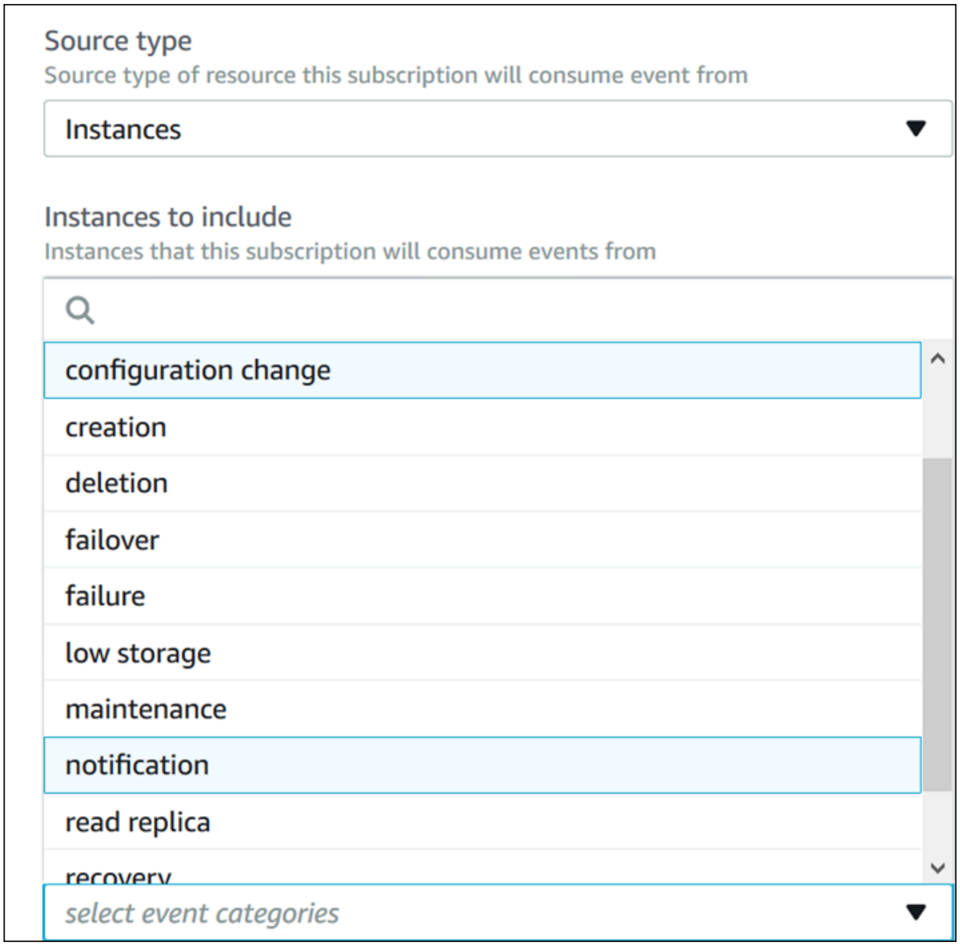
- `SubscriptionName`
- `SourceIdentifier`

Amazon RDS 이벤트 알림 카테고리의 목록 표시

리소스 유형의 이벤트는 모두 여러 카테고리로 그룹화됩니다. 이용 가능한 카테고리 목록을 보려면 다음 절차를 따릅니다.

콘솔

이벤트 알림 구독을 생성 또는 변경할 때는 이벤트 카테고리가 Amazon RDS 콘솔에 표시됩니다. 자세한 내용은 [Amazon RDS 이벤트 알림 구독 수정](#) 섹션을 참조하세요.



AWS CLI

Amazon RDS 이벤트 알림 범주를 표시하려면 AWS CLI `describe-event-categories` 명령을 사용합니다. 이 명령에는 필수 파라미터가 없습니다.

Example

```
aws rds describe-event-categories
```

API

Amazon RDS 이벤트 알림 범주를 표시하려면 Amazon RDS API [DescribeEventCategories](#) 명령을 사용합니다. 이 명령에는 필수 파라미터가 없습니다.

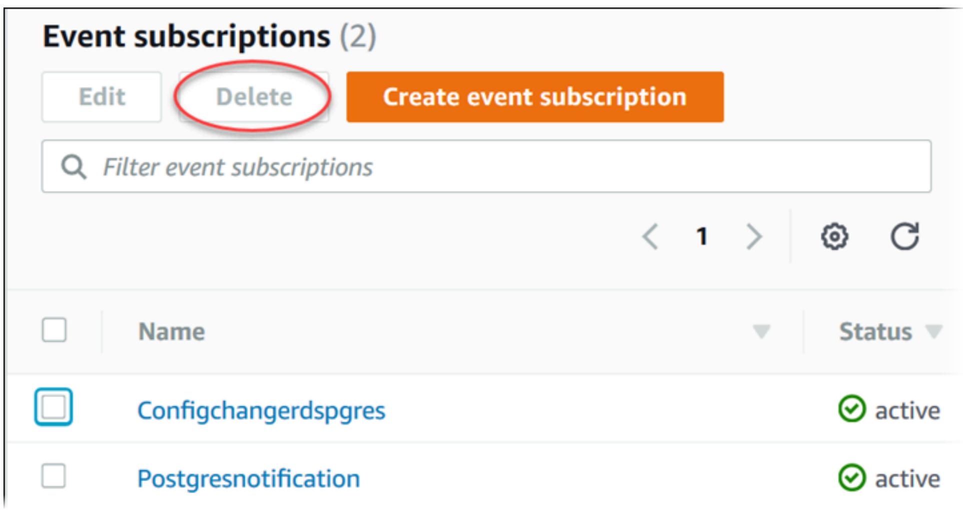
Amazon RDS 이벤트 알림 구독 삭제

필요 없는 구독은 삭제할 수 있습니다. 그러면 해당 주제의 모든 구독자에게는 구독 시 지정한 이벤트 알림 메시지가 발송되지 않습니다.

콘솔

Amazon RDS 이벤트 알림 구독을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [DB Event Subscriptions]를 선택합니다.
3. [My DB Event Subscriptions] 창에서 삭제할 구독을 선택합니다.
4. Delete(삭제)를 선택합니다.
5. Amazon RDS 콘솔에 현재 구독 삭제 중으로 나옵니다.



AWS CLI

Amazon RDS 이벤트 알림 구독을 삭제하려면, AWS CLI `delete-event-subscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`

Example

다음은 `myrdssubscription` 구독을 삭제하는 예제입니다.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

Amazon RDS 이벤트 알림 구독을 삭제하려면, RDS API DeleteEventSubscription 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- SubscriptionName

Amazon Aurora 이벤트에서 트리거되는 규칙 생성

Amazon EventBridge를 사용하면 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 응답할 수 있습니다.

주제

자습서: Amazon EventBridge를 사용하여 DB 인스턴스의 상태 변경 로깅

이 자습서에서는 인스턴스의 상태 변경을 로깅하는 AWS Lambda 함수를 생성합니다. 그런 다음 기존 RDS DB 인스턴스의 상태가 변경될 때마다 함수를 실행하는 규칙을 생성합니다. 이 자습서에서는 일시적으로 종료할 수 있는 실행 중인 작은 테스트 인스턴스가 있다고 가정합니다.

> **Important**
> 실행 중인 프로덕션 DB 인스턴스에서 이 자습서를 수행하지 마세요.

주제

1단계: AWS Lambda 함수 생성

Lambda 함수를 생성하여 상태 변경 이벤트를 기록합니다. 규칙을 생성할 때 이 함수를 지정합니다.

Lambda 함수를 만들려면

1. https://console.aws.amazon.com/lambda/에서 AWS Lambda 콘솔을 엽니다.
2. Lambda를 처음 사용하는 경우 시작 페이지가 표시됩니다. 지금 시작을 선택합니다. 그렇지 않은 경우에는 함수 생성을 선택합니다.
3. 새로 작성을 선택합니다.
4. 함수 생성 페이지에서 다음을 수행합니다.

   a. Lambda 함수의 이름과 설명을 입력합니다. 예를 들어 함수 이름을 **RDSInstanceStateChange**로 지정합니다.
   b. 런타임에서 Node.js 16x를 선택합니다.
   c. 아키텍처에서는 x86_64를 선택합니다.
   d. 실행 역할에서는 다음 중 하나를 수행합니다.
      - 기본 Lambda 권한을 가진 새 역할 생성을 선택합니다.
      - 기존 역할에서는 기존 역할 사용을 선택합니다. 사용하려는 역할을 선택합니다.
   e. 함수 생성을 선택합니다.

5. [RDSInstanceStateChange] 페이지에서 다음을 수행합니다.

   a. 코드 소스에서 index.js를 선택합니다.
   b. index.js 창에서 기존 코드를 삭제합니다.
   c. 다음 코드를 입력합니다.

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

   d. [Deploy]를 선택합니다.

2단계: 규칙 생성

Amazon RDS 인스턴스를 시작할 때마다 Lambda 함수를 실행하는 규칙을 생성합니다.

EventBridge 규칙을 만들려면

1. https://console.aws.amazon.com/events/에서 Amazon EventBridge 콘솔을 엽니다.
2. 탐색 창에서 규칙을 선택합니다.
3. 규칙 생성을 선택합니다.
4. 규칙에 대해 이름과 설명을 입력하십시오. 예를 들면 **RDSInstanceStateChangeRule**을 입력합니다.
5. 이벤트 패턴이 있는 규칙을 선택한 후다음을 선택합니다.
6. 이벤트 소스에서 AWS 이벤트 또는 EventBridge 파트너 이벤트를 선택합니다.

7. 이벤트 패턴 섹션까지 아래로 스크롤합니다.
8. 이벤트 소스에서 AWS 서비스를 선택합니다.
9. AWS 서비스에서는 Relational Database Service(RDS)를 선택합니다.
10. 이벤트 유형에서 RDS DB 인스턴스 이벤트를 선택합니다.
11. 기본 이벤트 패턴을 그대로 둡니다. 그리고 다음을 선택합니다.
12. 대상 유형에서 AWS 서비스를 선택합니다.
13. 대상 선택에서 Lambda 함수를 선택합니다.
14. 함수에서는 생성한 Lambda 함수를 선택합니다. 다음을 선택합니다.
15. 태그 구성에서는 다음을 선택합니다.
16. 규칙의 단계를 검토하십시오. 그런 다음 규칙 생성을 선택합니다.

3단계: 규칙 테스트

규칙을 테스트하려면 RDS DB 인스턴스를 종료합니다. 인스턴스가 종료될 때까지 몇 분 기다린 후에 Lambda 함수가 호출되었는지 확인합니다.

DB 인스턴스를 중지하여 규칙을 테스트하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. RDS DB 인스턴스를 중지합니다.
3. https://console.aws.amazon.com/events/에서 Amazon EventBridge 콘솔을 엽니다.
4. 탐색 창에서 [규칙(Rules)]을 선택하고 생성한 규칙의 이름을 선택합니다.
5. 규칙 세부 정보에서 모니터링을 선택합니다.

   그러면 Amazon CloudWatch 콘솔로 리디렉션됩니다. 리디렉션되지 않은 경우 CloudWatch에서 지표 보기를 클릭합니다.
6. [모든 지표(All metrics)]에서 생성한 규칙의 이름을 선택합니다.

   그래프에 규칙이 호출된 것으로 표시되어야 합니다.
7. 탐색 창에서 로그 그룹을 선택합니다.
8. Lambda 함수에 대한 로그 그룹 이름(/aws/lambda/***function-name***)을 선택합니다.
9. 로그 스트림 이름을 선택하여 시작한 인스턴스에서 함수를 통해 제공된 데이터를 확인합니다. 다음과 유사한 수신된 이벤트가 표시되어야 합니다.

```
{
```

```
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```


JSON 형식의 RDS 이벤트에 대한 자세한 예는 Aurora에 대한 이벤트 개요 단원을 참조하세요.

10. (선택 사항) 작업이 완료되면 Amazon RDS 콘솔을 열고 중지한 인스턴스를 시작할 수 있습니다.

Amazon RDS 이벤트 카테고리 및 Aurora용 이벤트 메시지

Amazon RDS는 Amazon RDS 콘솔, AWS CLI 또는 API를 사용하여 구독할 수 있는 이벤트 카테고리가 매우 많습니다.

주제

DB 클러스터 이벤트

다음 표에는 DB 클러스터가 원본 유형일 때 이벤트 카테고리와 이벤트 목록이 나와 있습니다.

> **Note**
> DB 클러스터 이벤트 유형에 Aurora Serverless의 이벤트 범주가 없습니다. Aurora Serverless 이벤트의 범위는 RDS-EVENT-0141에서 RDS-EVENT-0149입니다.

Category	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0016	마스터 보안 인증 정보를 재설정하세요.	
구성 변경	RDS-EVENT-0179	Database Activity Stream이 데이터베이스 클러스터에서 시작됩니다.	자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 섹션을 참조하세요.

Category	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0180	데이터베이스 클러스터에서 Database Activity Stream이 중지되었습니다.	자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 섹션을 참조하세요.
생성	RDS-EVENT-0170	DB 클러스터가 생성되었습니다.	
삭제	RDS-EVENT-0171	DB 클러스터가 삭제되었습니다.	
장애 조치	RDS-EVENT-0069	클러스터 장애 조치가 실패했습니다. 클러스터 인스턴스의 상태를 확인하고 다시 시도하세요.	
장애 조치	RDS-EVENT-0070	이전 프라이머리(*##*)를 다시 승격하는 중입니다.	
장애 조치	RDS-EVENT-0071	DB 인스턴스(*##*)로 장애 조치를 완료했습니다.	
장애 조치	RDS-EVENT-0072	DB 인스턴스(*##*)에 대한 동일한 AZ 장애 조치를 시작했습니다.	
장애 조치	RDS-EVENT-0073	DB 인스턴스(*##*)에 대한 크로스 AZ 장애 조치를 시작했습니다.	

Category	RDS 이벤트 ID	메시지	참고
실패	RDS-EVENT-0083	Amazon RDS에서 DB 클러스터(##)의 Amazon S3 버킷에 액세스하기 위한 보안 인증 정보를 생성할 수 없습니다. 이는 계정에 S3 스냅샷 수집 IAM 역할이 올바르게 구성되지 않았거나 지정된 Amazon S3 버킷을 찾을 수 없기 때문입니다. 자세한 내용은 Amazon RDS 설명서의 문제 해결 섹션을 참조하세요.	자세한 내용은 Percona XtraBackup과 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션 을 참조하십시오.
결함	RDS-EVENT-0143	다음과 같은 이유로 DB 클러스터를 ##에서 ##으로 규모 조정하지 못했습니다. ##.	Aurora Serverless DB 클러스터에 대해 조정이 실패하였습니다.
실패	RDS-EVENT-0354	호환되지 않는 리소스 때문에 DB 클러스터를 생성할 수 없습니다. ###.	###에는 실패에 대한 세부 정보가 포함되어 있습니다.
실패	RDS-EVENT-0355	리소스 제한이 충분하지 않아 DB 클러스터를 생성할 수 없습니다. ###.	###에는 실패에 대한 세부 정보가 포함되어 있습니다.
글로벌 장애 조치	RDS-EVENT-0181	리전(##)의 DB 클러스터(##)에 대한 글로벌 전환이 시작되었습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치'). DB 클러스터에서 다른 작업이 실행되고 있기 때문에 프로세스가 지연될 수 있습니다.

Category	RDS 이벤트 ID	메시지	참고
글로벌 장애 조치	RDS-EVENT-0182	리전(##)의 이전 프라이머리 DB 클러스터(##)가 성공적으로 종료되었습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치'). 글로벌 데이터베이스의 이전 기본 인스턴스는 쓰기를 허용하지 않습니다. 모든 볼륨이 동기화됩니다.
글로벌 장애 조치	RDS-EVENT-0183	전역 클러스터 멤버 간의 데이터 동기화를 기다리는 중입니다. 현재 프라이머리 DB 클러스터보다 지연되고 있습니다. ##.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치'). 글로벌 데이터베이스 장애 조치의 동기화 단계 중에 복제 지연이 발생합니다.
글로벌 장애 조치	RDS-EVENT-0184	리전(##)의 새로운 프라이머리 DB 클러스터(##)가 성공적으로 승격되었습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치'). 글로벌 데이터베이스의 볼륨 토폴로지가 새 기본 볼륨으로 다시 설정됩니다.
글로벌 장애 조치	RDS-EVENT-0185	리전(##)의 DB 클러스터(##)에 대한 글로벌 전환이 완료되었습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치'). 글로벌 데이터베이스 전환이 기본 DB 클러스터에서 완료됩니다. 장애 조치가 완료된 후 복제본이 온라인 상태로 전환되는 데 오랜 시간이 걸릴 수 있습니다.

Category	RDS 이벤트 ID	메시지	참고
글로벌 장애 조치	RDS-EVENT-0186	리전(##)의 DB 클러스터(##)에 대한 글로벌 전환이 취소되었습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치').
글로벌 장애 조치	RDS-EVENT-0187	리전(##)의 DB 클러스터(##)에 대한 글로벌 전환이 실패했습니다.	이 이벤트는 전환 작업에 해당합니다(이전 명칭: '계획된 관리형 장애 조치').
글로벌 장애 조치	RDS-EVENT-0238	리전(##)의 DB 클러스터(##)에 대한 글로벌 장애 조치가 완료되었습니다.	
글로벌 장애 조치	RDS-EVENT-0239	리전(##)의 DB 클러스터(##)에 대한 글로벌 장애 조치가 실패했습니다.	
글로벌 장애 조치	RDS-EVENT-0240	리전(##)의 DB 클러스터(##)에 대한 재동기화 구성 요소가 글로벌 장애 조치 후에 시작되었습니다.	
글로벌 장애 조치	RDS-EVENT-0241	리전(##)의 DB 클러스터(##)에 대한 재동기화 구성 요소가 글로벌 장애 조치 후에 완료되었습니다.	
유지 관리	RDS-EVENT-0156	DB 클러스터에 사용 가능한 DB 엔진 마이너 버전 업그레이드가 있습니다.	
유지 관리	RDS-EVENT-0173	데이터베이스 클러스터 엔진 버전이 업그레이드되었습니다.	DB 클러스터의 패치 적용이 완료되었습니다.

Category	RDS 이벤트 ID	메시지	참고
유지 관리	RDS-EVENT-0176	데이터베이스 클러스터 엔진 메이저 버전이 업그레이드되었습니다.	
유지 관리	RDS-EVENT-0177	데이터베이스 클러스터 업그레이드가 진행 중입니다.	
유지 관리	RDS-EVENT-0286	데이터베이스 클러스터 엔진 버전 업그레이드가 시작되었습니다.	
유지 관리	RDS-EVENT-0287	운영 체제 업그레이드 요구사항이 감지되었습니다.	
유지 관리	RDS-EVENT-0288	클러스터 운영 체제 업그레이드를 시작하는 중입니다.	
유지 관리	RDS-EVENT-0289	클러스터 운영 체제 업그레이드가 완료되었습니다.	
유지 관리	RDS-EVENT-0363	업그레이드 준비 진행 중: *cluster_name*	DB 클러스터에 대한 업그레이드 사전 점검이 시작되었습니다.
알림	RDS-EVENT-0076	*##*에서 *##*으로 마이그레이션하지 못했습니다. 이유: *##*.	Aurora DB 클러스터로 마이그레이션하지 못했습니다.
알림	RDS-EVENT-0077	*##.##*을 InnoDB로 변환하지 못했습니다. 이유: *##*.	Aurora DB 클러스터로 마이그레이션하는 중에 원본 데이터베이스의 테이블을 InnoDB로 변환하지 못했습니다.

Category	RDS 이벤트 ID	메시지	참고
알림	RDS-EVENT-0085	인스턴스(##)의 상태가 ##이므로 DB 클러스터(##)를 업그레이드할 수 없습니다. 문제를 해결하거나 인스턴스를 삭제한 후 다시 시도하세요.	Aurora DB 클러스터를 패치하는 동안 오류가 발생했습니다. 인스턴스 상태를 확인하고 문제를 해결한 다음 다시 시도하세요. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리](#) 섹션을 참조하세요.
알림	RDS-EVENT-0141	다음과 같은 이유로 DB 클러스터를 ##에서 ##으로 규모 조정하는 중입니다. ##.	Aurora Serverless DB 클러스터에 대해 조정이 시작되었습니다.
알림	RDS-EVENT-0142	DB 클러스터가 ##에서 ##으로 규모 조정되었습니다.	Aurora Serverless DB 클러스터에 대해 조정이 완료되었습니다.
알림	RDS-EVENT-0144	DB 클러스터를 일시 중지하는 중입니다.	Aurora Serverless DB 클러스터에서 자동 일시 중지가 시작되었습니다.
알림	RDS-EVENT-0145	DB 클러스터가 일시 중지되었습니다.	Aurora Serverless DB 클러스터가 일시 중지되었습니다.
알림	RDS-EVENT-0146	DB 클러스터 일시 중지가 취소되었습니다.	Aurora Serverless DB 클러스터 일시 중지가 취소되었습니다.
알림	RDS-EVENT-0147	DB 클러스터를 재개하는 중입니다.	Aurora Serverless DB 클러스터 재개 작업이 시작되었습니다.
알림	RDS-EVENT-0148	DB 클러스터가 재개되었습니다.	Aurora Serverless DB 클러스터 재개 작업이 완료되었습니다.

Category	RDS 이벤트 ID	메시지	참고
알림	RDS-EVENT-0149	DB 클러스터가 ##에서 ##으로 규모 조정되었지만, 다음 이유로 확장이 원활하지 않습니다. ##.	Aurora Serverless DB 클러스터에서 강제 적용 옵션을 통해 원활한 조정이 완료되었습니다. 필요에 따라 연결이 중단되었을 수 있습니다.
알림	RDS-EVENT-0150	DB 클러스터가 중지되었습니다.	
알림	RDS-EVENT-0151	DB 클러스터가 시작되었습니다.	
알림	RDS-EVENT-0152	DB 클러스터를 중지하지 못했습니다.	
알림	RDS-EVENT-0153	DB 클러스터가 중지 최대 허용 시간 초과로 인해 시작 중입니다.	
알림	RDS-EVENT-0172	클러스터 이름을 ##에서 ##으로 변경했습니다.	
알림	RDS-EVENT-0234	내보내기 작업에 실패했습니다.	DB 클러스터 내보내기 작업이 실패했습니다.
알림	RDS-EVENT-0235	내보내기 작업이 취소되었습니다.	DB 클러스터 내보내기 작업이 취소되었습니다.
알림	RDS-EVENT-0236	내보내기 작업이 완료되었습니다.	DB 클러스터 내보내기 작업이 완료되었습니다.

DB 인스턴스 이벤트

다음 표는 DB 인스턴스가 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

범주	RDS 이벤트 ID	메시지	참고
가용성	RDS-EVENT-0004	DB 인스턴스 종료.	
가용성	RDS-EVENT-0006	DB 인스턴스가 다시 시작되었습니다.	
가용성	RDS-EVENT-0022	MySQL을 다시 시작하는 중 오류가 발생했습니다. *###*.	Aurora MySQL 또는 RDS for MariaDB를 다시 시작하는 중 오류가 발생했습니다.
역추적	RDS-EVENT-0131	실제 역추적 기간은 지정한 대상 역추적 기간보다 작습니다. 대상 역추적 기간에서 시간에 해당하는 숫자를 줄이는 것을 고려해 보십시오.	역추적에 대한 자세한 내용은 [Aurora DB 클러스터 역추적](#) 단원을 참조하십시오.
역추적	RDS-EVENT-0132	실제 역추적 기간은 대상 역추적 기간과 같습니다.	
구성 변경	RDS-EVENT-0011	DBParameterGroup *##*을 사용하도록 업데이트되었습니다.	
구성 변경	RDS-EVENT-0012	수정을 데이터베이스 인스턴스 클래스에 적용.	
구성 변경	RDS-EVENT-0014	DB 인스턴스 클래스에 대한 수정 적용이 완료되었습니다.	
구성 변경	RDS-EVENT-0017	할당된 스토리지에 대한 수정 적용이 완료되었습니다.	
구성 변경	RDS-EVENT-0025	다중 AZ DB 인스턴스로 전환하기 위한 수정 적용이 완료되었습니다.	

범주	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0029	표준(단일 AZ) DB 인스턴스로 전환하기 위한 수정 적용이 완료되었습니다.	
구성 변경	RDS-EVENT-0033	마스터 사용자 이름과 일치하는 사용자가 ##명 있습니다. 특정 호스트에 연결되지 않은 사용자만 재설정합니다.	
구성 변경	RDS-EVENT-0067	암호를 재설정할 수 없습니다. 오류 정보: ###.	
구성 변경	RDS-EVENT-0078	모니터링 간격이 ##로 변경되었습니다.	Enhanced Monitoring 구성이 변경되었습니다.
구성 변경	RDS-EVENT-0092	DB 파라미터 그룹의 업데이트를 완료했습니다.	
생성	RDS-EVENT-0005	생성된 DB 인스턴스.	
삭제	RDS-EVENT-0003	DB 인스턴스가 삭제되었습니다.	
실패	RDS-EVENT-0035	데이터베이스 인스턴스가 ##로 전환되었습니다. ###.	DB 인스턴스에 잘못된 파라미터가 있습니다. 예를 들어 이 인스턴스 클래스의 메모리 관련 파라미터가 너무 높게 설정되어 있어 DB 인스턴스가 시작되지 않는 경우 사용자가 할 수 있는 작업은 메모리 파라미터를 수정하고 DB 인스턴스를 재부팅하는 것입니다.

범주	RDS 이벤트 ID	메시지	참고
실패	RDS-EVENT-0036	데이터베이스 인스턴스가 ## 상태입니다. ###.	DB 인스턴스가 호환되지 않는 네트워크에 있습니다. 특정 서브넷 ID 중 일부가 잘못되었거나 존재하지 않습니다.
실패	RDS-EVENT-0079	Amazon RDS가 확장 모니터링을 위한 보안 인증 정보를 생성할 수 없으며 이 기능은 비활성화되었습니다 . 이는 rds-monitoring-role 이 존재하지 않고 계정에 올바르게 구성되어 있지 않기 때문일 수 있습니다. 자세한 내용은 Amazon RDS 설명서의 문제 해결 섹션을 참조하세요.	확장 모니터링을 활성화하려면 확장 모니터링 IAM 역할이 있어야 합니다. IAM 역할 생성에 대한 자세한 내용은 Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면 섹션을 참조하세요.
실패	RDS-EVENT-0080	Amazon RDS가 인스턴스(##)에 확장 모니터링을 구성할 수 없으며 이 기능은 비활성화되었습니다. 이는 rds-monitoring-role이 존재하지 않고 계정에 올바르게 구성되어 있지 않기 때문일 수 있습니다. 자세한 내용은 Amazon RDS 설명서의 문제 해결 섹션을 참조하세요.	구성을 변경하는 동안 오류가 발생하여 확장 모니터링이 비활성화되었습니다 . 확장 모니터링 IAM 역할이 잘못 구성된 것 같습니다. 확장 모니터링 IAM 역할을 생성하는 방법에 대한 자세한 내용은 Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면 섹션을 참조하세요.

범주	RDS 이벤트 ID	메시지	참고
결함	RDS-EVENT-0082	Amazon RDS에서 DB 인스턴스(##)의 Amazon S3 버킷에 액세스하기 위한 보안 인증 정보를 생성할 수 없습니다. 이는 계정에 S3 스냅샷 수집 IAM 역할이 올바르게 구성되지 않았거나 지정된 Amazon S3 버킷을 찾을 수 없기 때문입니다. 자세한 내용은 Amazon RDS 설명서의 문제 해결 섹션을 참조하세요.	Aurora가 Amazon S3 버킷에서 백업 데이터를 복사할 수 없습니다. Amazon S3 버킷에 액세스하기 위한 Aurora의 권한이 잘못 구성된 것 같습니다. 자세한 내용은 Percona XtraBackup과 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션 을 참조하십시오.
실패	RDS-EVENT-0254	이 고객 계정의 기본 스토리지 할당량이 한도를 초과했습니다. 인스턴스에서 규모 조정이 진행되도록 허용된 스토리지 할당량을 늘리세요.	
실패	RDS-EVENT-0353	리소스 제한이 충분하지 않아 DB 인스턴스를 생성할 수 없습니다. ###.	###에는 실패에 대한 세부 정보가 포함되어 있습니다.
적은 스토리지	RDS-EVENT-0007	할당된 스토리지가 모두 소진되었습니다. 해결하려면 추가 스토리지를 할당하세요.	DB 인스턴스에 할당된 스토리지를 모두 사용했습니다. 이 문제를 해결하려면 DB 인스턴스에 스토리지를 추가 할당하세요. 자세한 내용은 RDS FAQ 단원을 참조하십시오. [Free Storage Space] 측정치를 사용하여 DB 인스턴스에 대한 스토리지 공간을 모니터링할 수 있습니다.

범주	RDS 이벤트 ID	메시지	참고
적은 스토리지	RDS-EVENT-0089	DB 인스턴스(##)의 사용 가능한 스토리지 용량이 프로비저닝된 스토리지의 ##%로 낮습니다[프로비저닝된 스토리지: ##, 사용 가능한 스토리지: ##]. 이 문제를 해결하려면 프로비저닝된 스토리지를 늘리는 것이 좋습니다.	DB 인스턴스가 할당된 스토리지의 90% 이상을 사용하였습니다. [Free Storage Space] 측정치를 사용하여 DB 인스턴스에 대한 스토리지 공간을 모니터링할 수 있습니다.
적은 스토리지	RDS-EVENT-0227	Aurora 클러스터의 스토리지가 ##테라바이트만 남아 있어 위험할 정도로 부족합니다. 클러스터의 스토리지 로드를 줄이기 위한 조치를 취하세요.	Aurora 스토리지 하위 시스템의 공간이 부족합니다.
유지 관리	RDS-EVENT-0026	DB 인스턴스에 오프라인 패치를 적용하는 중입니다.	DB 인스턴스의 오프라인 유지 관리가 진행 중입니다. 따라서 현재 DB 인스턴스는 사용할 수 없습니다.
유지 관리	RDS-EVENT-0027	DB 인스턴스에 대한 오프라인 패치 적용이 완료되었습니다.	DB 인스턴스의 오프라인 유지 관리가 완료되었습니다. 이제 DB 인스턴스를 사용할 수 있습니다.
유지 관리	RDS-EVENT-0047	데이터베이스 인스턴스가 패치되었습니다.	
유지 관리	RDS-EVENT-0155	DB 인스턴스에 사용 가능한 DB 엔진 마이너 버전 업그레이드가 있습니다.	
유지 관리	RDS-EVENT-0178	데이터베이스 인스턴스 업그레이드가 진행 중입니다.	

범주	RDS 이벤트 ID	메시지	참고
알림	RDS-EVENT-0044	*message*	운영자가 발행한 알림입니다. 자세한 내용은 이벤트 메시지 단원을 참조하십시오.
알림	RDS-EVENT-0048	이 인스턴스에는 먼저 업그레이드해야 하는 읽기 전용 복제본이 있으므로 데이터베이스 엔진 업그레이드가 지연됩니다.	DB 인스턴스의 패치 작업이 지연되었습니다.
알림	RDS-EVENT-0087	DB 인스턴스가 중지되었습니다.	
알림	RDS-EVENT-0088	DB 인스턴스가 시작되었습니다.	
읽기 전용 복제본	RDS-EVENT-0045	복제가 중지되었습니다.	스토리지 부족으로 DB 인스턴스 복제가 중지되었습니다. 복제를 계속하려면 스토리지의 규모를 조정하거나 다시 실행 로그의 최대 크기를 줄이세요. 크기가 *amount*MiB인 다시 실행 로그를 수용하려면 최소한 *amount*MiB의 사용 가능한 스토리지가 필요합니다.

범주	RDS 이벤트 ID	메시지	참고
읽기 전용 복제본	RDS-EVENT-0046	읽기 전용 복제본에 대한 복제가 재개되었습니다.	이 메시지는 읽기 전용 복제본을 처음 생성할 때 나타나거나 복제 기능의 정상 여부를 확인하는 모니터링 메시지로 나타납니다. `RDS-EVENT-0045` 알림 후에 이 메시지가 표시되면 오류가 표시된 후 또는 복제가 중단된 후 복제가 재개된 것입니다.
읽기 전용 복제본	RDS-EVENT-0057	복제 스트리밍이 종료되었습니다.	
복구	RDS-EVENT-0020	DB 인스턴스 복구가 시작되었습니다. 복구 시간은 복구할 데이터 용량에 따라 달라집니다.	
복구	RDS-EVENT-0021	DB 인스턴스 복구가 완료되었습니다.	
복구	RDS-EVENT-0023	긴급 스냅샷 요청이 있습니다. *###*.	수동 백업을 요청하였지만 Amazon RDS가 현재 DB 스냅샷을 생성 중입니다. 따라서 Amazon RDS가 DB 스냅샷 생성을 완료한 후에 다시 요청하십시오.
복구	RDS-EVENT-0052	다중 AZ 인스턴스 복구가 시작되었습니다.	복구 시간은 복구할 데이터 용량에 따라 달라집니다.
복구	RDS-EVENT-0053	다중 AZ 인스턴스 복구가 완료되었습니다. 장애 조치 또는 활성화가 보류 중입니다.	

범주	RDS 이벤트 ID	메시지	참고
복구	RDS-EVENT-0361	대기 DB 인스턴스 복구가 시작되었습니다.	복구 프로세스 중에 대기 DB 인스턴스가 다시 구축됩니다. 복구 프로세스 중에는 데이터베이스 성능이 영향을 받습니다.
복구	RDS-EVENT-0362	대기 DB 인스턴스 복구가 완료되었습니다.	복구 프로세스 중에 대기 DB 인스턴스가 다시 구축됩니다. 복구 프로세스 중에는 데이터베이스 성능이 영향을 받습니다.
복원	RDS-EVENT-0019	DB 인스턴스 *##*에서 *##*으로 복원되었습니다.	DB 인스턴스가 특정 시점으로 백업에서 복원되었습니다.
보안 패치	RDS-EVENT-0230	DB 인스턴스의 시스템 업데이트를 사용할 수 있습니다. 업데이트 적용에 대한 자세한 내용은 RDS 사용 설명서의 'DB 인스턴스 관리'를 참조하세요.	새 운영 체제 패치를 사용할 수 있습니다. DB 인스턴스의 운영 체제의 새로운 버전, 마이너 버전, 운영 체제 업데이트를 사용할 수 있습니다. 업데이트에 적용에 대한 자세한 내용은 ??? 단원을 참조하세요.

DB 파라미터 그룹 이벤트

다음 표는 DB 파라미터 그룹이 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

Category	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0037	메서드 `method` 적용을 사용하여 파라미터 `name`을	

Category	RDS 이벤트 ID	메시지	참고
		*value*로 업데이트했습니다.	

DB 보안 그룹 이벤트

다음 표는 DB 보안 그룹이 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

> **Note**
>
> DB 보안 그룹 EC2-Classic의 리소스입니다. EC2-Classic은 2022년 8월 15일에 사용 중지되었습니다. EC2-Classic에서 VPC로 마이그레이션하지 않은 경우 가능한 한 빨리 마이그레이션하는 것이 좋습니다. 자세한 내용은 Amazon EC2 사용 설명서의 Migrate from EC2-Classic to a VPC(EC2-Classic에서 VPC로 마이그레이션) 및 EC2-Classic Networking is Retiring – Here's How to Prepare(EC2-Classic 네트워킹 지원 중단에 대비하는 방법) 블로그 게시물을 참조하세요.

범주	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0038	보안 그룹에 변경 사항을 적용했습니다.	
실패	RDS-EVENT-0039	*###*로서의 권한 부여를 취소하는 중입니다.	*###*가 소유한 보안 그룹이 없습니다. 보안 그룹에 대한 권한 부여가 잘못되었기 때문에 취소되었습니다.

DB 클러스터 스냅샷 이벤트

다음 표에는 DB 클러스터 스냅샷이 원본 유형일 때 이벤트 카테고리와 이벤트 목록이 나와 있습니다.

범주	RDS 이벤트 ID	메시지	참고
백업	RDS-EVENT-0074	수동 클러스터 스냅샷을 생성하는 중입니다.	
백업	RDS-EVENT-0075	수동 클러스터 스냅샷이 생성되었습니다.	
알림	RDS-EVENT-0162	클러스터 스냅샷 내보내기 작업이 실패했습니다.	
알림	RDS-EVENT-0163	클러스터 스냅샷 내보내기 작업이 취소되었습니다.	
알림	RDS-EVENT-0164	클러스터 스냅샷 내보내기 작업이 완료되었습니다.	
백업	RDS-EVENT-0168	자동화된 클러스터 스냅샷을 생성하는 중입니다.	
백업	RDS-EVENT-0169	자동화된 클러스터 스냅샷이 생성되었습니다.	

RDS 프록시 이벤트

다음 표에서는 RDS 프록시가 소스 유형일 때 이벤트 범주와 이벤트 목록을 보여줍니다.

범주	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0204	RDS가 DB 프록시(##)를 수정했습니다.	
구성 변경	RDS-EVENT-0207	RDS가 DB 프록시(##)의 엔드포인트를 수정했습니다.	
구성 변경	RDS-EVENT-0213	RDS가 DB 인스턴스 추가를 감지하여 DB 프록시(##)의	

범주	RDS 이벤트 ID	메시지	참고
		대상 그룹에 자동으로 추가했습니다.	
구성 변경	RDS-EVENT-0213	RDS가 DB 인스턴스(*##*)의 생성을 감지하여 DB 프록시(*##*)의 대상 그룹(*##*)에 자동으로 추가했습니다.	
구성 변경	RDS-EVENT-0214	RDS가 DB 인스턴스(*##*)의 삭제를 감지하여 DB 프록시(*##*)의 대상 그룹(*##*)에서 자동으로 제거했습니다.	
구성 변경	RDS-EVENT-0215	RDS가 DB 클러스터(*##*)의 삭제를 감지하여 DB 프록시(*##*)의 대상 그룹(*##*)에서 자동으로 제거했습니다.	
생성	RDS-EVENT-0203	RDS가 DB 프록시(*##*)를 생성했습니다.	
생성	RDS-EVENT-0206	RDS가 DB 프록시(*##*)에 대한 엔드포인트(*##*)을 생성했습니다.	
삭제	RDS-EVENT-0205	RDS가 DB 프록시(*##*)를 삭제했습니다.	
삭제	RDS-EVENT-0208	RDS가 DB 프록시(*##*)에 대한 엔드포인트(*##*)을 삭제했습니다.	

범주	RDS 이벤트 ID	메시지	참고
실패	RDS-EVENT-0243	서브넷(##)에서 사용할 수 있는 IP 주소가 충분하지 않기 때문에 RDS가 프록시(##)의 용량을 프로비저닝하지 못했습니다. 이러한 문제를 해결하려면 RDS 프록시 설명서의 권장 사항에 따라 서브넷에 최소한의 미사용 IP 주소 개수가 있는지 확인하세요.	권장 인스턴스 클래스 수를 확인하려면 IP 주소 용량 계획 섹션을 참조하세요.
실패	RDS-EVENT-0275	RDS가 DB 프록시 ##에 대한 일부 연결을 제한했습니다. 클라이언트에서 프록시로의 동시 연결 요청 수가 제한을 초과했습니다.	

블루/그린 배포 이벤트

다음 표는 블루/그린 배포가 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

블루/그린 배포에 대한 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 섹션을 참조하세요.

범주	Amazon RDS 이벤트 ID	메시지	참고
생성	RDS-EVENT-0244	블루/그린 배포 작업이 완료되었습니다. 그린 환경 데이터베이스를 추가로 수정하거나 배포로 전환할 수 있습니다.	
실패	RDS-EVENT-0245	(소스/타겟) DB (인스턴스/클러스터)를 찾을 수 없어서 블	

범주	Amazon RDS 이벤트 ID	메시지	참고
		루/그린 배포를 생성하지 못했습니다.	
삭제	RDS-EVENT-0246	블루/그린 배포가 삭제되었습니다.	
알림	RDS-EVENT-0247	##에서 ##으로 전환이 시작되었습니다.	
알림	RDS-EVENT-0248	블루/그린 배포에서 전환이 완료되었습니다.	
실패	RDS-EVENT-0249	블루/그린 배포에서 전환이 취소되었습니다.	
알림	RDS-EVENT-0259	DB 클러스터 ##에서 ##으로 전환이 시작되었습니다.	
알림	RDS-EVENT-0260	DB 클러스터 ##에서 ##으로 전환이 완료되었습니다. ##는 ##-##으로, ##은 ##로 이름을 변경했습니다.	
실패	RDS-EVENT-0261	DB 클러스터 ##에서 ##으로의 전환이 ##(으)로 인해 취소되었습니다.	
알림	RDS-EVENT-0311	DB 클러스터 ##에서 ##으로의 전환을 위한 시퀀스 동기화가 시작되었습니다. 시퀀스를 사용할 때 전환하면 가동 중지가 길어질 수 있습니다.	

범주	Amazon RDS 이벤트 ID	메시지	참고
알림	RDS-EVENT-0312	DB 클러스터 ##에서 ##으로의 전환을 위한 시퀀스 동기화가 완료되었습니다.	
실패	RDS-EVENT-0314	시퀀스 동기화에 실패하여 DB 클러스터 ##에서 ##으로의 전환을 위한 시퀀스 동기화가 취소되었습니다.	

Amazon Aurora 로그 파일 모니터링

모든 RDS 데이터베이스 엔진은 감사 및 문제 해결을 위해 액세스할 수 있는 로그를 생성합니다. 로그 유형은 데이터베이스 엔진에 따라 다릅니다.

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 DB 인스턴스 로그에 액세스할 수 있습니다. 트랜잭션 로그를 보거나 다운로드할 수 없습니다.

> **Note**
>
> 경우에 따라 로그에 숨겨진 데이터가 포함됩니다. 따라서 AWS Management Console은 콘텐츠를 로그 파일에 표시할 수 있지만 다운로드할 때 로그 파일이 비어 있을 수 있습니다.

주제

데이터베이스 로그 파일 보기 및 나열

AWS Management Console을 사용하여 Amazon Aurora DB 엔진에 대한 데이터베이스 로그 파일을 볼 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 다운로드하거나 모니터링할 수 있는 로그 파일을 나열할 수 있습니다.

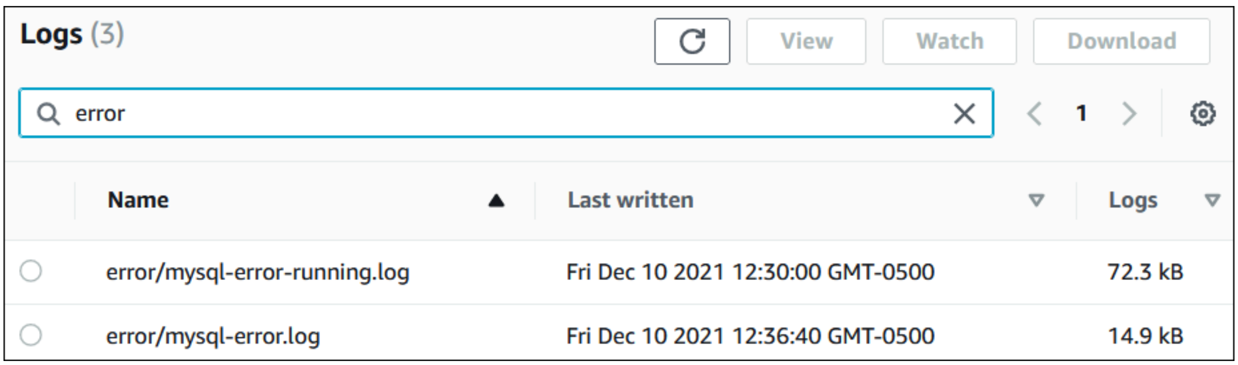
> **Note**
>
> RDS 콘솔에서 Aurora Serverless v1 DB 클러스터의 로그 파일을 볼 수 없습니다. 단, Amazon CloudWatch 콘솔(https://console.aws.amazon.com/cloudwatch/)에서는 볼 수 있습니다.

콘솔

데이터베이스 로그 파일을 보려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 보고자 하는 로그 파일을 보유한 DB 인스턴스의 이름을 선택합니다.
4. 로그 및 이벤트 탭을 선택합니다.
5. 아래로 스크롤하여 로그 섹션을 찾습니다.
6. (선택 사항) 검색어를 입력하여 결과를 필터링합니다.

   다음 예에는 텍스트 `error`로 필터링된 로그가 나와 있습니다.

7. 표시할 로그를 선택한 다음 보기(View)를 선택합니다.

AWS CLI

DB 인스턴스에 사용 가능한 데이터베이스 로그 파일을 나열하려면 AWS CLI `describe-db-log-files` 명령을 사용합니다.

다음 예에서는 `my-db-instance`라는 DB 인스턴스에 대한 로그 파일 목록을 반환합니다.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

DB 인스턴스에 사용 가능한 데이터베이스 로그 파일을 나열하려면 Amazon RDS API `DescribeDBLogFiles` 작업을 사용합니다.

데이터베이스 로그 파일 다운로드

AWS Management Console, AWS CLI 또는 API를 사용하여 데이터베이스 로그 파일을 다운로드할 수 있습니다.

콘솔

데이터베이스 로그 파일을 다운로드하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 보고자 하는 로그 파일을 보유한 DB 인스턴스의 이름을 선택합니다.
4. 로그 및 이벤트 탭을 선택합니다.
5. 아래로 스크롤하여 [Logs] 섹션을 찾습니다.
6. 로그 섹션에서 다운로드할 로그 옆에 있는 버튼을 선택한 다음 다운로드를 선택합니다.
7. 제공된 링크에 대한 컨텍스트(마우스 오른쪽 클릭) 메뉴를 열고 나서 [Save Link As]를 선택합니다. 로그 파일을 저장할 위치를 입력한 다음 저장을 선택합니다.

AWS CLI

데이터베이스 로그 파일을 다운로드하려면 AWS CLI 명령 `download-db-log-file-portion`을 사용합니다. 기본적으로 이 명령은 로그 파일의 최신 부분만을 다운로드합니다. 하지만 `--starting-token 0` 파라미터를 지정하여 전체 파일을 다운로드할 수 있습니다.

다음 예제에서는 log/ERROR.4라는 로그 파일의 내용을 다운로드하여 errorlog.txt라는 로컬 파일에 저장하는 방법을 보여줍니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

Windows의 경우:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

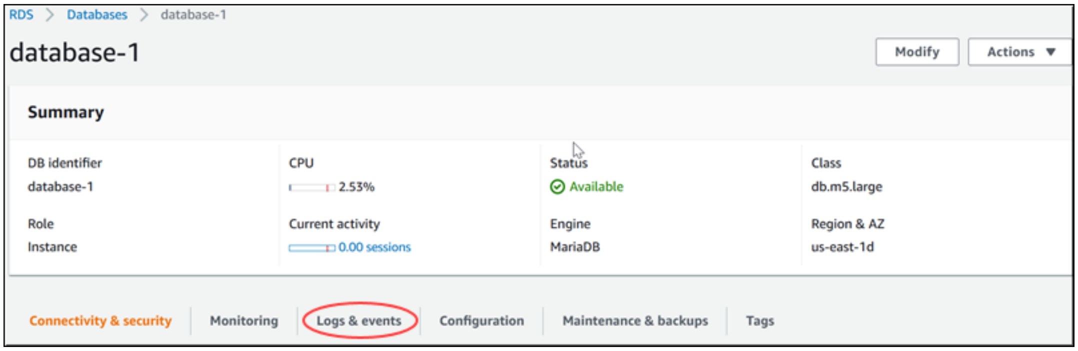
데이터베이스 로그 파일을 다운로드하려면 Amazon RDS API `DownloadDBLogFilePortion` 작업을 사용합니다.

데이터베이스 로그 파일 조사

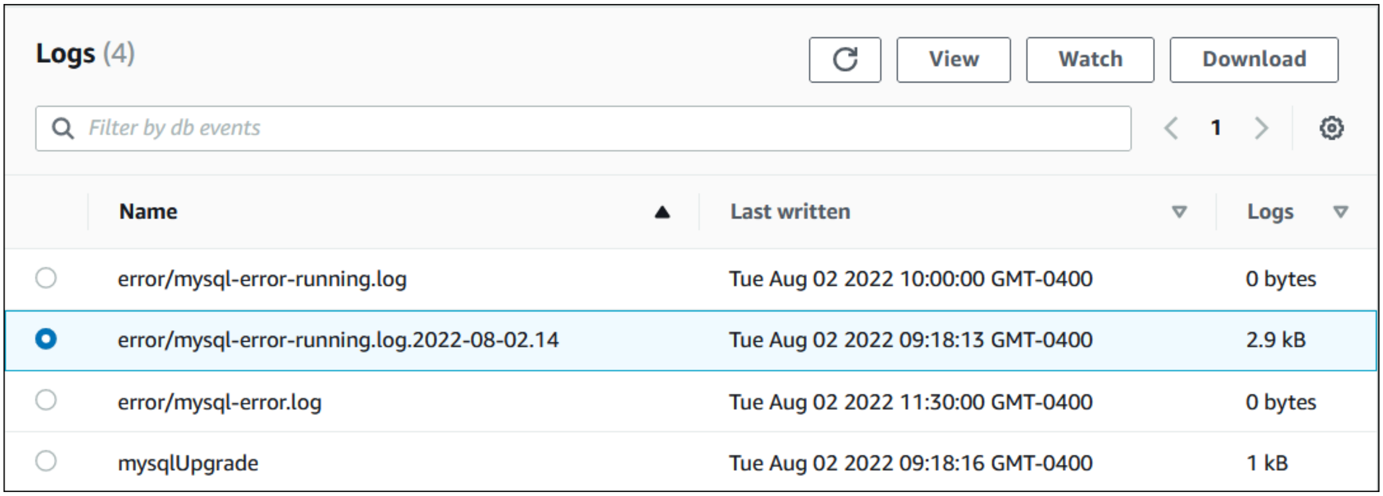
데이터베이스 로그 파일을 관찰하는 것은 UNIX 또는 Linux 시스템에서 파일을 추적하는 것과 같습니다. AWS Management Console을 사용하여 로그 파일을 볼 수 있습니다. RDS는 5초마다 로그 테일을 새로 고칩니다.

데이터베이스 로그 파일을 조사하려면

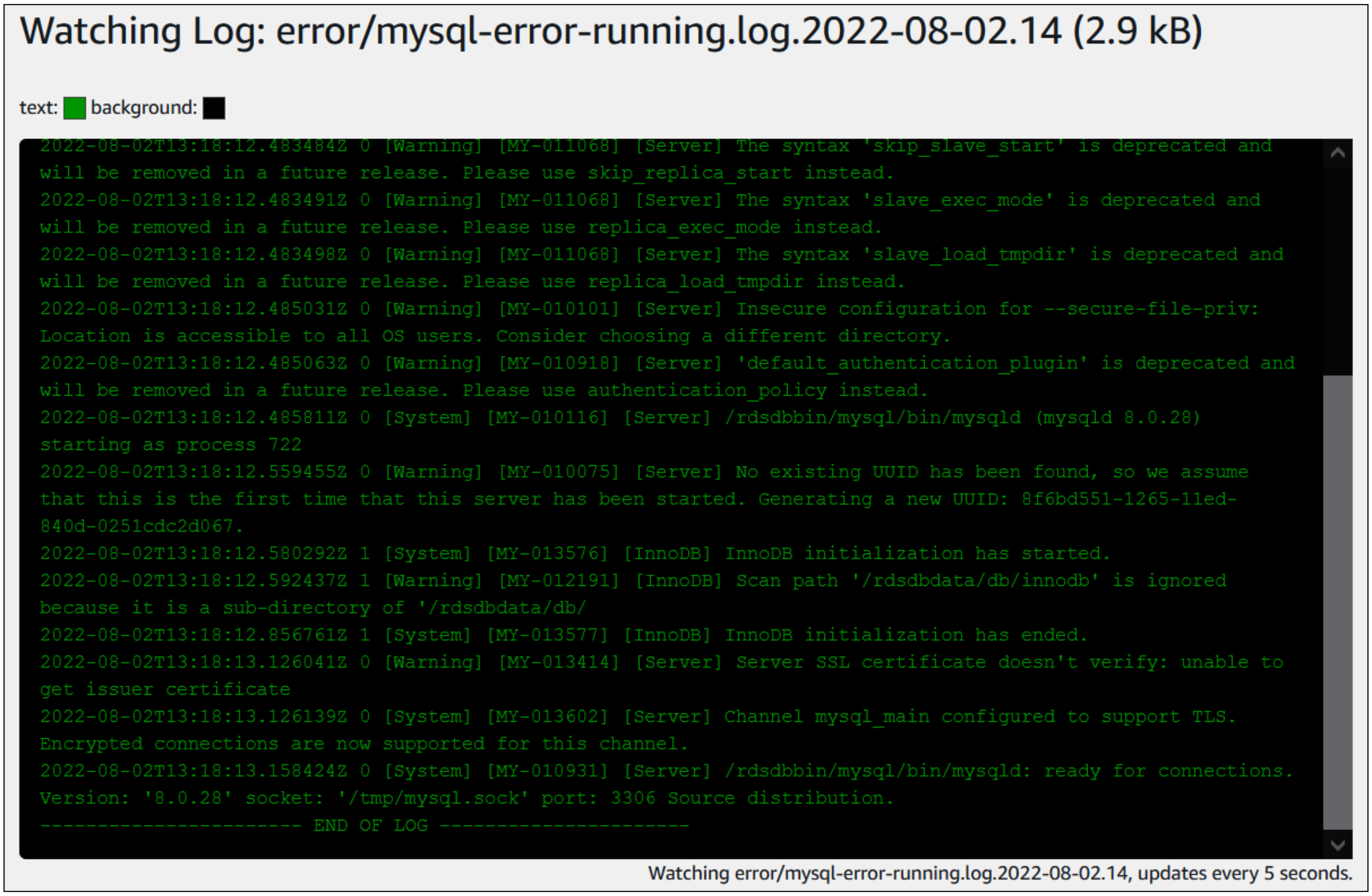
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 보고자 하는 로그 파일을 보유한 DB 인스턴스의 이름을 선택합니다.
4. 로그 및 이벤트 탭을 선택합니다.



5. 로그 섹션에서 로그 파일을 선택한 다음 보기를 선택합니다.



RDS는 다음 MySQL 예제와 같이 로그의 끝부분을 보여줍니다.



Amazon CloudWatch Logs에 데이터베이스 로그 게시

온프레미스 데이터베이스에서 데이터베이스 로그는 파일 시스템에 있습니다. Amazon RDS는 DB 클러스터의 파일 시스템에 있는 데이터베이스 로그에 대한 호스트 액세스를 제공하지 않습니다. 이러한 이유로 Amazon RDS를 사용하면 데이터베이스 로그를 Amazon CloudWatch Logs로 내보낼 수 있습니다. CloudWatch Logs를 통해 로그 데이터에 대한 실시간 분석을 수행할 수 있습니다. 또한 내구성이 뛰어난 스토리지에 데이터를 저장하고, CloudWatch Logs Agent로 데이터를 관리할 수 있습니다.

주제

- RDS와 CloudWatch Logs에 대한 통합 개요
- CloudWatch Logs에 게시할 로그 결정
- CloudWatch Logs에 게시할 로그 지정
- CloudWatch Logs에서 로그 검색 및 필터링

RDS와 CloudWatch Logs에 대한 통합 개요

CloudWatch Logs에서 로그 스트림은 동일한 소스를 공유하는 일련의 로그 이벤트입니다. CloudWatch Logs에서 각 별도의 로그 소스가 별도의 로그 스트림을 구성합니다. 로그 그룹은 동일한 보존 기간, 모니터링 및 액세스 제어 설정을 공유하는 로그 스트림 그룹입니다.

Amazon Aurora는 DB 클러스터 로그 레코드를 로그 그룹으로 지속적으로 스트리밍합니다. 예를 들어 게시하는 각 유형의 로그에 대한 로그 그룹 `/aws/rds/cluster/cluster_name/log_type`이 있습니다. 이 로그 그룹은 로그를 생성하는 데이터베이스 인스턴스와 동일한 AWS 리전에 있습니다.

보존 기간을 지정하지 않는 한 AWS는 CloudWatch Logs에 게시된 로그 데이터를 무기한 보존합니다. 자세한 내용은 CloudWatch Logs에서 로그 데이터 보존 기간 변경을 참조하세요.

CloudWatch Logs에 게시할 로그 결정

각 RDS 데이터베이스 엔진은 자체 로그 세트를 지원합니다. 데이터베이스 엔진 옵션에 대해 알아보려면 다음 주제를 검토하십시오.

- the section called “CloudWatch Logs에 Aurora MySQL 로그 게시”
- the section called “CloudWatch Logs에 Aurora PostgreSQL 로그 게시”

CloudWatch Logs에 게시할 로그 지정

콘솔에 게시할 로그를 지정합니다. AWS Identity and Access Management(IAM)에 서비스 연결 역할이 있는지 확인합니다. 서비스 연결 역할에 대한 자세한 내용은 Amazon Aurora에 서비스 연결 역할 사용를 참조하세요.

게시할 로그 지정

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 다음 중 하나를 수행하세요.
   - 데이터베이스 생성을 선택합니다.
   - 목록에서 데이터베이스를 선택한 다음 수정을 선택합니다.
4. 로그 내보내기에서 게시할 로그를 선택합니다.

   다음 예시에서는 Aurora MySQL DB 클러스터에 대한 감사 로그, 오류 로그, 일반 로그, 인스턴스 로그, IAM 데이터베이스 인증 오류 로그 및 느린 쿼리 로그를 지정합니다.

CloudWatch Logs에서 로그 검색 및 필터링

CloudWatch Logs 콘솔을 이용하여 지정된 기준을 충족하는 로그 항목을 검색할 수 있습니다. CloudWatch Logs 콘솔로 연결되는 RDS 콘솔이나 CloudWatch Logs 콘솔에서 직접 로그에 액세스할 수 있습니다.

콘솔을 이용하여 로그 항목 검색

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. DB 클러스터 또는DB 인스턴스를 고르십시오.
4. 구성을 선택합니다.
5. 게시된 로그에서 보려는 데이터베이스 로그를 선택합니다.

CloudWatch Logs 콘솔을 사용하여 플로우 로그 레코드 검색

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 로그 그룹을 선택합니다.
3. 필터 상자에 **/aws/rds**를 입력합니다.
4. 로그 그룹에서 검색할 로그 스트림이 포함된 로그 그룹의 이름을 선택합니다.
5. 로그 스트림에서 검색할 로그 스트림의 이름을 선택합니다.
6. 로그 이벤트에서 사용할 필터 구문을 입력합니다.

자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 로그 데이터 검색 및 필터링을 참조하십시오. RDS 로그를 모니터링하는 방법을 설명하는 블로그의 자습서는 Amazon CloudWatch Logs, AWS Lambda 및 Amazon SNS를 사용하여 Amazon RDS에 대한 사전 예방적 데이터베이스 모니터링 구축을 참조하십시오.

REST를 사용하여 로그 파일 내용 읽기

Amazon RDS는 DB 인스턴스 로그 파일 액세스를 허용하는 REST 엔드포인트를 제공합니다. Amazon RDS 로그 파일 내용을 스트리밍하는 애플리케이션을 작성해야 하는 경우 유용합니다.

구문은 다음과 같습니다.

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
```

```
Content-type: application/json
host: rds.region.amazonaws.com
```

다음 파라미터는 필수 파라미터입니다.

- *DBInstanceIdentifier* — 다운로드하려는 로그 파일이 있는 DB 인스턴스에 고객이 할당하는 이름입니다.
- *LogFileName* — 다운로드할 로그 파일의 이름입니다.

응답에는 스트림으로 요청된 로그 파일의 내용이 포함됩니다.

다음 예제에서는 us-west-2 리전에 sample-sql로 명명된 DB 인스턴스에 대해 log/ERROR.6으로 명명된 로그 파일을 다운로드합니다.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

존재하지 않는 DB 인스턴스를 지정하는 경우 응답에 다음 오류가 포함됩니다.

- DBInstanceNotFound — *DBInstanceIdentifier*는 기존 DB 인스턴스를 참조하지 않습니다. (HTTP 상태 코드: 404)

Aurora MySQL 데이터베이스 로그 파일

Amazon RDS 콘솔, Amazon RDS API, AWS CLI 또는 AWS SDK를 통해 Aurora MySQL 로그를 직접 모니터링할 수 있습니다. 또한, 주 데이터베이스에 있는 데이터베이스 테이블로 로그를 전송하고 그 테이블을 쿼리하여 MySQL 로그에 액세스할 수 있습니다. mysqlbinlog 유틸리티를 사용하여 이진 로그를 다운로드할 수 있습니다.

파일 기반 데이터베이스 로그 보기, 다운로드 및 조사 방법에 대한 자세한 내용은 Amazon Aurora 로그 파일 모니터링 단원을 참조하십시오.

주제

- Aurora MySQL 데이터베이스 로그 개요
- Aurora MySQL 로그 출력을 표로 보내기
- 단일 AZ 데이터베이스의 Aurora MySQL 이진 로깅 구성
- MySQL 이진 로그 액세스

Aurora MySQL 데이터베이스 로그 개요

다음 유형의 Aurora MySQL 로그 파일을 모니터링할 수 있습니다.

- 오류 로그
- 느린 쿼리 로그
- 일반 로그
- 감사 로그
- 인스턴스 로그
- IAM 데이터베이스 인증 오류 로그

Aurora MySQL 오류 로그는 기본적으로 생성됩니다. DB 파라미터 그룹에서 파라미터를 설정하여 느린 쿼리 및 일반 로그를 생성할 수 있습니다.

주제

- Aurora MySQL 오류 로그
- Aurora MySQL 느린 쿼리 로그 및 일반 로그
- Aurora MySQL 감사 로그
- Aurora MySQL 인스턴스 로그

- Aurora MySQL용 로그 교체 및 보존
- Amazon CloudWatch Logs에 Aurora MySQL 로그 게시

Aurora MySQL 오류 로그

Aurora MySQL은 `mysql-error.log` 파일에 오류를 기록합니다. 각 로그 파일이 생성된 시간(UTC)이 파일 이름에 추가됩니다. 로그 파일에는 타임스탬프도 포함되어 있어, 로그 항목이 작성된 시간을 확인하는 데 도움이 됩니다.

Aurora MySQL에서는 시작, 종료 및 오류 발생 시에만 오류 로그에 데이터가 로그됩니다. DB 인스턴스는 오류 로그에 새 항목이 기록되지 않는 상태로 몇 시간이나 며칠씩 작동할 수 있습니다. 최근 항목이 보이지 않으면 이는 서버에서 로그에 입력될 만한 오류가 발생하지 않았기 때문입니다.

기본적으로 오류 로그는 오류와 같은 예기치 않은 이벤트만 표시되도록 필터링됩니다. 그러나 오류 로그에는 쿼리 진행률과 같이 표시되지 않는 몇 가지 추가 데이터베이스 정보도 포함되어 있습니다. 따라서 실제 오류가 없더라도 진행 중인 데이터베이스 작업으로 인해 오류 로그의 크기가 커질 수 있습니다. 그리고 AWS Management Console에서 오류 로그의 특정 크기(바이트 또는 킬로바이트)를 볼 수 있지만 다운로드할 때 0바이트일 수 있습니다.

Aurora MySQL은 5분마다 `mysql-error.log`를 디스크에 씁니다. 또한, 로그의 콘텐츠를 `mysql-error-running.log`에 추가합니다.

Aurora MySQL은 `mysql-error-running.log` 파일을 1시간마다 교체합니다.

> **Note**
> 로그 보존 기간은 Amazon RDS와 Aurora 간에 다릅니다.

Aurora MySQL 느린 쿼리 로그 및 일반 로그

Aurora MySQL 느린 쿼리 로그와 일반 로그를 파일이나 데이터베이스 테이블에 기록할 수 있습니다. 그러려면 DB 파라미터 그룹의 파라미터를 설정합니다. DB 파라미터 그룹의 생성 및 변경에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오. 이런 파라미터를 설정해야 Amazon RDS 콘솔에서 느린 쿼리 로그 또는 일반 로그를 볼 수 있습니다. 아니면 Amazon RDS API, Amazon RDS CLI 또는 AWS SDK를 사용하여 볼 수 있습니다.

이 목록에 있는 파라미터를 사용하여 Aurora MySQL 로깅을 제어할 수 있습니다.

- `slow_query_log`: 느린 쿼리 로그를 만들려면 1로 설정합니다. 기본값은 0입니다.

- general_log: 일반 로그를 만들려면 1로 설정합니다. 기본값은 0입니다.
- long_query_time: 빠르게 실행되는 쿼리가 느린 쿼리 로그에 기록되지 않도록 하려면 로깅할 쿼리의 최단 런타임 값(초)을 지정합니다. 기본값은 10초이고, 최소값은 0초입니다. log_output = FILE인 경우에는 마이크로초 단위까지 부동 소수점 값을 지정할 수 있습니다. log_output = TABLE인 경우에는 초 단위로 정수 값을 지정해야 합니다. 런타임이 long_query_time 값을 초과하는 쿼리만 로깅됩니다. 예를 들어 long_query_time을 0.1로 설정하면 100밀리초 미만의 시간 동안 작동하는 쿼리가 로그에 기록되지 않습니다.
- log_queries_not_using_indexes: 인덱스를 사용하지 않는 모든 쿼리를 느린 쿼리 로그에 기록하려면 1로 설정합니다. 인덱스를 사용하지 않는 쿼리는 런타임이 long_query_time 파라미터의 값보다 짧아도 로깅됩니다. 기본값은 0입니다.
- log_output *option*: log_output 파라미터에 대해 다음 옵션 중 하나를 지정할 수 있습니다.
  - TABLE – 일반 쿼리를 mysql.general_log테이블에 쓰고 느린 쿼리를 mysql.slow_log 테이블에 씁니다.
  - FILE – 일반 쿼리 로그와 느린 쿼리 로그를 모두 파일 시스템에 씁니다.
  - NONE – 로깅을 비활성화합니다.

  Aurora MySQL 버전 2 및 3의 경우 log_output의 기본값은 FILE입니다.

느린 쿼리 데이터가 Amazon CloudWatch Logs에 표시되려면 다음 조건을 충족해야 합니다.

- 느린 쿼리 로그를 포함하도록 CloudWatch Logs를 구성해야 합니다.
- slow_query_log를 활성화해야 합니다.
- log_output를 FILE로 설정해야 합니다.
- 쿼리는 long_query_time에 구성된 시간보다 오래 걸릴 수 있습니다.

느린 쿼리 및 일반 로그에 대한 자세한 내용은 MySQL 문서에서 다음 항목을 참조하십시오.

- [느린 쿼리 로그]
- [일반 쿼리 로그]

Aurora MySQL 감사 로그

Aurora MySQL에 대한 감사 로깅을 고급 감사라고 합니다. 고급 감사를 켜려면 특정 DB 클러스터 파라미터를 설정합니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 섹션을 참조하세요.

Aurora MySQL 인스턴스 로그

Aurora는 자동 일시 중지가 사용 설정된 DB 인스턴스에 대해 별도의 로그 파일을 만듭니다. 이 instance.log 파일은 예상 시 이러한 DB 인스턴스를 일시 중지할 수 없는 이유를 기록합니다. 인스턴스 로그 파일 동작 및 Aurora 자동 일시 중지 기능에 대한 자세한 내용은 Aurora Serverless v2 일시 중지 및 재개 활동 모니터링을 참조하시기 바랍니다.

Aurora MySQL용 로그 교체 및 보존

로깅을 사용하는 경우 Amazon Aurora는 로그 파일을 정기적으로 교체하거나 삭제합니다. 이러한 예방 조치를 취하면 데이터베이스 사용에 방해가 되거나 성능에 영향을 미치는 큰 로그 파일이 생성될 가능성을 줄일 수 있습니다. Aurora MySQL은 교체 및 삭제를 다음과 같이 처리합니다.

- Aurora MySQL 오류 로그 파일 크기는 DB 인스턴스 로컬 스토리지의 15% 이하로 제한됩니다. 이 임계값을 유지하기 위해 매시간 로그가 자동으로 교체됩니다. Aurora MySQL은 30일 후 또는 디스크 공간의 15%가 차면 로그를 제거합니다. 오래된 로그 파일을 제거한 후 로그 파일의 총 크기가 임계값을 초과하는 경우에는 로그 파일의 전체 크기가 임계값 이하로 작아질 때까지 가장 오래된 로그 파일부터 차례대로 삭제됩니다.
- Aurora MySQL는 24시간 후 또는 스토리지의 15%가 사용된 후 감사, 일반 및 느린 쿼리 로그를 제거합니다.
- `FILE` 로깅을 사용하는 경우 일반 로그 및 느린 쿼리 로그 파일은 매시간 검사되며 24시간 이상 지난 로그 파일은 삭제됩니다. 삭제 후 남은 결합 로그 파일 크기가 DB 인스턴스에 할당된 로컬 공간 중 15%의 임계값을 초과하는 경우도 있습니다. 이 경우 로그 파일의 전체 크기가 임계값 이하로 작아질 때까지 가장 오래된 로그 파일부터 차례대로 삭제됩니다.
- `TABLE` 로깅을 사용하는 경우 로그 테이블이 교체되거나 삭제되지 않습니다. 결합된 모든 로그의 크기가 너무 크면 로그 테이블이 잘립니다. 공간 확보를 위해 로그 테이블을 수동으로 교체하거나 삭제해야 할 때 알림을 받기 위해 `low storage` 이벤트 범주를 구독할 수 있습니다. 자세한 내용은 Amazon RDS 이벤트 알림 작업 섹션을 참조하세요.

  `mysql.rds_rotate_general_log` 절차를 호출하면 수동으로 `mysql.general_log` 테이블을 교체할 수 있습니다. `mysql.slow_log` 절차를 호출하면 `mysql.rds_rotate_slow_log` 테이블을 순환할 수 있습니다.

  로그 테이블을 수동으로 교체하면 현재 로그 테이블이 백업 로그 테이블에 복사되고 현재 로그 테이블의 항목이 제거됩니다. 백업 로그 테이블이 이미 존재할 경우, 현재 로그 테이블이 백업으로 복사되기 전에 백업 로그 테이블이 삭제됩니다. 필요하다면 백업 로그 테이블을 쿼리할 수 있습니다. `mysql.general_log` 테이블에 대한 백업 로그 테이블 이름은

mysql.general_log_backup으로 지정됩니다. mysql.slow_log 테이블에 대한 백업 로그 테이블 이름은 mysql.slow_log_backup으로 지정됩니다.

- Aurora MySQL 감사 로그는 파일 크기가 100MB에 도달하면 교체되고 24시간 후에 제거됩니다.
- Amazon RDS는 10MB보다 큰 IAM 데이터베이스 인증 오류 로그 파일을 교체합니다. Amazon RDS는 5일 이상 또는 100MB 이상의 IAM 데이터베이스 인증 오류 로그 파일을 제거합니다.

Amazon RDS 콘솔, Amazon RDS API, Amazon RDS CLI 또는 AWS SDK에서 로그를 사용하여 작업하려면 log_output 파라미터를 FILE로 설정합니다. Aurora MySQL 오류 로그와 마찬가지로, 이러한 로그 파일은 매시간 교체됩니다. 이전의 24시간 동안 생성된 로그 파일이 보존됩니다. 보존 기간은 Amazon RDS와 Aurora 간에 다릅니다.

Amazon CloudWatch Logs에 Aurora MySQL 로그 게시

Amazon CloudWatch Logs의 로그 그룹에 로그 데이터를 게시하도록 Aurora MySQL DB 클러스터를 구성할 수 있습니다. CloudWatch Logs를 통해 로그 데이터에 대한 실시간 분석을 수행할 수 있고, CloudWatch를 사용하여 경보를 만들고 지표를 볼 수 있습니다. CloudWatch Logs를 사용하여 내구성이 뛰어난 스토리지에 로그 레코드를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 섹션을 참조하세요.

Aurora MySQL 로그 출력을 표로 보내기

DB 파라미터 그룹을 만들고 log_output 서버 파라미터를 TABLE로 설정하여 일반 및 느린 쿼리 로그를 DB 인스턴스 상의 테이블로 전송할 수 있습니다. 그러면 일반 쿼리는 mysql.general_log 테이블에, 느린 쿼리는 mysql.slow_log 테이블에 로그가 기록됩니다. 이들 테이블을 쿼리하여 로그 정보에 액세스할 수 있습니다. 이 로깅을 활성화하면 데이터베이스에 기록되는 데이터의 양이 증가하여 성능이 저하될 수 있습니다.

일반 로그와 느린 쿼리 로그는 모두 기본적으로 비활성화됩니다. 테이블에 대한 로깅을 활성화하려면 general_log 및 slow_query_log 서버 파라미터도 1로 설정해야 합니다.

관련 파라미터를 0으로 설정하여 각 로깅 활동을 해제할 때까지 로그 테이블은 계속 커집니다. 흔히 대량의 데이터가 시간 경과에 따라 누적되어 할당된 스토리지 공간 중 상당한 비율을 사용할 수 있습니다. Amazon Aurora에서는 로그 테이블을 자를 수 없지만 테이블의 콘텐츠를 이동할 수 있습니다. 테이블을 순환하면 그 내용이 백업 테이블에 저장된 다음 빈 로그 테이블이 새로 생성됩니다. 다음 명령줄 프로시저로 로그 테이블을 수동으로 순환시킬 수 있으며, 이때 명령 프롬프트는 PROMPT>로 표시됩니다.

```
PROMPT> CALL mysql.rds_rotate_slow_log;
```

```
PROMPT> CALL mysql.rds_rotate_general_log;
```

오래된 데이터를 완전히 제거하고 디스크 공간을 회수하려면 알맞은 프로시저를 두 번 연속으로 호출합니다.

단일 AZ 데이터베이스의 Aurora MySQL 이진 로깅 구성

이진 로그는 Aurora MySQL 서버 인스턴스의 데이터 수정에 대한 정보를 포함하는 로그 파일 세트입니다. 이진 로그에는 다음과 같은 정보가 포함되어 있습니다.

- 테이블 생성 또는 행 수정과 같은 데이터베이스 변경 사항을 설명하는 이벤트
- 데이터를 업데이트한 각 문의 기간에 대한 정보
- 데이터를 업데이트할 수 있었지만 업데이트하지 않은 문에 대한 이벤트

이진 로그는 복제 중 전송된 문을 기록합니다. 일부 복구 작업에도 필요합니다. 자세한 내용은 MySQL 설명서의 [바이너리 로그](#) 단원을 참조하십시오.

바이너리 로그는 복제본이 아닌 프라이머리 DB 인스턴스에서만 액세스할 수 있습니다.

Amazon Aurora의 MySQL은 행 기반, 문 기반, 혼합 바이너리 로깅 형식을 지원합니다. 특정한 binlog 형식이 필요한 경우가 아니면 혼합하는 것이 좋습니다. 다양한 Aurora MySQL 이진 로그 형식에 대한 자세한 내용은 MySQL 설명서의 Binary Logging Formats을 참조하세요.

복제를 사용하려는 경우 바이너리 로깅 형식은 원본에 기록되고 복제 대상으로 전송되는 데이터 변경 내용의 레코드를 확인하므로 중요합니다. 복제와 관련된 다양한 바이너리 로깅 형식의 장/단점에 대한 자세한 내용은 MySQL 설명서의 Advantages and Disadvantages of Statement-Based and Row-Based Replication을 참조하십시오.

> **Important**
>
> MySQL 8.0.34에서 MySQL이 `binlog_format` 파라미터를 더 이상 사용하지 않습니다. 이후 MySQL 버전에서 MySQL은 파라미터를 제거하고 행 기반 복제만 지원할 계획입니다. 따라서 새 MySQL 복제 설정에 행 기반 로깅을 사용하는 것이 좋습니다. 자세한 내용은 MySQL 설명서의 binlog_format을 참조하세요.
> MySQL 버전 8.0 및 8.4는 `binlog_format` 파라미터를 수락합니다. 이 파라미터를 사용하면 MySQL에서 사용 중단 경고가 표시됩니다. 향후 메이저 릴리스에서 MySQL로부터 `binlog_format` 파라미터가 제거될 예정입니다.

명령문 기반 복제를 수행하면 원본 DB 클러스터와 읽기 전용 복제본 간에 불일치가 발생할 수 있습니다. 자세한 내용은 MySQL 설명서의 Determination of Safe and Unsafe Statements in Binary Logging을 참조하십시오.
이진 로깅을 활성화하면 DB 클러스터에 대한 평균 디스크 쓰기 I/O 작업 수가 증가합니다. `VolumeWriteIOPs` CloudWatch 지표를 사용하여 IOPS 사용량을 모니터링할 수 있습니다.

MySQL 이진 로깅 형식을 설정하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 수정할 DB 클러스터와 연결된 DB 클러스터 파라미터 그룹을 선택합니다.

   기본 파라미터 그룹을 수정할 수 없습니다. DB 클러스터가 기본 파라미터 그룹을 사용하고 있는 경우 새 파라미터 그룹을 생성하여 DB 클러스터와 연결합니다.

   파라미터 그룹에 대한 자세한 정보는 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.
4. 작업에서 편집을 선택합니다.
5. `binlog_format` 파라미터를 선택한 이진 로깅 형식(`ROW`, `STATEMENT`, 또는 `MIXED`)으로 설정합니다. `OFF` 값을 사용하여 이진 로깅을 해제할 수도 있습니다.

   Note

   DB 클러스터 파라미터 그룹에서 `binlog_format`을 `OFF`로 설정하면 `log_bin` 세션 변수가 비활성화됩니다. 그러면 Aurora MySQL DB 클러스터에 대한 이진 로깅이 비활성화되며, 이는 다시 `binlog_format` 세션 변수를 데이터베이스의 기본값인 `ROW` 값으로 재설정합니다.
6. 변경 내용 저장을 선택하여 업데이트를 DB 클러스터 파라미터 그룹에 저장합니다.

이 단계를 수행한 후 변경 사항을 적용하기 위해 DB 클러스터에서 라이터 인스턴스를 재부팅해야 합니다. Aurora MySQL 버전 2.09 이하에서는 라이터 인스턴스를 재부팅하면 DB 클러스터의 모든 리더 인스턴스도 재부팅됩니다. Aurora MySQL 버전 2.10 이상에서는 모든 리더 인스턴스를 수동으로 재부팅해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터 또는 Amazon Aurora DB 인스턴스 재부팅 섹션을 참조하세요.

> ⚠ Important
>
> DB 클러스터 파라미터 그룹을 변경하면 해당 파라미터 그룹을 사용하는 모든 DB 클러스터에 영향을 미칩니다. AWS 리전의 다양한 Aurora MySQL DB 클러스터에 대해 서로 다른 이진 로깅 형식을 지정하려면 DB 클러스터에서 서로 다른 DB 클러스터 파라미터 그룹을 사용해야 합니다. 이러한 파라미터 그룹은 다양한 로깅 형식을 식별합니다. 각 DB 클러스터에 적절한 DB 클러스터 파라미터 그룹을 할당합니다. Aurora MySQL 파라미터에 대한 자세한 내용은 Aurora MySQL 구성 파라미터 단원을 참조하십시오.

MySQL 이진 로그 액세스

mysqlbinlog 유틸리티를 사용하여 RDS for MySQL DB 인스턴스에서 이진 로그를 다운로드하거나 스트리밍할 수 있습니다. 이진 로그를 로컬 컴퓨터로 다운로드하면 mysql 유틸리티를 사용하여 로그를 재생하는 것과 같은 작업을 수행할 수 있습니다. mysqlbinlog 유틸리티 사용에 대한 자세한 내용은 MySQL 설명서의 mysqlbinlog를 사용하여 이진 로그 파일 백업을 참조하세요.

Amazon RDS 인스턴스에 대해 mysqlbinlog 유틸리티를 실행하려면 다음 옵션을 사용합니다.

- `--read-from-remote-server` - 필수입니다.
- `--host` – 인스턴스의 엔드포인트에서 DNS 이름.
- `--port` – 인스턴스에서 사용되는 포트.
- `--user` - `REPLICATION SLAVE` 권한이 부여된 MySQL 사용자.
- `--password` – MySQL 사용자의 암호. 또는 유틸리티에서 암호 입력을 요구하는 메시지가 표시되도록 암호 값을 생략.
- `--raw` - 파일을 이진 형식으로 다운로드합니다.
- `--result-file` - 원시 출력을 수신할 로컬 파일.
- `--stop-never` - 이진 로그 파일을 스트리밍합니다.
- `--verbose` - ROW binlog 형식을 사용할 경우 이 옵션을 포함하면 행 이벤트를 유사 SQL 문으로 볼 수 있습니다. `--verbose` 옵션에 대한 자세한 내용은 MySQL 설명서의 mysqlbinlog 행 이벤트 표시를 참조하세요.
- 하나 이상의 이진 로그 파일의 이름을 지정합니다. 사용 가능한 로그의 목록을 획득하려면 SQL 명령 `SHOW BINARY LOGS`를 사용합니다.

mysqlbinlog 옵션에 대한 자세한 내용은 MySQL 설명서의 mysqlbinlog - 이진 로그 파일 처리용 유틸리티를 참조하세요.

다음 예시에서는 mysqlbinlog 유틸리티를 사용하는 방법을 보여줍니다.

대상 LinuxmacOS, 또는Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

Windows의 경우:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

mysqlbinlog 유틸리티가 바이너리 로그에 액세스할 수 있도록 DB 인스턴스에서 바이너리 로그를 계속 사용할 수 있어야 합니다. 가용성을 보장하기 위해 mysql.rds_set_configuration 저장 프로시저를 사용하고 로그를 다운로드할 수 있는 충분한 시간을 지정합니다. 이 구성이 설정되지 않은 경우 Amazon RDS가 이진 로그를 가능한 한 빨리 제거하므로 mysqlbinlog 유틸리티에서 검색하는 바이너리 로그에 간극이 발생합니다.

다음 예제에서는 보존 기간을 1일로 설정합니다.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

현재 설정을 표시하려면 mysql.rds_show_configuration 저장 프로시저를 사용합니다.

```
call mysql.rds_show_configuration;
```

Aurora PostgreSQL 데이터베이스 로그 파일

다음 유형의 Aurora PostgreSQL 로그 파일을 모니터링할 수 있습니다.

- PostgreSQL 로그
- 인스턴스 로그
- IAM 데이터베이스 인증 오류 로그

> **Note**
>
> IAM 데이터베이스 인증 오류 로그를 활성화하려면 먼저 Aurora PostgreSQL DB 클러스터에 IAM 데이터베이스 인증을 활성화해야 합니다. IAM 데이터베이스 인증 활성화에 대한 자세한 내용은 IAM 데이터베이스 인증의 활성화 및 비활성화 섹션을 참조하세요.

Aurora PostgreSQL은 기본 PostgreSQL 로그 파일에 데이터베이스 활동을 로깅합니다. 온프레미스 PostgreSQL DB 인스턴스의 경우 이러한 메시지가 `log/postgresql.log`에 로컬로 저장됩니다. Aurora PostgreSQL DB 클러스터 의 경우 Aurora 클러스터에서 로그 파일을 사용할 수 있습니다. 또한 이러한 로그는 AWS Management Console을 통해 액세스할 수 있으며, 여기에서 해당 로그를 보거나 다운로드할 수 있습니다. 기본 로깅 수준에서는 로그인 실패, 치명적인 서버 오류, 교착 상태 및 쿼리 실패를 캡처합니다.

파일 기반 데이터베이스 로그 보기, 다운로드 및 조사 방법에 대한 자세한 내용은 Amazon Aurora 로그 파일 모니터링 섹션을 참조하세요. PostgreSQL 로그에 대한 자세한 내용은 Working with Amazon RDS and Aurora PostgreSQL logs: Part 1(RDS 및 Aurora PostgreSQL 로그로 작업하기: 파트 1) 및 Working with Amazon RDS and Aurora PostgreSQL logs: Part 2(RDS 및 Aurora PostgreSQL 로그로 작업하기: 파트 2)를 참조하세요.

이 주제에서 설명하는 표준 PostgreSQL 로그 외에 Aurora PostgreSQL은 PostgreSQL Auidt 확장(`pgAudit`)도 지원합니다. 대부분의 규제 대상 산업 및 정부 기관은 법적 요구 사항을 준수하기 위해 데이터 변경에 대한 감사 로그 또는 감사 추적을 보존해야 합니다. pgAudit 설치 및 사용에 대한 자세한 내용은 pgAudit를 사용하여 데이터베이스 활동 로깅 섹션을 참조하세요.

Aurora는 자동 일시 중지가 사용 설정된 DB 인스턴스에 대해 별도의 로그 파일을 만듭니다. 이 instance.log 파일은 예상 시 이러한 DB 인스턴스를 일시 중지할 수 없는 이유를 기록합니다. 인스턴스 로그 파일 동작 및 Aurora 자동 일시 중지 기능에 대한 자세한 내용은 Aurora Serverless v2 일시 중지 및 재개 활동 모니터링을 참조하시기 바랍니다.

주제

- Aurora PostgreSQL에서의 로깅을 위한 파라미터
- Aurora PostgreSQL DB 클러스터에 쿼리 로깅을 활성화합니다.

Aurora PostgreSQL에서의 로깅을 위한 파라미터

다양한 파라미터를 수정하여 Aurora PostgreSQL DB 클러스터의 로깅 동작을 사용자 지정할 수 있습니다. 다음 표에서는 로그 저장 기간, 로그 교체 시기, 로그를 CSV(쉼표로 구분된 값) 형식으로 출력할지 여부 등 다양한 파라미터를 확인할 수 있습니다. 다른 설정에서 STDERR로 전송된 텍스트 출력도 찾을 수 있습니다. 수정 가능한 파라미터의 설정을 변경하려면 Aurora PostgreSQL DB 클러스터에 대한 사용자 지정 DB 클러스터 파라미터 그룹을 사용하세요. 자세한 내용은 Amazon Aurora의 파라미터 그룹. 섹션을 참조하세요.

파라미터	Default	설명
log_destination	stderr	로그에 대한 출력 형식을 설정합니다. 기본값은 `stderr`이지만 설정에 `csvlog`를 추가하여 CSV(쉼표로 구분된 값)를 지정할 수도 있습니다. 자세한 내용은 로그 대상 설정(`stderr`, `csvlog`) 섹션을 참조하세요.
log_filename	postgresql.log.%Y-%m-%d-%H%M	로그 파일 이름의 패턴을 지정합니다. 이 파라미터는 기본값 외에도 `postgresql.log.%Y-%m-%d` 및 `postgresql.log.%Y-%m-%d-%H` 파일 이름 패턴을 지원합니다. Aurora PostgreSQL 버전 17.4 이상에서는 이 파라미터를 수정할 수 없습니다.
log_line_prefix	%t:%r:%u@%d:[%p]:	시간(%t), 원격 호스트(%r), 사용자(%u), 데이터베이스(%d) 및 프로세스 ID(%p)를 기록하기 위해 `stderr`에 기록되는 각 로그 줄의 접두사를 정의합니다.
log_rotation_age	60	몇 분 후에 로그 파일이 자동으로 교체됩니다. 이 값을 1분에서 1,440분 사이의 값으로 변경할 수 있습니다. 자세한 내용은 로그 파일 회전 설정 섹션을 참조하세요.

파라미터	Default	설명
log_rotation_size	–	로그 교체가 자동으로 이루어질 때의 크기(kB)입니다. 50,000~1,000,000KB 범위 내에서 이 값을 변경할 수 있습니다. 자세한 내용은 로그 파일 회전 설정를 참조하세요.
rds.log_retention_period	4320	지정된 시간(분)보다 오래된 PostgreSQL 로그가 삭제됩니다. 기본값인 4,320분으로 지정하면 3일이 지난 로그 파일이 삭제됩니다. 자세한 내용은 로그 보존 기간 설정 섹션을 참조하세요.

애플리케이션 문제를 식별하기 위해 로그에서 쿼리 실패, 로그인 실패, 교착 상태 및 치명적인 서버 오류를 찾아볼 수 있습니다. 예를 들어, 레거시 애플리케이션을 Oracle에서 Aurora PostgreSQL로 변환했지만 일부 쿼리가 올바르게 변환되지 않았다고 가정합시다. 이러한 잘못된 형식의 쿼리는 오류 메시지를 생성하고, 로그에서 오류 메시지를 찾아 문제를 식별할 수 있습니다. 쿼리 로깅에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터에 쿼리 로깅을 활성화합니다. 섹션을 참조하세요.

다음 주제에서 PostgreSQL 로깅의 기본 세부 정보를 제어하는 다양한 파라미터에 관한 정보를 찾을 수 있습니다.

주제

로그 보존 기간 설정

`rds.log_retention_period` 파라미터는 Aurora PostgreSQL DB 클러스터가 로그 파일을 보관하는 기간을 지정합니다. 기본 설정은 3일(4,320분)이지만 1일(1,440분)에서 7일(10,080분) 사이로 이 값을 설정할 수 있습니다. Aurora PostgreSQL DB 클러스터에 일정 기간 동안 로그 파일을 보관할 수 있는 충분한 스토리지가 있는지 확인하세요.

로그를 정기적으로 Amazon CloudWatch Logs에 게시하여 로그가 Aurora PostgreSQL DB 클러스터에서 제거된 후에도 시스템 데이터를 보고 분석할 수 있도록 하는 것이 좋습니다. 자세한 내용은 Amazon

CloudWatch Logs에 Aurora PostgreSQL 로그 게시. CloudWatch 게시를 설정하면 로그가 CloudWatch Logs에 게시될 때까지 Aurora에서 해당 로그를 삭제하지 않습니다.

DB 인스턴스의 스토리지가 임계값에 도달하면 Amazon Aurora가 오래된 PostgreSQL 로그를 압축합니다. Aurora는 gzip 압축 유틸리티를 사용하여 파일을 압축합니다. 자세한 내용은 gzip 웹 사이트를 참조하세요.

DB 인스턴스의 스토리지가 부족하고 사용 가능한 모든 로그가 압축되면 다음과 같은 경고가 표시됩니다.

```
Warning: local storage for PostgreSQL log files is critically low for 
this Aurora PostgreSQL instance, and could lead to a database outage.
```

스토리지가 부족하면 Aurora는 지정된 보존 기간이 끝나기 전에 압축된 PostgreSQL 로그를 삭제할 수 있습니다. 이 경우 다음과 유사한 메시지가 나타납니다.

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

로그 파일 회전 설정

Aurora는 기본적으로 매시간 새 로그 파일을 생성합니다. 타이밍은 `log_rotation_age` 파라미터로 제어됩니다. 이 파라미터의 기본값은 60(분)이지만 1분에서 24시간(1,440분) 사이로 설정할 수 있습니다. 교체 시간이 되면 새 로그 파일이 생성됩니다. 파일의 이름은 `log_filename` 파라미터로 지정된 패턴에 따라 결정됩니다.

로그 파일은 `log_rotation_size` 파라미터에 지정된 대로 크기에 따라 교체될 수도 있습니다. 이 파라미터는 로그가 지정된 크기(KB)에 도달하면 교체되도록 지정합니다. 기본값 `log_rotation_size`는 Aurora PostgreSQL DB 클러스터의 경우 100,000KB(킬로바이트)이지만 이 값을 50,000~1,000,000킬로바이트 사이로 설정할 수 있습니다.

로그 파일 이름은 `log_filename` 파라미터에 지정된 파일 이름 패턴을 기반으로 합니다. 이 파라미터에 사용할 수 있는 설정은 다음과 같습니다.

- `postgresql.log.%Y-%m-%d` - 로그 파일 이름의 기본 형식입니다. 로그 파일 이름에 년, 월, 일이 포함됩니다.
- `postgresql.log.%Y-%m-%d-%H` - 로그 파일 이름 형식에 시간이 포함됩니다.
- `postgresql.log.%Y-%m-%d-%H%M` - 로그 파일 이름 형식에 시간:분이 포함됩니다.

log_rotation_age 파라미터를 60분 미만으로 설정한 경우 log_filename 파라미터도 분 형식으로 설정하세요.

자세한 내용은 PostgreSQL 설명서의 log_rotation_age 및 log_rotation_size 섹션을 참조하세요.

로그 대상 설정(**stderr**, **csvlog**)

기본적으로 Aurora PostgreSQL은 표준 오류(stderr) 형식으로 로그를 생성합니다. 이 형식이 log_destination 파라미터에 대한 기본 설정입니다. 각 메시지에는 log_line_prefix 파라미터에 지정된 패턴을 사용하여 접두사가 붙습니다. 자세한 내용은 log_line_prefix 파라미터 이해하기 섹션을 참조하세요.

Aurora PostgreSQL도 로그를 csvlog 형식으로 생성할 수 있습니다. csvlog는 로그 데이터를 쉼표로 구분된 값(CSV) 데이터로 분석하는 데 유용합니다. 예를 들어 외부 테이블로 로그를 사용하기 위해 log_fdw 확장을 사용한다고 가정하겠습니다. stderr 로그 파일에 만들어진 외부 테이블에는 로그 이벤트 데이터가 있는 하나의 열이 포함되어 있습니다. log_destination 파라미터에 csvlog를 추가하면 외부 테이블의 여러 열에 대한 구분이 있는 CSV 형식의 로그 파일을 얻게 됩니다. 이제 로그를 더 쉽게 정렬하고 분석할 수 있습니다.

이 파라미터에 csvlog를 지정하는 경우 stderr 및 csvlog 파일이 모두 생성된다는 점에 유의하세요. rds.log_retention_period 및 로그 스토리지와 회전율에 영향을 주는 다른 설정을 고려하여 로그에서 사용하는 스토리지를 모니터링해야 합니다. stderr 및 csvlog를 사용하는 경우 로그에서 소비하는 스토리지가 두 배 이상 늘어납니다.

log_destination에 csvlog를 추가하고 stderr로만 되돌리려면 파라미터를 재설정해야 합니다. 이렇게 하려면 Amazon RDS 콘솔을 연 다음 인스턴스에 대한 사용자 지정 DB 클러스터 파라미터 그룹을 엽니다. log_destination 파라미터를 선택하고 파라미터 편집을 선택한 다음 재설정을 선택합니다.

로깅 구성에 대한 자세한 내용은 Amazon RDS 및 Aurora PostgreSQL 로그 작업: 1부를 참조하세요.

log_line_prefix 파라미터 이해하기

stderr 로그 형식은 각 로그 메시지에 log_line_prefix 파라미터에 지정된 세부 정보를 접두사로 지정합니다. 기본값은 다음과 같습니다.

```
%t:%r:%u@%d:[%p]:t
```

Aurora PostgreSQL 버전 16부터는 다음을 선택할 수도 있습니다.

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

stderr로 전송되는 각 로그 항목에는 선택한 값에 따라 다음 정보가 포함됩니다.

- `%t` - 밀리초 단위를 제외한 로그 항목 시간
- `%m` - 밀리초 단위를 포함한 로그 항목 시간
- `%r` - 원격 호스트 주소
- `%u@%d` - 사용자 이름 @ 데이터베이스 이름
- `[%p]` - 프로세스 ID(사용 가능한 경우)
- `%l` - 세션당 로그 행 번호
- `%e` – SQL 오류 코드
- `%s` – 프로세스 시작 타임스탬프
- `%v` - 가상 트랜잭션 ID
- `%x` – 트랜잭션 ID
- `%c` – 세션 ID
- `%q` - 세션이 아닌 종결자
- `%a` – 애플리케이션 이름

Aurora PostgreSQL DB 클러스터에 쿼리 로깅을 활성화합니다.

다음 테이블에 나열된 파라미터 중 일부를 설정하여 쿼리, 잠금을 기다리는 쿼리, 체크포인트 및 기타 여러 세부 정보를 포함하여 데이터베이스 활동에 대한 보다 자세한 정보를 수집할 수 있습니다. 이 주제에서는 쿼리 로깅에 중점을 둡니다.

파라미터	Default	설명
log_connections	–	성공한 연결을 모두 기록합니다. 이 파라미터를 `log_disconnections` 에서 사용하여 연결 이탈을 감지하는 방법을 알아보려면 [폴링으로 Aurora PostgreSQL 연결 이탈 관리](#) 섹션을 참조하세요.
log_disconnections	–	각 세션의 끝과 기간을 로깅합니다. 이 파라미터를 `log_connections` 에서 사용하여 연결

파라미터	Default	설명
		이탈을 감지하는 방법을 알아보려면 폴링으로 Aurora PostgreSQL 연결 이탈 관리 섹션을 참조하세요.
log_checkpoints	–	Aurora PostgreSQL에는 적용되지 않음
log_lock_waits	–	오랜 잠금 대기 시간을 기록합니다. 이 파라미터는 기본적으로 설정되어 있지 않습니다.
log_min_duration_sample	–	(밀리초) 문 샘플이 로그되는 최소 실행 시간을 설정합니다. 샘플 크기는 `log_statement_sample_rate` 파라미터를 사용하여 설정합니다.
log_min_duration_statement	–	지정된 시간 이상 실행되는 모든 SQL 문은 로깅됩니다. 이 파라미터는 기본적으로 설정되어 있지 않습니다. 이 파라미터를 활성화하면 최적화되지 않은 쿼리를 찾는 데 도움이 될 수 있습니다.
log_statement	–	기록할 문 유형을 설정합니다. 기본적으로 이 파라미터는 설정되어 있지 않지만 `all`, `ddl` 또는 `mod`로 변경하여 로깅하려는 SQL 문 유형을 지정할 수 있습니다. 이 파라미터에 none 이외의 다른 것을 지정하면 추가 단계를 수행하여 로그 파일에 암호가 노출되지 않도록 해야 합니다. 자세한 내용은 쿼리 로깅 사용 시 암호 노출 위험 완화 섹션을 참조하세요.
log_statement_sample_rate	–	로깅되도록 `log_min_duration_sample` 에 지정된 시간을 초과하는 문의 비율로, 0.0에서 1.0 사이의 부동 소수점 값으로 표시됩니다.
log_statement_stats	–	누적 성능 통계를 서버 로그에 기록합니다.

로깅을 사용하여 성능이 느린 쿼리 찾기

SQL 문 및 쿼리를 로깅하여 성능이 느린 쿼리를 찾을 수 있습니다. 이 섹션에 설명된 대로 `log_statement` 및 `log_min_duration` 파라미터 설정을 수정하여 이 기능을 활성화합니다. Aurora PostgreSQL DB 클러스터에 대한 쿼리 로깅을 활성화하기 전에 로그에서 암호가 노출될 수 있는 사항과 위험을 완화하는 방법을 알고 있어야 합니다. 자세한 내용은 [쿼리 로깅 사용 시 암호 노출 위험 완화] 섹션을 참조하세요.

다음에서 `log_statement` 및 `log_min_duration` 파라미터에 대한 참조 정보를 확인할 수 있습니다.

log_statement

이 파라미터는 로그에 전송해야 하는 SQL 문의 유형을 지정합니다. 기본값은 none입니다. 이 파라미터를 `all`, `ddl` 또는 `mod`로 변경할 경우 로그에 암호가 노출될 위험을 줄이려면 권장 조치를 취해야 합니다. 자세한 내용은 [쿼리 로깅 사용 시 암호 노출 위험 완화] 섹션을 참조하세요.

모두

모든 문을 로깅합니다. 이 설정은 디버깅 용도로 사용하는 것이 좋습니다.

ddl

모든 데이터 정의 언어(DDL) 문(CREATE, ALTER, DROP 등)을 로깅합니다.

mod

NSERT, UPDATE, DELETE 등 데이터를 수정하는 모든 DDL 문과 데이터 조작 언어(DML) 문을 로깅합니다.

없음

SQL 문이 로깅되지 않습니다. 로그에서 암호가 노출될 위험을 피하려면 이 설정을 사용하는 것이 좋습니다.

log_min_duration_statement

지정된 시간 이상 실행되는 모든 SQL 문은 로깅됩니다. 이 파라미터는 기본적으로 설정되어 있지 않습니다. 이 파라미터를 활성화하면 최적화되지 않은 쿼리를 찾는 데 도움이 될 수 있습니다.

–1–2147483647

문이 로깅되는 런타임의 밀리초(ms) 수입니다.

쿼리 로깅 설정

이 단계에서는 Aurora PostgreSQL DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용한다고 가정합니다.

1. `log_statement` 파라미터를 `all`로 설정합니다. 다음 예에서는 이 파라미터가 설정에서 `postgresql.log`에 기록되는 정보를 보여줍니다.

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
----------------------- END OF LOG ----------------------
```

2. `log_min_duration_statement` 파라미터를 설정합니다. 다음 예제에서는 이 파라미터가 `postgresql.log`로 설정되어 있을 때 1 파일에 기록되는 정보를 보여줍니다.

   `log_min_duration_statement` 파라미터에 지정된 기간을 초과하는 쿼리가 로깅됩니다. 다음은 그 한 예입니다. Amazon RDS 콘솔에서 Aurora PostgreSQL DB 클러스터에 대한 로그 파일을 볼 수 있습니다.

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
```

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
----------------------- END OF LOG ----------------------
```

쿼리 로깅 사용 시 암호 노출 위험 완화

암호가 노출되지 않도록 `log_statement`를 none으로 설정하는 것이 좋습니다. `log_statement`를 `all`, `ddl`, 또는 `mod`로 설정하는 경우 다음 단계 중 하나 이상을 수행하는 것이 좋습니다.

- 클라이언트의 경우 민감한 정보를 암호화합니다. 자세한 내용은 PostgreSQL 설명서의 [암호화 옵션에 관한 문서]를 참조하세요. `CREATE` 및 `ALTER` 문의 `ENCRYPTED`(및 `UNENCRYPTED`) 옵션을 사용합니다. 자세한 내용은 PostgreSQL 설명서에서 [CREATE USER]를 참조하세요.
- Aurora PostgreSQL DB 클러스터의 경우 PostgreSQL Auditing(pgAudit) 확장을 설정하고 사용합니다. 이 확장은 로그로 전송된 CREATE 및 ALTER 문에서 민감한 정보를 삭제합니다. 자세한 내용은 [pgAudit를 사용하여 데이터베이스 활동 로깅] 섹션을 참조하세요.
- CloudWatch 로그에 대한 액세스를 제한합니다.
- IAM 등의 보다 강력한 인증 메커니즘을 사용합니다.

AWS CloudTrail에서 Amazon Aurora API 호출 모니터링

AWS CloudTrail은 AWS 계정을 감사하는 데 도움이 되는 AWS 서비스입니다. AWS CloudTrail을 생성하면 AWS 계정에서 활성화됩니다. CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

주제

- CloudTrail를 Amazon Aurora와 통합
- Amazon Aurora 로그 파일 항목

CloudTrail를 Amazon Aurora와 통합

모든 Amazon Aurora 작업은 CloudTrail에 의해 로깅됩니다. CloudTrail은 Amazon Aurora에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공합니다.

CloudTrail 이벤트

CloudTrail은 Amazon Aurora에 대한 API 호출을 이벤트로 캡처합니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. 이벤트에는 Amazon RDS 콘솔 호출과 Amazon RDS API 작업에 대한 코드 호출이 포함됩니다.

Amazon Aurora 작업은 이벤트 기록(Event history)의 CloudTrail 이벤트에 기록됩니다. CloudTrail 콘솔을 사용하여 AWS 리전에서 지난 90일간 기록된 API 활동 및 이벤트를 확인할 수 있습니다. 자세한 내용은 CloudTrail 이벤트 기록에서 이벤트 보기를 참조하세요.

CloudTrail 추적

Amazon Aurora에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. 추적은 지정된 Amazon S3 버킷에 이벤트를 전송할 수 있도록 하는 구성입니다. CloudTrail은 일반적으로 계정 활동 15분 이내에 로그 파일을 전송합니다.

> **Note**
>
> 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다.

AWS 계정에 대해 두 가지 추적 유형(모든 리전에 적용되는 추적 또는 한 리전에 적용되는 추적)을 생성할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 지역에 추적이 적용됩니다.

또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 수신 및 여러 계정에서 CloudTrail 로그 파일 수신

Amazon Aurora 로그 파일 항목

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출에 대한 순서 지정된 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

다음은 `CreateDBInstance` 작업을 보여 주는 CloudTrail 로그 항목이 나타낸 예제입니다.


```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
```

```
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
```

```
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
```

```
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


앞의 예의 `userIdentity` 요소에 표시된 것처럼 모든 이벤트 또는 로그 항목에는 요청을 생성한 사용자에 대한 정보가 포함됩니다. 자격 증명 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 보안 인증 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

`userIdentity`에 대한 자세한 내용은 [CloudTrail userIdentity 요소]를 참조하십시오. `CreateDBInstance` 및 기타 Amazon Aurora 작업에 대한 자세한 내용은 [Amazon RDS API 참조]를 참조하세요.

데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링

데이터베이스 활동 스트림을 사용하면 데이터베이스 활동 스트림을 거의 실시간으로 모니터링할 수 있습니다.

주제

데이터베이스 활동 스트림 개요

Amazon Aurora 데이터베이스 관리자는 데이터베이스를 보호하고 규정 준수 및 규제 요건을 충족해야 합니다. 한 가지 전략은 데이터베이스 활동 스트림을 모니터링 도구와 통합하는 것입니다. 이렇게 하면 Amazon Aurora 클러스터에서 감사 활동에 대한 경보를 모니터링하고 설정할 수 있습니다.

보안 위협은 외부 및 내부 위협입니다. 내부 위협으로부터 보호하기 위해 데이터베이스 활동 스트림 기능을 구성하여 데이터 스트림에 대한 관리자 액세스를 제어할 수 있습니다. DBA는 스트림의 수집, 전송, 저장 및 처리에 대한 액세스 권한이 없습니다.

목차

데이터베이스 활동 스트림 작동 방식

Amazon Aurora에서는 클러스터 수준에서 데이터베이스 활동 스트림을 시작합니다. 클러스터 내의 모든 DB 인스턴스는 데이터베이스 활동 스트림을 사용하도록 설정되어 있습니다.

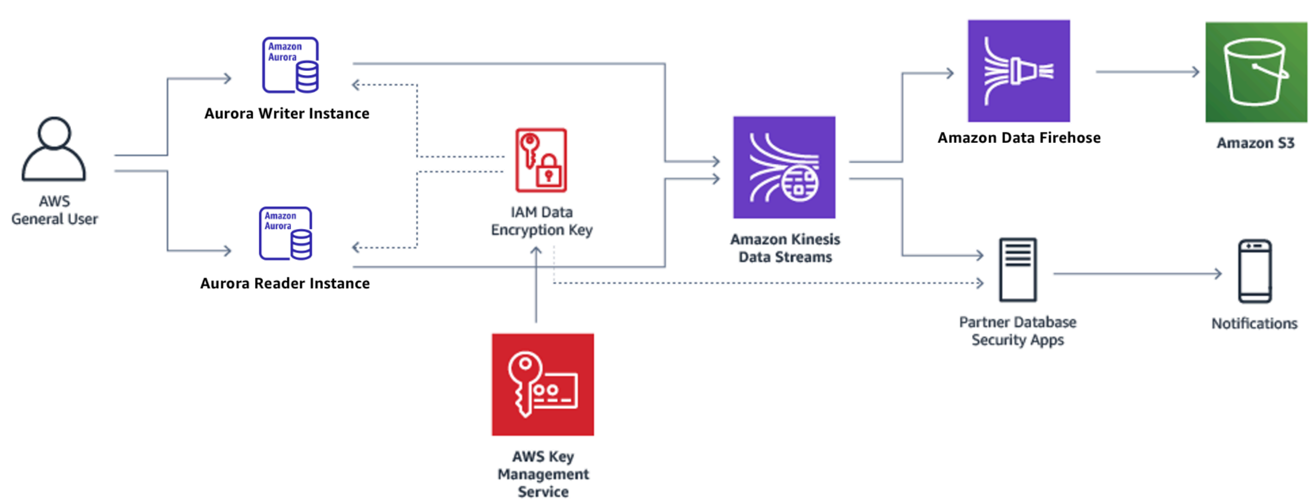
Aurora DB 클러스터는 활동을 Amazon Kinesis 데이터 스트림에 거의 실시간으로 푸시합니다. Kinesis 스트림이 자동으로 생성됩니다. Kinesis에서 Amazon Data Firehose 및 AWS Lambda와 같은 AWS 서비스를 구성하여 스트림을 사용하고 데이터를 저장할 수 있습니다.

> **Important**
>
> Amazon Aurora에서 데이터베이스 활동 스트림은 무료 기능이지만, Amazon Kinesis는 데이터 스트림에 대한 요금을 부과합니다. 자세한 내용은 [Amazon Kinesis Data Streams 요금]을 참조하세요.

Aurora Global Database를 사용하는 경우 데이터베이스 활동 스트림을 각 DB 클러스터에서 별도로 시작합니다. 각 클러스터는 자체 AWS 리전 내의 자체 Kinesis 스트림에 감사 데이터를 제공합니다. 장애 조치 중에 활동 스트림이 다르게 작동하지 않습니다. 평소와 같이 글로벌 데이터베이스를 계속 감사합니다.

규정 준수 관리용 애플리케이션이 데이터베이스 활동 스트림을 사용하도록 구성할 수 있습니다. Aurora PostgreSQL의 경우, 규정 준수 애플리케이션에는 IBM의 Security Guardium, Imperva의 SecureSphere Database Audit and Protection이 포함됩니다. 이 애플리케이션은 스트림을 사용하여 Aurora DB 클러스터에 대한 경보를 생성하고 활동 감사를 수행할 수 있습니다.

다음 그래픽은 Amazon Data Firehose로 구성된 Aurora DB 클러스터를 보여줍니다.



데이터베이스 활동 스트림에 대한 비동기식 및 동기식 모드

다음 모드 중 하나로 데이터베이스 세션에서 데이터베이스 활동 이벤트를 처리하도록 선택할 수 있습니다.

- 비동기식 모드 – 데이터베이스 세션이 활동 스트림 이벤트를 생성하면 세션은 즉시 정상 활동으로 되돌아갑니다. 배경에서 내구성 있는 레코드에 활동 스트림 이벤트가 생성됩니다. 배경 작업에서 오류가 발생하면 RDS 이벤트가 전송됩니다. 이 이벤트는 활동 스트림 이벤트 레코드가 분실되었을 수 있는 기간의 시작과 끝을 나타냅니다.

  비동기 모드는 활동 스트림의 정확성보다 데이터베이스 성능을 우선시합니다.

  > **Note**
  >
  > 비동기 모드는 Aurora PostgreSQL 및 Aurora MySQL 모두에 사용할 수 있습니다.

- 동기식 모드 – 데이터베이스 세션이 활동 스트림 이벤트를 생성할 때, 이 세션은 이벤트가 지속될 때까지 다른 활동을 차단합니다. 어떤 이유로 이벤트를 내구성 있게 만들 수 없으면 데이터베이스 세션은 정상적인 활동으로 돌아갑니다. 그러나 활동 스트림 레코드가 잠시 분실되었을 수 있음을 나타내는 RDS 이벤트가 전송됩니다. 두 번째 RDS 이벤트는 시스템이 정상 상태로 돌아간 후에 전송됩니다.

  동기 모드는 데이터베이스 성능보다 활동 스트림의 정확성을 우선시합니다.

  > **Note**
  >
  > Aurora PostgreSQL에 동기 모드를 사용할 수 있습니다. Aurora MySQL에는 동기 모드를 사용할 수 없습니다.

데이터베이스 활동 스트림의 요구 사항 및 제한 사항

Aurora에서 데이터베이스 활동 스트림에는 다음과 같은 요구 사항과 제한 사항이 있습니다.

- 데이터베이스 활동 스트림에는 Amazon Kinesis를 사용해야 합니다.
- 데이터베이스 활동 스트림은 항상 암호화되므로 AWS Key Management Service(AWS KMS)를 사용해야 합니다.
- Amazon Kinesis 데이터 스트림에 추가 암호화를 적용하는 것은 이미 AWS KMS 키를 사용하여 암호화된 데이터베이스 활동 스트림과 호환되지 않습니다.

- DB 클러스터 수준에서 데이터베이스 활동 스트림을 시작합니다. 클러스터에 DB 인스턴스를 추가하는 경우 인스턴스에서 활동 스트림을 시작할 필요가 없이 자동으로 감사됩니다.
- Aurora Global Database에서는 활동 스트림을 각 DB 클러스터에서 별도로 시작해야 합니다. 각 클러스터는 자체 AWS 리전 내의 자체 Kinesis 스트림에 감사 데이터를 제공합니다.
- Aurora PostgreSQL에서는 메이저 버전 업그레이드 전에 데이터베이스 활동 스트림을 중지해야 합니다. 업그레이드가 완료된 후 데이터베이스 활동 스트림을 시작할 수 있습니다.

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. Aurora 및 데이터베이스 활동 스트림의 버전 및 지역 가용성에 대한 자세한 내용은 데이터베이스 활동 스트림을 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

데이터베이스 활동 스트림이 지원되는 DB 인스턴스 클래스

Aurora MySQL의 경우 데이터베이스 활동 스트림을 다음 DB 인스턴스 클래스와 함께 사용할 수 있습니다.

- db.r8g.*large
- db.r7g.*large
- db.r7i.*large
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

Aurora PostgreSQL의 경우 데이터베이스 활동 스트림을 다음 DB 인스턴스 클래스와 함께 사용할 수 있습니다.

- db.r8g.*large
- db.r7i.*large
- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large

- db.r5.*large
- db.r4.*large
- db.x2g.*

Aurora MySQL 데이터베이스 활동 스트림에 대한 네트워크 사전 조건

다음 섹션에서는 Virtual Private Cloud(VPC)를 데이터베이스 활동 스트림에 사용하도록 구성하는 방법을 알아볼 수 있습니다.

> **Note**
>
> Aurora MySQL 네트워크 사전 조건은 다음 엔진 버전에 적용됩니다.
>
> - Aurora MySQL 버전 2에서 최대 2.11.3까지
> - Aurora MySQL 버전 2.12.0
> - Aurora MySQL 버전 3에서 최대 3.04.2까지

주제

- AWS KMS 엔드포인트의 사전 조건
- 퍼블릭 가용성을 위한 사전 조건
- 프라이빗 가용성을 위한 사전 조건

AWS KMS 엔드포인트의 사전 조건

활동 스트림을 사용하는 Aurora MySQL 클러스터의 인스턴스는 AWS KMS 엔드포인트에 액세스할 수 있어야 합니다. Aurora MySQL 클러스터에 대해 데이터베이스 활동 스트림을 활성화하기 전에 이 요구 사항이 충족되었는지 확인합니다. Aurora 클러스터를 퍼블릭으로 사용할 수 있는 경우 이 요구 사항은 자동으로 충족됩니다.

> **Important**
>
> Aurora MySQL DB 클러스터가 AWS KMS 엔드포인트에 액세스할 수 없는 경우, 활동 스트림의 작동이 중지됩니다. 이 경우 Aurora에서는 RDS 이벤트를 사용하여 이 문제에 대해 알립니다.

퍼블릭 가용성을 위한 사전 조건

Aurora DB 클러스터를 퍼블릭으로 설정하려면 다음 요구 사항을 충족해야 합니다.

- AWS Management Console클러스터 세부 정보 페이지의 퍼블릭 액세스 가능이 예입니다.
- DB 클러스터가 Amazon VPC의 퍼블릭 서브넷에 있습니다. 퍼블릭 액세스 DB 인스턴스에 대한 자세한 내용은 VPC에서 DB 클러스터를 사용한 작업 단원을 참조하십시오. 퍼블릭 Amazon VPC 서브넷에 대한 자세한 내용은 VPC 및 서브넷을 참조하세요.

프라이빗 가용성을 위한 사전 조건

Aurora DB 클러스터가 VPC 퍼블릭 서브넷 내에 있고 퍼블릭 액세스 상태가 아니면 프라이빗 상태입니다. 클러스터를 프라이빗으로 유지하고 데이터베이스 활동 스트림과 함께 사용하려면 다음과 같은 옵션이 있습니다.

- VPC에 NAT(Network Address Translation)를 구성합니다. 자세한 내용은 NAT 게이트웨이 단원을 참조하세요.
- VPC에 AWS KMS 엔드포인트를 생성합니다. 이 옵션은 구성하기가 더 쉽기 때문에 권장됩니다.

VPC에 AWS KMS 엔드포인트 생성하기

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 엔드포인트를 선택합니다.
3. 엔드포인트 생성을 선택합니다.

   엔드포인트 생성페이지가 나타납니다.
4. 다음을 따릅니다.
   - 서비스 범주에서 AWS 서비스를 선택합니다.
   - 서비스 이름(Service Name)에서 com.amazonaws.***region***.kms를 선택합니다. 여기서 *region*은 클러스터가 위치한 AWS 리전입니다.
   - VPC에서 클러스터가 위치한 VPC 선택합니다.
5. 엔드포인트 생성을 선택합니다.

VPC 종단점 구성에 대한 자세한 내용은 VPC Endpoints를 참조하십시오.

데이터베이스 활동 스트림 시작

클러스터 수준에서 활동 스트림을 시작하여 Aurora DB 클러스터의 모든 인스턴스에서 데이터베이스 활동을 모니터링합니다. 또한 클러스터에 추가된 모든 DB 인스턴스도 자동으로 모니터링됩니다. Aurora Global Database를 사용하는 경우 데이터베이스 활동 스트림을 각 DB 클러스터에서 별도로 시작합니다. 각 클러스터는 자체 AWS 리전 내의 자체 Kinesis 스트림에 감사 데이터를 제공합니다.

활동 스트림을 시작하면 감사 정책에 구성한 각 데이터베이스 활동 이벤트가 활동 스트림 이벤트를 생성합니다. `CONNECT` 및 `SELECT` 같은 SQL 명령은 액세스 이벤트를 생성합니다. `CREATE` 및 `INSERT` 같은 SQL 명령은 변경 이벤트를 생성합니다.

Console

데이터베이스 활동 스트림을 시작하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 활동 스트림을 시작하려는 DB 클러스터를 선택합니다.
4. 작업의 경우 Start activity stream(활동 스트림 시작)을 선택합니다.

   데이터베이스 활동 스트림 시작: *##* 창이 나타납니다. 여기서 *##*은 사용자의 DB 클러스터입니다.
5. 다음 설정을 입력합니다.

   - AWS KMS key의 경우, AWS KMS keys 목록에서 키를 선택합니다.

     > Note
     >
     > Aurora MySQL 클러스터에서 KMS 키에 액세스할 수 없는 경우, Aurora MySQL 데이터베이스 활동 스트림에 대한 네트워크 사전 조건 지침에 따라 먼저 해당 액세스를 활성화합니다.

     Aurora가 KMS 키를 사용하여 키를 암호화하면 암호화된 키가 데이터베이스 활동을 암호화합니다. 기본 키가 아닌 KMS 키를 선택합니다. 암호화 키 및 AWS KMS에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 AWS Key Management Service란 무엇입니까?를 참조하세요.

- Database activity stream mode(데이터베이스 활동 스트림 모드)의 경우 Asynchronous(비동기) 또는 Synchronous(동기)를 선택하십시오.

> **Note**
>
> 이 선택 사항은 Aurora PostgreSQL에만 적용됩니다. Aurora MySQL의 경우 비동기 모드만 사용할 수 있습니다.

- [즉시(Immediately))를 선택합니다.

  [즉시(Immediately)]를 선택하면 DB 클러스터가 바로 다시 시작됩니다. [다음 유지 관리 기간 중(During the next maintenance window)]을 선택하면 DB 클러스터는 즉시 다시 시작되지 않습니다. 이 경우 데이터베이스 활동 스트림은 다음 유지 관리 기간까지 시작되지 않습니다.

6. 데이터베이스 활동 스트림 시작을 선택합니다.

   DB 클러스터에 대한 상태는 활동 스트림이 시작 중임을 보여줍니다.

> **Note**
>
> `You can't start a database activity stream in this configuration` 오류가 발생하는 경우 DB 클러스터가 지원되는 인스턴스 클래스를 사용하고 있는지 데이터베이스 활동 스트림이 지원되는 DB 인스턴스 클래스에서 확인하세요.

AWS CLI

DB 클러스터에 대해 데이터베이스 활동 스트림을 시작하려면 start-activity-stream AWS CLI 명령을 사용하여 DB 클러스터를 구성합니다.

- `--resource-arn arn` - DB 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다.
- `--mode sync-or-async` - 동기식(sync) 또는 비동기식(async) 모드를 지정합니다. Aurora PostgreSQL의 경우 두 값 중 하나를 선택할 수 있습니다. Aurora SQL의 경우 `async`를 지정합니다.
- `--kms-key-id key` - 데이터베이스 활동 스트림에서 메시지를 암호화하기 위한 KMS 키 식별자를 지정합니다. AWS KMS 키 식별자는 AWS KMS key의 키 ARN, 키 ID, 별칭 ARN 또는 별칭 이름입니다.

다음 예에서는 비동기 모드에서 DB 클러스터 에 대한 데이터베이스 활동 스트림을 시작합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

Windows의 경우:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

Amazon RDS API

DB 클러스터에 대해 데이터베이스 활동 스트림을 시작하려면 StartActivityStream 작업을 사용하여 클러스터를 구성합니다.

아래 파라미터를 사용하여 작업을 호출하세요.

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

> **Note**
>
> Aurora PostgreSQL 데이터베이스의 현재 버전으로 데이터베이스 활동 스트림을 시작할 수 없다는 오류가 발생하면 데이터베이스 활동 스트림을 시작하기 전에 Aurora PostgreSQL에 최신 패치를 적용합니다. Aurora PostgreSQL 데이터베이스 업그레이드에 대한 자세한 내용은 Amazon Aurora DB 클러스터 업그레이드 섹션을 참조하세요.
> 다음은 Aurora PostgreSQL을 사용하여 데이터베이스 활동 스트림을 시작하기 위한 최소 패치 버전입니다.
>
> - 3.4.15(11.9.15), 11.21.10

- 12.9.15, 12.15.9, 12.16.10, 12.17.7, 12.18.5, 12.19.4, 12.20.3, 12.22.3
- 13.9.12, 13.11.9, 13.12.10, 13.13.7, 13.14.5, 13.15.4, 13.16.3, 13.18.3
- 14.6.12, 14.8.9, 14.9.10, 14.10.7, 14.11.5, 14.12.4, 14.13.3, 14.15.3
- 15.3.9, 15.4.10, 15.5.7, 15.6.5, 15.7.4, 15.8.3, 15.10.3
- 16.1.7, 16.2.5, 16.3.4, 16.4.3, 16.6.3

데이터베이스 활동 스트림 상태 가져오기

콘솔 또는 AWS CLI를 사용하여 에 대한 활동 스트림의 상태를 가져올 수 있습니다.

콘솔

데이터베이스 활동 스트림 상태 가져오기

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [데이터베이스(Databases)]를 선택하고 DB 클러스터 링크를 선택합니다.
3. 구성 탭을 선택하고 Database activity stream(데이터베이스 활동 스트림)에서 상태를 확인하십시오.

AWS CLI

describe-db-clusters CLI 요청에 대한 응답으로 DB 클러스터에 대한 활동 스트림 구성을 가져올 수 있습니다.

다음 예는 *my-cluster*를 설명합니다.

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

다음 예에서는 JSON 응답을 보여 줍니다. 다음 필드가 표시됩니다.

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
-

이러한 필드는 Aurora PostgreSQL 및 Aurora MySQL에서 동일합니다. 단, Aurora MySQL의 경우 `ActivityStreamMode`는 항상 `async`인 반면, Aurora PostgreSQL의 경우 `sync` 또는 `async`입니다.

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

RDS API

DescribeDBClusters 작업에 대한 응답으로 DB 클러스터에 대한 활동 스트림 구성을 가져올 수 있습니다.

데이터베이스 활동 스트림 중지

콘솔 또는 AWS CLI를 사용하여 활동 스트림을 중지할 수 있습니다.

DB 클러스터를 삭제하면 활동 스트림이 중지되고 기본 Amazon Kinesis 스트림이 자동으로 삭제됩니다.

콘솔

활동 스트림을 끄려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 활동 스트림을 중지하려는 DB 클러스터를 선택하세요.
4. 작업의 경우 Stop activity stream(작업 스트림 중지)을 선택합니다. Database Activity Stream(데이터베이스 활동 스트림) 창이 나타납니다.

a. [즉시(Immediately))를 선택합니다.

   [즉시(Immediately)]를 선택하면 DB 클러스터가 바로 다시 시작됩니다. [다음 유지 관리 기간 중(During the next maintenance window)]을 선택하면 DB 클러스터는 즉시 다시 시작되지 않습니다. 이 경우 데이터베이스 활동 스트림은 다음 유지 관리 기간까지 중지되지 않습니다.

b. [Continue]를 선택합니다.

AWS CLI

DB 클러스터에 대한 데이터베이스 활동 스트림을 중지하려면 AWS CLI 명령 stop-activity-stream을 사용하여 DB 클러스터를 구성하세요. `--region` 파라미터를 사용하여 DB 클러스터에 대한 AWS 리전을 식별합니다. `--apply-immediately` 파라미터는 선택 항목입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

Windows의 경우:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS API

DB 클러스터에 대해 데이터베이스 활동 스트림을 중지하려면 StopActivityStream 작업을 사용하여 클러스터를 구성합니다. `Region` 파라미터를 사용하여 DB 클러스터에 대한 AWS 리전을 식별합니다. `ApplyImmediately` 파라미터는 선택 항목입니다.

데이터베이스 활동 스트림 모니터링

데이터베이스 활동 스트림은 활동을 모니터링하고 보고합니다. 활동 스트림이 수집되어 Amazon Kinesis에 전송됩니다. Kinesis에서 활동 스트림을 모니터링하거나 다른 서비스 및 애플리케이션이 추가 분석을 위해 활동 스트림을 사용할 수 있습니다. AWS CLI 명령 `describe-db-clusters` 또는 RDS API `DescribeDBClusters` 작업을 사용하여 기본 Kinesis 스트림 이름을 찾을 수 있습니다.

Aurora는 다음과 같이 Kinesis 스트림을 관리합니다.

- Aurora는 24시간 보존 기간으로 Kinesis 스트림을 자동으로 생성합니다.
- Aurora는 필요한 경우 Kinesis 스트림 크기를 조정합니다.
- 데이터베이스 활동 스트림을 중지하거나 DB 클러스터를 삭제하면 Aurora에서 Kinesis 스트림을 삭제합니다.

다음 활동 범주가 모니터링되고 활동 스트림 감사 로그에 기록됩니다.

- SQL 명령 – 모든 SQL 명령이 감사되고 PL/SQL에서 준비된 문, 내장 함수 및 함수도 제공됩니다. 저장 프로시저에 대한 호출이 감사됩니다. 저장 프로시저 또는 함수 내에서 발급된 모든 SQL 문도 감사됩니다.
- 다른 데이터베이스 정보 – 모니터링되는 활동에는 전체 SQL 문, DML 명령의 영향을 받은 행의 행 수, 액세스된 객체 및 고유한 데이터베이스 이름이 포함됩니다. Aurora PostgreSQL의 경우 데이터베이스 활동 스트림은 바인딩 변수 및 저장 프로시저 파라미터도 모니터링합니다.

> **Important**
>
> 각 문의 전체 SQL 텍스트는 중요한 데이터를 포함하여 활동 스트림 감사 로그에 표시됩니다. 그러나 데이터베이스 사용자 암호는 다음 SQL 문에서와 같이 Aurora이 컨텍스트에서 판별할 수 있는 경우 수정됩니다.
>
> ```
> ALTER ROLE role-name WITH password
> ```

- 연결 정보 – 모니터링되는 활동에는 세션 및 네트워크 정보, 서버 프로세스 ID 및 종료 코드가 포함됩니다.

DB 인스턴스를 모니터링하는 동안 활동 스트림에 오류가 발생하면 RDS 이벤트를 통해 사용자에게 알립니다.

다음 섹션에서는 데이터베이스 활동 스트림에 액세스하고, 이를 감사 및 처리할 수 있습니다.

주제

- [AWS SDK를 사용하여 데이터베이스 활동 스트림 처리]

Amazon Kinesis에서 활동 스트림에 액세스

DB 클러스터에 대해 활동 스트림을 활성화하면 Kinesis 스트림이 생성됩니다. Kinesis에서 데이터베이스 활동을 실시간으로 모니터링할 수 있습니다. 데이터베이스 활동을 추가 분석하려면 Kinesis 스트림을 소비자 애플리케이션에 연결하면 됩니다. 또한 IBM의 Security Guardium 또는 Imperva의 SecureSphere Database Audit and Protection과 같은 규정 준수 관리 애플리케이션에 스트림을 연결할 수 있습니다.

RDS 콘솔 또는 Kinesis 콘솔에서 Kinesis 스트림에 액세스할 수 있습니다.

RDS 콘솔을 사용하여 Kinesis에서 활동 스트림에 액세스하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 활동 스트림을 시작한 DB 클러스터를 선택합니다.
4. Configuration(구성)을 선택합니다.
5. 데이터베이스 활동 스트림에서 Kinesis 스트림 아래의 링크를 선택합니다.
6. Kinesis 콘솔에서 모니터링을 선택하여 데이터베이스 활동 관찰을 시작합니다.

Kinesis 콘솔을 사용하여 Kinesis에서 활동 스트림에 액세스하는 방법

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis 스트림 목록에서 활동 스트림을 선택합니다.

   활동 스트림의 이름에는 접두사 `aws-rds-das-cluster-`와 그 뒤의 DB 클러스터의 리소스 ID가 포함됩니다. 다음은 예제입니다.

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   Amazon RDS 콘솔을 사용하여 DB 클러스터의 리소스 ID를 찾으려면 데이터베이스 목록에서 DB 클러스터를 선택한 다음 구성(Configuration) 탭을 선택합니다.

   AWS CLI를 사용하여 활동 스트림의 전체 Kinesis 스트림 이름을 찾으려면 [describe-db-clusters] 요청을 사용하고 응답에서 `ActivityStreamKinesisStreamName` 값을 기록합니다.
3. 데이터베이스 활동을 관찰하려면 모니터링을 선택하십시오.

Amazon Kinesis 사용에 대한 자세한 내용은 Amazon Kinesis Data Streams이란 무엇입니까?를 참조하십시오.

데이터베이스 활동 스트림에 대한 감사 로그 내용 및 예제

모니터링되는 이벤트는 데이터베이스 활동 스트림에 JSON 문자열로 표시됩니다. 구조는 `DatabaseActivityMonitoringRecord`를 포함하는 JSON 객체로 구성되며, 여기에는 `databaseActivityEventList` 활동 이벤트 배열이 포함됩니다.

> **Note**
>
> 데이터베이스 활동 스트림의 경우 `paramList` JSON 배열에는 최대 절전 모드 애플리케이션의 null 값이 포함되지 않습니다.

주제

- 활동 스트림 감사 로그 예제
- DatabaseActivityMonitoringRecords JSON 객체
- databaseActivityEvents JSON 객체

활동 스트림 감사 로그 예제

다음은 활동 이벤트 레코드의 해독된 JSON 감사 로그 샘플입니다.

Example Aurora PostgreSQL CONNECT SQL 문 의 활동 이벤트 레코드

다음 활동 이벤트 레코드는 의 psql 클라이언트(`clientApplication`)에서 `CONNECT` SQL 문(command)을 사용하여 로그인한 것을 보여줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
```

```
            "logTime": "2019-10-30 00:39:49.990579+00",
            "statementId": 1,
            "substatementId": 1,
            "objectType": null,
            "command": "CONNECT",
            "objectName": null,
            "databaseName": "postgres",
            "dbUserName": "rdsadmin",
            "remoteHost": "172.31.3.195",
            "remotePort": "49804",
            "sessionId": "5ce5f7f0.474b",
            "rowCount": null,
            "commandText": null,
            "paramList": [],
            "pid": 18251,
            "clientApplication": "psql",
            "exitCode": null,
            "class": "MISC",
            "serverVersion": "2.3.1",
            "serverType": "PostgreSQL",
            "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
            "serverHost": "172.31.3.192",
            "netProtocol": "TCP",
            "dbProtocol": "Postgres 3.0",
            "type": "record",
            "errorMessage": null
          }
        ]
      },
   "key":"decryption-key"
}
```


Example Aurora MySQL CONNECT SQL 문의 활동 이벤트 레코드

다음 활동 이벤트 레코드는 mysql 클라이언트(`clientApplication`)가 CONNECT SQL 문(command)을 사용하여 로그인한 것을 보여줍니다.


```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
```

```
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Example Aurora PostgreSQL CREATE TABLE 문의 활동 이벤트 레코드

다음 예시는 Aurora PostgreSQL에 대한 `CREATE TABLE` 이벤트를 보여줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
```

```
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```


Example Aurora MySQL CREATE TABLE 문의 활동 이벤트 레코드

다음 예시는 Aurora MySQL의 `CREATE TABLE` 문을 보여줍니다. 이 작업은 두 개의 개별 이벤트 레코드로 표시됩니다. 한 이벤트에는 `"class":"MAIN"`이 있습니다. 다른 이벤트에는 `"class":"AUX"`가 있습니다. 메시지는 순서에 관계없이 도착할 수 있습니다. `logTime` 이벤트의 `MAIN` 필드는 항상 해당 `logTime` 이벤트의 `AUX` 필드보다 빠릅니다.

다음 예제에서는 `class` 값이 MAIN인 이벤트를 보여 줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

다음 예제에서는 `class` 값이 AUX인 해당 이벤트를 보여 줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Aurora PostgreSQL SELECT 문의 활동 이벤트 레코드

다음 예시는 에 대한 SELECT 이벤트를 보여줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
```

```
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x010500000000000515000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```


Example Aurora MySQL SELECT 문의 활동 이벤트 레코드

다음 예시는 SELECT 이벤트를 보여줍니다.

다음 예제에서는 class 값이 MAIN인 이벤트를 보여 줍니다.


```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
```

```
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

다음 예제에서는 `class` 값이 AUX인 해당 이벤트를 보여 줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
```

```
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

DatabaseActivityMonitoringRecords JSON 객체

데이터베이스 작업 이벤트 레코드는 다음 정보가 포함된 JSON 객체에 있습니다.

JSON 필드	데이터 형식	설명
`type`	string	JSON 레코드 형식입니다. 이 값은 `DatabaseActivityMonitoringRecords` 입니다.
`version`	string	데이터베이스 작업 모니터링 레코드의 버전입니다. 생성되는 데이터베이스 활동 레코드의 버전은 DB 클러스터의 엔진 버전에 따라 다릅니다. • 버전 1.1 데이터베이스 작업 레코드는 엔진 버전 10.10 이상 마이너 버전 및 엔진 버전

JSON 필드	데이터 형식	설명
		11.5 이상을 실행하는 Aurora PostgreSQL DB 클러스터에 대해 생성됩니다. • 버전 1.0 데이터베이스 작업 레코드는 엔진 버전 10.7 및 11.4를 실행하는 Aurora PostgreSQL DB 클러스터에 대해 생성됩니다. 달리 명시된 경우를 제외하고 다음 필드는 모두 버전 1.0과 버전 1.1에 있습니다.
databaseActivityEvents	문자열	작업 이벤트를 포함하는 JSON 객체입니다.
키	문자열	databaseActivityEventList JSON 배열를 해독하는 데 사용되는 암호화 키

databaseActivityEvents JSON 객체

`databaseActivityEvents` JSON 객체에는 다음과 같은 정보가 포함되어 있습니다.

JSON 레코드의 최상위 필드

감사 로그의 각 이벤트는 JSON 형식의 레코드 내에 래핑됩니다. 이 레코드에는 다음 필드가 포함되어 있습니다.

type

이 필드는 항상 값이 `DatabaseActivityMonitoringRecords`입니다.

version

이 필드는 데이터베이스 활동 스트림 데이터 프로토콜 또는 계약 버전을 나타냅니다. 이는 사용 가능한 필드를 정의합니다.

버전 1.0은 Aurora PostgreSQL 버전 10.7 및 11.4에 대한 원래 데이터 활동 스트림 지원을 나타냅니다. 버전 1.1은 Aurora PostgreSQL 버전 10.10 이상 및 Aurora PostgreSQL 11.5 이상에 대한 데이터 활동 스트림 지원을 나타냅니다. 버전 1.1에는 추가 필드 `errorMessage` 및 `startTime`이

포함되어 있습니다. 버전 1.2는 Aurora MySQL 2.08 이상에 대한 데이터 활동 스트림 지원을 나타냅니다. 버전 1.2에는 추가 필드 `endTime` 및 `transactionId`가 포함되어 있습니다.

databaseActivityEvents

하나 이상의 활동 이벤트를 나타내는 암호화된 문자열입니다. base64 바이트 배열로 표현됩니다. 문자열을 해독하면 결과는 이 단원의 예제와 같이 필드가 있는 JSON 형식의 레코드입니다.

키

`databaseActivityEvents` 문자열을 암호화하는 데 사용되는 암호화된 데이터 키입니다. 이 키는 데이터베이스 활동 스트림을 시작할 때 제공한 AWS KMS key와(과) 동일합니다.

다음 예제에서는 이 레코드의 형식을 보여줍니다.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

`databaseActivityEvents` 필드의 내용을 해독하려면 다음 단계를 수행합니다.

1. 데이터베이스 활동 스트림을 시작할 때 제공한 키를 사용하여 key JSON 필드의 값을 복호화합니다. 이렇게 하면 데이터 암호화 키가 일반 텍스트로 반환됩니다.
2. Base64로 `databaseActivityEvents` JSON 필드의 값을 디코딩하여 감사 페이로드의 암호화 텍스트를 이진 형식으로 가져옵니다.
3. 첫 번째 단계에서 디코딩한 데이터 암호화 키를 사용하여 이진 암호화 텍스트를 해독합니다.
4. 해독된 페이로드의 압축을 풉니다.
   - 암호화된 페이로드가 `databaseActivityEvents` 필드에 있습니다.
   - `databaseActivityEventList` 필드에는 감사 레코드 배열이 포함되어 있습니다. 배열의 `type` 필드는 `record` 또는 `heartbeat`일 수 있습니다.

감사 로그 활동 이벤트 레코드는 다음 정보가 포함된 JSON 객체입니다.

JSON 필드	데이터 형식	설명
`type`	string	JSON 레코드 형식입니다. 이 값은 `DatabaseActivityMonitoringRecord` 입니다.
`clusterId`	string	DB 클러스터 리소스 식별자입니다. DB 클러스터 속성 `DbClusterResourceId` 에 해당합니다.
`instanceId`	string	DB 인스턴스 리소스 식별자입니다. DB 인스턴스 속성 `DbiResourceId` 에 해당합니다.
databaseActivityEventList JSON 배열	string	활동 감사 레코드 또는 하트비트 메시지의 배열입니다.

데이터베이스 활동 스트림에 대한 databaseActivityEventList JSON 배열

감사 로그 페이로드는 암호화된 `databaseActivityEventList` JSON 배열입니다. 다음 표에는 감사 로그의 복호화된 `DatabaseActivityEventList` 배열에 있는 각 활동 이벤트의 필드가 알파벳 순으로 나열되어 있습니다. 이 필드는 Aurora PostgreSQL을 사용하는지 Aurora MySQL을 사용하는지에 따라 다릅니다. 해당 데이터베이스 엔진에 적용되는 표를 참조하십시오.

> **Important**
>
> 이벤트 구조는 변경될 수 있습니다. Aurora는 향후 활동 이벤트에 새 필드를 추가할 수 있습니다. JSON 데이터를 구문 분석하는 애플리케이션에서 코드는 알 수 없는 필드 이름에 대해 무시할 수 있는지 또는 적절한 작업을 수행할 수 있는지 확인합니다.

Aurora PostgreSQL의 databaseActivityEventList 필드

Aurora PostgreSQL에 대한 `databaseActivityEventList` 필드는 다음과 같습니다.

필드	데이터 형식	설명
`class`	string	활동 이벤트의 클래스입니다. Aurora PostgreSQL에 유효한 값은 다음과 같습니다.

필드	데이터 형식	설명
		• ALL • CONNECT – 연결 또는 연결 해제 이벤트. • DDL – ROLE 클래스의 문 목록에 포함되지 않은 DDL 문. • FUNCTION – 함수 호출 또는 DO 블록. • MISC – DISCARD, FETCH, CHECKPOINT 또는 VACUUM 같은 기타 명령 • NONE • READ – 소스가 관계 또는 쿼리인 경우 SELECT 또는 COPY 문입니다. • ROLE – GRANT, REVOKE, CREATE/ALTER/DROP ROLE을 포함한 역할 및 권한과 관련된 문. • WRITE – 대상이 관계인 경우 INSERT, UPDATE, DELETE, TRUNCATE 또는 COPY 문.
clientApplication	string	클라이언트가 보고한 대로 클라이언트가 연결에 사용한 애플리케이션입니다. 클라이언트는 이 정보를 제공할 필요가 없으므로 값은 null일 수 있습니다.
command	string	명령 세부 정보가 없는 SQL 명령의 이름.

필드	데이터 형식	설명
`commandText`	string	사용자가 전달한 실제 SQL 문. Aurora PostgreSQL의 경우 값은 원래 SQL 문과 동일합니다. 이 필드는 연결 또는 연결 해제 레코드를 제외한 모든 유형의 레코드에 사용되며 이 경우 값은 null입니다. **Important** 각 문의 전체 SQL 텍스트는 중요한 데이터를 포함하여 활동 스트림 감사 로그에 표시됩니다. 그러나 데이터베이스 사용자 암호는 다음 SQL 문에서와 같이 Aurora가 컨텍스트에서 판별할 수 있는 경우 수정됩니다. `ALTER ROLE role-name WITH password`
`databaseName`	string	사용자가 연결된 데이터베이스입니다.
`dbProtocol`	string	데이터베이스 프로토콜입니다(예: `Postgres 3.0`).
`dbUserName`	string	클라이언트가 인증한 데이터베이스 사용자.

필드	데이터 형식	설명
`errorMessage` (버전 1.1 데이터베이스 작업 레코드만 해당)	string	오류가 발생할 경우 DB 서버에서 생성된 오류 메시지로 이 필드가 채워집니다. 오류가 발생하지 않은 정상적인 문의 경우 `errorMessage` 값은 null입니다. 오류는 클라이언트에 표시되는 심각도 수준 `ERROR` 이상의 PostgreSQL 오류 로그 이벤트를 생성하는 모든 활동으로 정의됩니다. 자세한 내용은 PostgreSQL 메시지 심각도 수준을 참조하십시오. 예를 들어, 구문 오류 및 쿼리 취소는 오류 메시지를 생성합니다. 백그라운드 체크포인터 프로세스 오류와 같은 내부 PostgreSQL 서버 오류는 오류 메시지를 생성하지 않습니다. 그러나 이러한 이벤트에 대한 레코드는 로그 심각도 수준의 설정에 관계없이 생성됩니다. 이로 인해 공격자가 로깅을 해제하여 탐지를 방지할 수 없습니다. `exitCode` 필드도 참조하십시오.
`exitCode`	int	세션 종료 레코드에 사용되는 값. 정리 종료의 경우, 여기에 종료 코드가 포함됩니다. 일부 결함 시나리오에서는 종료 코드를 항상 얻을 수 있는 것은 아닙니다. PostgreSQL이 `exit()`을 수행하거나 연산자가 `kill -9` 같은 명령을 수행하는 경우가 그 예입니다. 오류가 발생할 경우 PostgreSQL 오류 코드에 나열된 SQL 오류 코드 `SQLSTATE`가 `exitCode` 필드에 표시됩니다. `errorMessage` 필드도 참조하십시오.
`logTime`	string	감사 코드 경로에 기록된 타임스탬프. SQL 문의 실행이 종료된 시간을 나타냅니다. `startTime` 필드도 참조하십시오.
`netProtocol`	string	네트워크 통신 프로토콜.

필드	데이터 형식	설명
`objectName`	string	SQL 문이 하나에서 작동하는 경우 데이터베이스 객체의 이름. 이 필드는 SQL 문이 데이터베이스 객체에서 작동하는 경우에만 사용됩니다. SQL 문이 객체에서 작동하지 않는 경우, 이 값은 null입니다.
`objectType`	string	테이블, 인덱스, 뷰 등의 데이터베이스 객체 유형입니다. 이 필드는 SQL 문이 데이터베이스 객체에서 작동하는 경우에만 사용됩니다. SQL 문이 객체에서 작동하지 않는 경우, 이 값은 null입니다. 유효한 값은 다음과 같습니다. • `COMPOSITE TYPE` • `FOREIGN TABLE` • `FUNCTION` • `INDEX` • `MATERIALIZED VIEW` • `SEQUENCE` • `TABLE` • `TOAST TABLE` • `VIEW` • `UNKNOWN`
`paramList`	string	SQL 문에 전달되는 쉼표로 구분된 파라미터의 배열입니다. SQL 문에 파라미터가 없는 경우 이 값은 빈 배열입니다.
`pid`	int	클라이언트 연결 서비스를 위해 할당된 백엔드 프로세스의 프로세스 ID입니다.
`remoteHost`	string	클라이언트 IP 주소 또는 호스트 이름입니다. Aurora PostgreSQL의 경우 데이터베이스의 `log_hostname` 파라미터 설정에 따라 사용되는 값이 달라집니다. `remoteHost` 값에는 `rdsadmin` 사용자의 활동을 나타내는 `[local]` 및 `localhost` 값도 포함됩니다.
`remotePort`	문자열	클라이언트 포트 번호.

필드	데이터 형식	설명
rowCount	int	SQL 문에 의해 영향을 받거나 검색된 테이블 행 수입니다. 이 필드는 데이터 조작 언어(DML) 문인 SQL 문에만 사용됩니다. SQL 문이 DML 문이 아닌 경우 이 값은 null입니다.
serverHost	문자열	데이터베이스 서버 호스트 IP 주소. serverHost 값에는 rdsadmin 사용자의 활동을 나타내는 [local] 및 localhost 값도 포함됩니다.
serverType	문자열	데이터베이스 서버 유형(예: PostgreSQL).
serverVersion	string	데이터베이스 서버 버전(예: Aurora PostgreSQL의 경우 2.3.1).
serviceName	string	서비스의 이름(예: Amazon Aurora PostgreSQL-Compatible edition).
sessionId	int	의사 고유 세션 식별자.
sessionId	int	의사 고유 세션 식별자.
startTime (버전 1.1 데이터베이스 작업 레코드만 해당)	string	SQL 문의 실행이 시작된 시간입니다. SQL 문의 대략적인 실행 시간을 계산하려면 logTime - startTime 을 사용합니다. logTime 필드도 참조하십시오.
statementId	int	클라이언트의 SQL 문에 대한 식별자입니다. 카운터는 세션 수준에 있으며 클라이언트가 입력한 각 SQL 문 단위로 증가합니다.
substatementId	int	SQL 하위 문에 대한 식별자입니다. 이 값은 statementId 필드로 식별되는 각 SQL 문에 대해 포함된 하위 명령문을 계산합니다.
type	string	이벤트 유형입니다. 유효한 값은 record 또는 heartbeat 입니다.

Aurora MySQL의 databaseActivityEventList 필드

Aurora MySQL에 대한 `databaseActivityEventList` 필드는 다음과 같습니다.

필드	데이터 형식	설명
`class`	string	활동 이벤트의 클래스입니다. Aurora MySQL에 유효한 값은 다음과 같습니다. • `MAIN` – SQL 문을 나타내는 기본 이벤트입니다. • `AUX` – 추가 세부 정보를 포함하는 보충 이벤트입니다. 예를 들어 객체의 이름을 바꾸는 문에는 새 이름을 반영하는 `AUX` 클래스가 포함된 이벤트가 있을 수 있습니다. 동일한 문에 해당하는 `MAIN` 및 `AUX` 이벤트를 찾으려면 `pid` 필드와 `statementId` 필드에 대해 동일한 값을 가진 다른 이벤트를 확인합니다.
`clientApplication`	string	클라이언트가 보고한 대로 클라이언트가 연결에 사용한 애플리케이션입니다. 클라이언트는 이 정보를 제공할 필요가 없으므로 값은 null일 수 있습니다.
`command`	string	SQL 문의 일반 범주입니다. 이 필드의 값은 `class` 값에 따라 달라집니다. `class`가 `MAIN`인 경우 값에는 다음이 포함됩니다. • `CONNECT` – 클라이언트 세션이 연결된 경우. • `QUERY` – SQL 문. `class` 값이 `AUX`인 하나 이상의 이벤트가 포함됩니다. • `DISCONNECT` – 클라이언트 세션의 연결이 끊어진 경우. • `FAILED_CONNECT` – 클라이언트가 연결을 시도하지만 연결할 수 없는 경우. • `CHANGEUSER` – MySQL 네트워크 프로토콜에 속하는 상태 변경으로, 사용자가 실행한 문이 아닌 경우. `class`가 `AUX`인 경우 값에는 다음이 포함됩니다.

<table>
<tr><th>필드</th><th>데이터 형식</th><th>설명</th></tr>
<tr><td></td><td></td><td>

<code>READ</code> – 소스가 관계 또는 쿼리인 경우 <code>SELECT</code> 또는 <code>COPY</code> 문입니다.
<code>WRITE</code> – 대상이 관계인 경우 <code>INSERT</code>, <code>UPDATE</code>, <code>DELETE</code>, <code>TRUNCATE</code> 또는 <code>COPY</code> 문.
<code>DROP</code> – 객체 삭제.
<code>CREATE</code> – 객체 생성.
<code>RENAME</code> – 객체 이름 바꾸기.
<code>ALTER</code> – 객체의 속성 변경.

</td></tr>
<tr><td><code>commandText</code></td><td>string</td><td>
<code>class</code> 값이 <code>MAIN</code>인 이벤트의 경우 이 필드는 사용자가 전달한 실제 SQL 문을 나타냅니다. 이 필드는 연결 또는 연결 해제 레코드를 제외한 모든 유형의 레코드에 사용되며 이 경우 값은 null입니다.
<code>class</code> 값이 <code>AUX</code>인 이벤트의 경우 이 필드에는 이벤트와 관련된 객체에 대한 보충 정보가 포함됩니다.
Aurora MySQL의 경우 따옴표와 같은 문자 앞에는 이스케이프 문자를 나타내는 백슬래시가 옵니다.
Important
각 문의 전체 SQL 텍스트는 중요한 데이터를 포함하여 감사 로그에 표시됩니다. 그러나 데이터베이스 사용자 암호는 다음 SQL 문에서와 같이 Aurora가 컨텍스트에서 판별할 수 있는 경우 수정됩니다.
<pre>mysql> SET PASSWORD = 'my-password ';</pre>
Note
보안 모범 사례로 여기에 표시된 프롬프트 이외의 암호를 지정하는 것이 좋습니다.
</td></tr>
</table>

필드	데이터 형식	설명
databaseName	문자열	사용자가 연결된 데이터베이스입니다.
dbProtocol	string	데이터베이스 프로토콜. Aurora MySQL의 경우 현재 이 값은 항상 MySQL입니다.
dbUserName	string	클라이언트가 인증한 데이터베이스 사용자.
endTime (버전 1.2 데이터베이스 활동 레코드만 해당)	string	SQL 문의 실행이 끝난 시간입니다. 협정 세계시(UTC) 형식으로 표시됩니다. SQL 문의 실행 시간을 계산하려면 endTime - startTime 을 사용합니다. startTime 필드도 참조하십시오.
errorMessage (버전 1.1 데이터베이스 작업 레코드만 해당)	string	오류가 발생할 경우 DB 서버에서 생성된 오류 메시지로 이 필드가 채워집니다. 오류가 발생하지 않은 정상적인 문의 경우 errorMessage 값은 null입니다. 오류는 클라이언트에 표시되는 심각도 수준 ERROR 이상의 MySQL 오류 로그 이벤트를 생성하는 모든 활동으로 정의됩니다. 자세한 내용은 MySQL 참조 설명서의 The Error Log를 참조하세요. 예를 들어, 구문 오류 및 쿼리 취소는 오류 메시지를 생성합니다. 백그라운드 체크포인터 프로세스 오류와 같은 내부 MySQL 서버 오류는 오류 메시지를 생성하지 않습니다. 그러나 이러한 이벤트에 대한 레코드는 로그 심각도 수준의 설정에 관계없이 생성됩니다. 이로 인해 공격자가 로깅을 해제하여 탐지를 방지할 수 없습니다. exitCode 필드도 참조하십시오.
exitCode	int	세션 종료 레코드에 사용되는 값. 정리 종료의 경우, 여기에 종료 코드가 포함됩니다. 일부 결함 시나리오에서는 종료 코드를 항상 얻을 수 있는 것은 아닙니다. 이 경우 이 값은 0이거나 비어 있을 수 있습니다.

필드	데이터 형식	설명
logTime	string	감사 코드 경로에 기록된 타임스탬프. 협정 세계시(UTC) 형식으로 표시됩니다. 문 기간을 계산하는 가장 정확한 방법은 startTime 및 endTime 필드를 참조하십시오.
netProtocol	string	네트워크 통신 프로토콜. Aurora MySQL의 경우 현재 이 값은 항상 TCP입니다.
objectName	string	SQL 문이 하나에서 작동하는 경우 데이터베이스 객체의 이름. 이 필드는 SQL 문이 데이터베이스 객체에서 작동하는 경우에만 사용됩니다. SQL 문이 객체에서 작동하지 않는 경우, 이 값은 비어 있습니다. 객체의 정규화된 이름을 구성하려면 databaseName 과 objectName 을 결합합니다. 쿼리에 여러 객체가 포함된 경우 이 필드는 쉼표로 구분된 이름 목록이 될 수 있습니다.
objectType	string	테이블, 인덱스 등의 데이터베이스 객체 유형입니다. 이 필드는 SQL 문이 데이터베이스 객체에서 작동하는 경우에만 사용됩니다. SQL 문이 객체에서 작동하지 않는 경우, 이 값은 null입니다. Aurora MySQL에 유효한 값은 다음과 같습니다. • INDEX • TABLE • UNKNOWN
paramList	string	이 필드는 Aurora MySQL에 사용되지 않으며 항상 null입니다.
pid	int	클라이언트 연결 서비스를 위해 할당된 백엔드 프로세스의 프로세스 ID입니다. 데이터베이스 서버가 다시 시작되면 pid가 변경되고 statementId 필드의 카운터가 다시 시작됩니다.

필드	데이터 형식	설명
remoteHost	string	SQL 문을 실행한 클라이언트의 IP 주소 또는 호스트 이름입니다. Aurora MySQL의 경우 데이터베이스의 skip_name_resolve 파라미터 설정에 따라 사용되는 값이 달라집니다. localhost 값은 rdsadmin 특수 사용자의 활동을 나타냅니다.
remotePort	string	클라이언트 포트 번호.
rowCount	int	SQL 문에 의해 반환되는 행의 수입니다. 예를 들어 SELECT 문에서 10개의 행을 반환하는 경우 rowCount는 10입니다. INSERT 또는 UPDATE 문의 경우 rowCount는 0입니다.
serverHost	문자열	데이터베이스 서버 인스턴스 식별자입니다.
serverType	문자열	데이터베이스 서버 유형(예: MySQL).
serverVersion	string	데이터베이스 서버 버전. Aurora MySQL의 경우 현재 이 값은 항상 MySQL 5.7.12입니다.
serviceName	string	서비스의 이름입니다. Aurora MySQL의 경우 현재 이 값은 항상 Amazon Aurora MySQL입니다.
sessionId	int	의사 고유 세션 식별자.
startTime (버전 1.1 데이터베이스 작업 레코드만 해당)	string	SQL 문의 실행이 시작된 시간입니다. 협정 세계시(UTC) 형식으로 표시됩니다. SQL 문의 실행 시간을 계산하려면 endTime - startTime 을 사용합니다. endTime 필드도 참조하십시오.
statementId	int	클라이언트의 SQL 문에 대한 식별자입니다. 카운터는 클라이언트가 입력한 각 SQL 문 단위로 증가합니다. DB 인스턴스를 다시 시작하면 카운터가 재설정됩니다.

필드	데이터 형식	설명
substatementId	int	SQL 하위 문에 대한 식별자입니다. 이 값은 `MAIN` 클래스가 있는 이벤트의 경우 1이고, AUX 클래스가 있는 이벤트의 경우 2입니다. `statementId` 필드를 사용하여 동일한 문에 의해 생성된 모든 이벤트를 식별합니다.
transactionId (버전 1.2 데이터베이스 활동 레코드만 해당)	int	트랜잭션의 식별자입니다.
type	string	이벤트 유형입니다. 유효한 값은 `record` 또는 `heartbeat` 입니다.

AWS SDK를 사용하여 데이터베이스 활동 스트림 처리

AWS SDK를 사용하여 프로그래밍 방식으로 활동 스트림을 처리할 수 있습니다. 다음은 제대로 작동하는 Java 및 Python 예시로, Kinesis 데이터 스트림을 처리하는 방법을 보여줍니다.

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
```

```
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
```

```
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
```

```
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
 GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
 NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;
```

```
        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

                // Decrypt
                final DecryptRequest decryptRequest = new DecryptRequest()
```

```
.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
```

```
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
```

```
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
 CREDENTIALS_PROVIDER, workerId);

 kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
 APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
```

```
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)
```

```
    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


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

데이터베이스 활동 스트림의 IAM 정책 예제

데이터베이스 활동 스트림에 대한 적절한 AWS Identity and Access Management(IAM) 역할 권한이 있는 사용자는 DB 클러스터에 대한 활동 스트림 설정을 작성, 시작, 중지하고 수정할 수 있습니다. 이러한 작업은 스트림의 감사 로그에 포함됩니다. 규정을 준수하는 최선의 방법은 DBA에 이러한 권한을 제공하지 않는 것입니다.

IAM 정책을 사용하여 데이터베이스 활동 스트림에 대한 액세스를 설정합니다. Aurora 인증에 대한 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요. IAM 정책 생성에 대한 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 단원을 참조하십시오.

Example 데이터베이스 활동 스트림 구성을 허용하는 정책

사용자에게 활동 스트림을 수정할 수 있는 세분화된 액세스를 제공하려면 IAM 정책에서 서비스별 작업 컨텍스트 키 `rds:StartActivityStream` 및 `rds:StopActivityStream`을 사용하세요. 다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 구성할 수 있도록 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
         {
             "Sid":"ConfigureActivityStreams",
             "Effect":"Allow",
             "Action": [
                 "rds:StartActivityStream",
                 "rds:StopActivityStream"
             ],
             "Resource":"*",
           }
      ]
}
```

Example 데이터베이스 활동 스트림 시작을 허용하는 정책

다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 시작할 수 있도록 허용합니다.

```
{
     "Version":"2012-10-17",
     "Statement":[
         {
             "Sid":"AllowStartActivityStreams",
             "Effect":"Allow",
             "Action":"rds:StartActivityStream",
             "Resource":"*"
         }
     ]
}
```

Example 데이터베이스 활동 스트림 중지를 허용하는 정책

다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 중지할 수 있도록 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example 데이터베이스 활동 스트림 시작을 거부하는 정책

다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 시작하지 못하게 합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example 데이터베이스 활동 스트림 중지를 거부하는 정책

다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 중지하지 못하게 합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
```

```
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

Amazon Aurora용 Amazon GuardDuty RDS Protection을 이용한 위협 모니터링

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

GuardDuty RDS Protection은 Amazon Aurora 데이터베이스에 대한 잠재적 액세스 위협 관련 로그인 이벤트를 분석하고 프로파일링합니다. RDS Protection을 켜면 GuardDuty는 Aurora 데이터베이스에서 RDS 로그인 이벤트를 사용합니다. RDS Protection은 이러한 이벤트를 모니터링하고 잠재적인 내부자 위협 또는 외부 행위자가 있는지 프로파일링합니다.

GuardDuty RDS Protection 활성화에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 GuardDuty RDS Protection을 참조하세요.

RDS Protection이 성공하거나 실패한 로그인 시도에서의 비정상적인 패턴과 같은 잠재적 위협을 탐지하면, GuardDuty는 손상되었을지도 모르는 데이터베이스 관련 세부 정보가 포함된 새 탐지 결과를 생성합니다. Amazon GuardDuty 콘솔의 결과 요약 섹션에서 결과 세부 정보를 볼 수 있습니다. 결과 세부 정보는 결과 유형에 따라 달라집니다. 기본 세부 정보, 리소스 유형 및 리소스 역할에 따라 결과에 사용할 수 있는 정보의 종류가 결정됩니다. 일반적으로 사용할 수 있는 검색 결과 세부 정보 및 검색 유형에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서에서 Finding details(결과 세부 정보) 및 GuardDuty RDS Protection finding types(GuardDuty RDS Protection 결과 유형)를 각각 참조하세요.

RDS Protection 기능은 이 기능을 사용할 수 있는 모든 AWS 리전의 AWS 계정에 대해 켜거나 해제할 수 있습니다. RDS Protection이 활성화되지 않은 경우 GuardDuty는 손상되었을지도 모르는 Aurora 데이터베이스를 탐지하거나 보안 침해 관련 세부 정보를 제공하지 않습니다.

기존 GuardDuty 계정은 30일 평가판 기간으로 RDS Protection을 활성화할 수 있습니다. 새 GuardDuty 계정의 경우 RDS Protection이 이미 활성화되어 있으며 30일 무료 평가판 기간에 포함되어 있습니다. 자세한 내용은 Amazon GuardDuty 사용 설명서의 Estimating GuardDuty cost(GuardDuty 비용 추정)를 참조하세요.

GuardDuty가 아직 RDS Protection을 지원하지 않는 AWS 리전에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Region-specific feature availability를 참조하세요.

GuardDuty RDS 보호가 지원하는 Aurora 데이터베이스 버전에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서에 나와 있는 Supported Amazon Aurora, Amazon RDS, and Aurora Limitless databases를 참조하시기 바랍니다.

Amazon Aurora MySQL 작업

Amazon Aurora MySQL은 완전 관리형의 MySQL 호환 관계형 데이터베이스 엔진으로, 하이엔드 상용 데이터베이스의 속도와 안정성에 오픈 소스 데이터베이스의 단순성과 비용 효율성을 결합합니다. Aurora MySQL은 MySQL을 대체하는 제품으로, 새 MySQL 및 기존 MySQL 배포를 간편하고 비용 효율적으로 설정, 운영 및 확장할 수 있으므로 비즈니스 및 애플리케이션에 집중할 수 있습니다. Amazon RDS는 프로비저닝, 패치 적용, 백업, 복구, 장애 감지 및 수리와 같은 일상적인 데이터베이스 태스크를 처리하여 Aurora를 관리합니다. 또한 Amazon RDS는 기존 Amazon RDS for MySQL 애플리케이션을 Aurora MySQL로 변환하는 푸시 버튼 마이그레이션 도구를 제공합니다.

주제

Amazon Aurora MySQL의 개요

다음 단원에는 Amazon Aurora MySQL의 개요가 나와 있습니다.

주제

- Amazon Aurora MySQL 성능 개선 사항
- Amazon Aurora MySQL 및 지형 정보 데이터
- Aurora MySQL 버전 3은 MySQL 8.0과 호환
- Aurora MySQL 버전 2는 MySQL 5.7과 호환

Amazon Aurora MySQL 성능 개선 사항

Amazon Aurora에는 고급 상용 데이터베이스의 다양한 필요를 지원하는 성능 향상이 포함되어 있습니다.

빠른 입력 기능

빠른 입력 기능은 기본 키에 의해 정렬되는 병렬 입력을 빠르게 처리해 주며, 특히 `LOAD DATA` 및 `INSERT INTO ... SELECT ...` 문 사용 시 유용합니다. 빠른 입력 기능은 문을 실행할 때 인덱스 순회에서 커서의 위치를 캐싱합니다. 이에 따라 인덱스를 불필요하게 다시 순회하지 않도록 해줍니다.

빠른 입력은 Aurora MySQL 버전 3.03.2 이상에서 일반 InnoDB 테이블에 대해서만 활성화됩니다. InnoDB 임시 테이블에서는 이 최적화가 작동하지 않습니다. Aurora MySQL 버전 2에서는 2.11 및 2.12 버전 모두 비활성화되어 있습니다. 빠른 입력 최적화는 적응형 해시 인덱스 최적화가 비활성화된 경우에만 작동합니다.

다음 측정치를 모니터링하면 DB 클러스터에서 빠른 입력 기능의 효과를 확인할 수 있습니다.

- `aurora_fast_insert_cache_hits`: 캐싱된 커서가 성공적으로 검색 및 확인되면 증가하는 카운터입니다.
- `aurora_fast_insert_cache_misses`: 캐싱된 커서가 더 이상 유효하지 않고 Aurora가 정상적인 인덱스 순회를 수행하면 증가하는 카운터입니다.

다음 명령을 사용하면 빠른 입력 측정치의 현재 값을 검색할 수 있습니다.

```
mysql> show global status like 'Aurora_fast_insert%';
```

출력은 다음과 비슷합니다.

```
+---------------------------------+-----------+
| Variable_name                   | Value     |
+---------------------------------+-----------+
| Aurora_fast_insert_cache_hits   | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+---------------------------------+-----------+
```

Amazon Aurora MySQL 및 지형 정보 데이터

다음 목록은 기본 Aurora MySQL 공간 기능을 요약한 것으로서 해당 기능이 MySQL의 공간 기능에 어떻게 상응하는지 설명합니다.

- Aurora MySQL 버전 2는 MySQL 5.7과 동일한 공간 데이터 유형 및 공간 관계 기능을 지원합니다. 이러한 데이터 유형 및 기능에 대한 자세한 내용은 MySQL 5.7 설명서의 공간 데이터 유형 및 공간 관계 기능을 참조하세요.
- Aurora MySQL 버전 3은 MySQL 8.0과 동일한 공간 데이터 유형 및 공간 관계 기능을 지원합니다. 이러한 데이터 유형 및 기능에 대한 자세한 내용은 MySQL 8.0 설명서의 공간 데이터 유형 및 공간 관계 기능을 참조하세요.
- Aurora MySQL은 InnoDB 테이블에서 공간 인덱싱 기능을 지원합니다. 이 기능을 사용하면 공간 데이터에 대한 쿼리를 위해 대규모 데이터 세트에 대한 쿼리 성능을 향상시킬 수 있습니다. MySQL의 경우 MySQL 5.7과 8.0에서 InnoDB 테이블에 대한 공간 인덱싱 기능이 제공됩니다.

  Aurora MySQL은 공간 쿼리 시 성능을 높이기 위해 MySQL과는 다른 공간 인덱싱 전략을 사용합니다. Aurora 공간 인덱스 구현에서는 R-트리보다 나은 공간 범위 스캔 성능을 제공하기 위해 B-트리에서 공간 채움 곡선을 사용합니다.

> **Note**
>
> Aurora MySQL에서는 SRID(공간 참조 식별자)가 있는 열에 공간 인덱스가 정의된 테이블의 트랜잭션은 다른 트랜잭션에서 업데이트할 수 있도록 선택한 영역에 삽입할 수 없습니다.

다음 데이터 정의 언어(DDL) 문은 공간 데이터 유형을 사용하는 열에 인덱스를 생성할 목적으로 지원됩니다.

테이블 생성

`SPATIAL INDEX` 문에서 `CREATE TABLE` 키워드를 사용하여 새 테이블의 열에 공간 인덱스를 추가할 수 있습니다. 다음은 그 한 예입니다.

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

테이블 변경

SPATIAL INDEX 문에서 ALTER TABLE 키워드를 사용하여 기존 테이블의 열에 공간 인덱스를 추가할 수 있습니다. 다음은 그 한 예입니다.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

인덱스 생성

SPATIAL 문에서 CREATE INDEX 키워드를 사용하여 기존 테이블의 열에 공간 인덱스를 추가할 수 있습니다. 다음은 한 예입니다.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL 버전 3은 MySQL 8.0과 호환

Aurora MySQL 버전 3을 사용하여 최신 MySQL 호환 기능, 성능 향상 및 버그 수정을 수행할 수 있습니다. 다음에서 MySQL 8.0 호환성을 갖춘 Aurora MySQL 버전 3에 대해 배울 수 있습니다. 클러스터 및 애플리케이션을 Aurora MySQL 버전 3으로 업그레이드하는 방법을 배울 수 있습니다.

Aurora Serverless v2와 같은 일부 Aurora 기능에는 Aurora MySQL 버전 3이 필요합니다.

주제

MySQL 8.0 커뮤니티 에디션의 기능

Aurora MySQL 버전 3의 초기 릴리스는 MySQL 8.0.23 커뮤니티 에디션과 호환됩니다. MySQL 8.0에는 다음과 같은 여러 새로운 기능이 도입되었습니다.

- 원자성 데이터 정의 언어(DDL) 지원 자세한 내용은 원자성 데이터 정의 언어(DDL) 지원 단원을 참조하십시오.
- JSON 함수. 사용 정보는 MySQL 참조 설명서의 JSON 함수를 참조하세요.
- 창 함수. 사용 정보는 MySQL 참조 설명서의 Window 함수를 참조하세요.
- `WITH` 절을 사용한 공통 테이블 표현식(CTE)입니다. 사용 정보는 MySQL 참조 설명서의 공통 테이블 표현식(WITH)을 참조하세요.
- `ALTER TABLE` 문에 대해 최적화된 `ADD COLUMN` 및 `RENAME COLUMN` 절입니다. 이러한 최적화를 '인스턴트 DDL'이라고 합니다. Aurora MySQL 버전 3은 커뮤니티 MySQL 인스턴트 DDL 기능과 호환됩니다. 이전의 Aurora 빠른 DDL 기능은 사용되지 않습니다. 인스턴트 DDL의 사용 정보는 인스턴트 DDL(Aurora MySQL 버전 3) 섹션을 참조하세요.
- 내림차순 인덱스, 함수 인덱스 및 보이지 않는 인덱스. 사용 정보는 MySQL 참조 설명서의 보이지 않는 인덱스, 내림차순 인덱스 및 CREATE INDEX 문을 참조하세요.
- SQL 문을 통해 제어되는 역할 기반 권한입니다. 권한 모델 변경에 대한 자세한 내용은 역할 기반 권한 모델 섹션을 참조하세요.
- `SELECT ... FOR SHARE` 문이 포함된 `NOWAIT` 및 `SKIP LOCKED` 절입니다. 이러한 절은 다른 트랜잭션이 행 잠금을 해제할 때까지 기다리지 않습니다. 사용 정보는 MySQL 참조 설명서의 읽기 잠금을 참조하세요.
- 이진 로그(binlog) 복제 향상. Aurora MySQL의 자세한 내용은 이진 로그 복제 섹션을 참조하세요. 특히 필터링된 복제를 할 수 있습니다. 필터링된 복제에 대한 사용 정보는 MySQL 참조 설명서의 서버가 복제 필터링 규칙을 평가하는 방법을 참조하세요.
- 힌트. 일부 MySQL 8.0 호환 힌트는 이미 Aurora MySQL 버전 2로 백포팅되었습니다. Aurora MySQL에서 힌트 사용에 대한 자세한 내용은 Aurora MySQL 힌트 섹션을 참조하세요. 커뮤니티 MySQL 8.0의 전체 힌트 목록은 MySQL 참조 설명서의 최적화 프로그램 힌트를 참조하세요.

MySQL 8.0 커뮤니티 버전에 추가된 기능의 전체 목록은 블로그 게시물, MySQL 8.0의 전체 새 기능 목록을 참조하세요.

Aurora MySQL 버전 3에는 커뮤니티 MySQL 8.0.26에서 백포팅된 포괄적 언어에 대한 키워드 변경도 포함되어 있습니다. 이러한 변경 사항에 대한 자세한 내용은 Aurora MySQL 버전 3에 대한 포괄적 언어 변경 섹션을 참조하세요.

Aurora MySQL Serverless v2를 위한 Aurora MySQL 버전 3 전제 조건

Aurora MySQL 버전 3은 Aurora MySQL Serverless v2 클러스터의 모든 DB 인스턴스에 대한 전제 조건입니다. Aurora MySQL 서버리스 v2에는 DB 클러스터의 리더 인스턴스 및 Aurora MySQL 서버리스 v1에서 사용할 수 없는 기타 Aurora 기능에 대한 지원이 포함되어 있습니다. 또한 Aurora MySQL 서버리스 v1보다 빠르고 세분화된 크기 조정이 가능합니다.

Aurora MySQL 버전 3에 대한 릴리스 정보

모든 Aurora MySQL 버전 3 릴리스에 대한 릴리스 정보는 Aurora MySQL 릴리스 정보의 Amazon Aurora MySQL 버전 3에 대한 데이터베이스 엔진 업데이트를 참조하세요.

새로운 병렬 쿼리 최적화

Aurora 병렬 쿼리 최적화가 이제 SQL 작업에 더 많이 적용됩니다.

- 병렬 쿼리는 이제 768바이트보다 긴 `TEXT`, `BLOB`, `JSON`, `GEOMETRY`, `VARCHAR` 및 `CHAR`와 같은 데이터 유형이 포함된 테이블에 적용됩니다.
- 병렬 쿼리는 파티셔닝된 테이블과 관련된 쿼리를 최적화할 수 있습니다.
- 병렬 쿼리는 선택 목록 및 `HAVING` 절에서 집계 함수 호출과 관련된 쿼리를 최적화할 수 있습니다.

이런 향상에 대한 자세한 내용은 Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드 섹션을 참조하세요. Aurora 병렬 쿼리에 대한 일반적인 정보는 Amazon Aurora MySQL용 병렬 쿼리 섹션을 참조하세요.

데이터베이스 재시작 시간 단축을 위한 최적화

Aurora MySQL DB 클러스터는 계획된 운영 중단과 예기치 않은 운영 중단 모두에서 가용성이 높아야 합니다.

데이터베이스 관리자는 때때로 데이터베이스 유지 관리를 수행해야 합니다. 이 유지 관리에는 데이터베이스 패치 적용, 업그레이드, 수동 재부팅이 필요한 데이터베이스 파라미터 수정, 인스턴스 클래스 변경에 걸리는 시간을 줄이기 위한 장애 조치 수행 등이 포함됩니다. 이러한 계획된 작업에는 가중 중지가 필요합니다.

그러나 가동 중지는 기본 하드웨어 장애로 인한 예상치 못한 장애 조치 또는 데이터베이스 리소스 제한과 같은 예기치 않은 작업으로 인해 발생할 수도 있습니다. 이러한 모든 계획된 작업과 예기치 않은 작업으로 인해 데이터베이스가 다시 시작됩니다.

Aurora MySQL 버전 3.05 이상에서 데이터베이스 재시작 시간을 단축하는 최적화 기능을 도입했습니다. 이러한 최적화를 통해 최적화를 사용하지 않을 때보다 가동 중지 시간이 최대 65% 줄어들고 재시작 후 데이터베이스 워크로드가 중단되는 일도 줄어듭니다.

데이터베이스를 시작하는 동안 많은 내부 메모리 구성 요소가 초기화됩니다. 이 중 가장 큰 것은 InnoDB 버퍼 풀로, Aurora MySQL에서는 기본적으로 인스턴스 메모리 크기의 75%입니다. 테스트 결과, 초기화 시간은 InnoDB 버퍼 풀의 크기에 비례하므로 DB 인스턴스 클래스 크기에 따라 조정되는 것으로 나타났습니다. 이 초기화 단계에서는 데이터베이스가 연결을 수락할 수 없어 재시작 시 가동 중지가 길어집니다. Aurora MySQL 고속 재시작의 첫 번째 단계는 버퍼 풀 초기화를 최적화하여 데이터베이스 초기화 시간을 줄임으로써 전체 재시작 시간을 줄입니다.

자세한 내용은 Aurora MySQL 최적화, 데이터베이스 재시작 시간 단축 블로그를 참조하세요.

Aurora MySQL 버전 3의 새로운 임시 테이블 동작

Aurora MySQL 버전 3에서는 이전 Aurora MySQL 버전과 다르게 임시 테이블을 처리합니다. 이 새로운 동작은 MySQL 8.0 커뮤니티 에디션에서 상속되었습니다. Aurora MySQL 버전 3에서 생성할 수 있는 임시 테이블에는 다음과 같이 2가지 유형이 있습니다.

- 내부(또는 암묵적) 임시 테이블 – Aurora MySQL 엔진이 집계 정렬, 파생 테이블 또는 공통 테이블 표현식(CTE)과 같은 작업을 처리하기 위해 생성합니다.
- 사용자 생성(또는 명시적) 임시 테이블 - `CREATE TEMPORARY TABLE` 문을 사용할 때 Aurora MySQL 엔진에서 생성합니다.

Aurora 리더 DB 인스턴스의 내부 및 사용자 생성 임시 테이블 모두에 대해 추가로 고려해야 할 점이 있습니다. 다음 섹션에서 이러한 변경 사항에 대해 살펴봅니다.

주제

내부(암묵적) 임시 테이블에 대한 스토리지 엔진

중간 결과 세트를 생성할 때 Aurora MySQL은 처음에 메모리 내 임시 테이블에 쓰기를 시도합니다. 호환되지 않는 데이터 유형이나 구성된 제한으로 인해 실패할 수 있습니다. 그렇다면 임시 테이블은 메모리에 보관되지 않고 온디스크 임시 테이블로 변환됩니다. 이에 대한 자세한 내용은 MySQL 설명서의 MySQL의 내부 임시 테이블 사용에서 확인할 수 있습니다.

Aurora MySQL 버전 3에서는 내부 임시 테이블이 작동하는 방식이 이전 Aurora MySQL 버전과 다릅니다. 이러한 임시 테이블에 대해 InnoDB와 MyISAM 스토리지 엔진 중에서 선택하지 않고 이제 `TempTable` 및 `MEMORY` 스토리지 엔진 중에서 선택합니다.

`TempTable` 스토리지 엔진을 사용하면 특정 데이터를 처리하는 방법을 추가로 선택할 수 있습니다. 영향을 받은 데이터는 DB 인스턴스의 모든 내부 임시 테이블을 저장하는 메모리 풀을 오버플로합니다.

예를 들어 대형 테이블에 `GROUP BY`와 같은 집계를 수행하는 동안 이러한 선택은 많은 양의 임시 데이터를 생성하는 쿼리의 성능에 영향을 줄 수 있습니다.

> **Tip**
>
> 워크로드에 내부 임시 테이블을 생성하는 쿼리가 포함되어 있는 경우 벤치마크를 실행하고 성능 관련 지표를 모니터링하여 애플리케이션이 이 변경으로 수행하는 방법을 확인합니다.
> 경우에 따라 임시 데이터의 양을 `TempTable` 메모리 풀 내에 맞추거나 적은 양으로 메모리 풀만 오버플로합니다. 이러한 경우 내부 임시 테이블 및 메모리 매핑 파일에 대한 `TempTable` 설정을 사용하여 오버플로 데이터를 저장하는 것이 좋습니다. 이게 기본 설정입니다.

`TempTable` 스토리지 엔진이 기본값입니다. `TempTable`에서는 테이블당 최대 메모리 제한 대신 이 엔진을 사용하는 모든 임시 테이블에 대한 공통 메모리 풀을 사용합니다. 이 메모리 풀의 크기는 temptable_max_ram 파라미터에서 지정합니다. 메모리가 16GiB 이상인 DB 인스턴스에서는 1GiB, 메모리가 16GiB 미만인 DB 인스턴스에서는 16MB가 기본값으로 설정됩니다. 메모리 풀의 크기는 세션 수준 메모리 소비에 영향을 줍니다.

경우에 따라 `TempTable` 스토리지 엔진을 사용할 때 임시 데이터가 메모리 풀의 크기를 초과할 수 있습니다. 그렇다면 Aurora MySQL은 보조 메커니즘을 사용하여 오버플로 데이터를 저장합니다.

temptable_max_mmap 파라미터를 설정하여 데이터가 메모리 매핑된 임시 파일에 오버플로될지, 디스크의 InnoDB 내부 임시 테이블에 오버플로될지 선택할 수 있습니다. 이러한 오버플로 메커니즘의 다양한 데이터 형식과 오버플로 기준은 쿼리 성능에 영향을 줄 수 있습니다. 이렇게 하려면 디스크에 기록되는 데이터의 양과 디스크 스토리지 처리량에 대한 수요에 영향을 주면 됩니다.

Aurora MySQL 버전 3은 다음과 같은 방식으로 오버플로 데이터를 저장합니다.

- 라이터 DB 인스턴스에서 InnoDB 내부 임시 테이블 또는 메모리 매핑된 임시 파일로 오버플로되는 데이터는 인스턴스의 로컬 스토리지에 상주합니다.
- 리더 DB 인스턴스에서 오버플로 데이터는 항상 로컬 스토리지의 메모리 매핑된 임시 파일에 상주합니다.

  읽기 전용 인스턴스는 Aurora 클러스터 볼륨에 데이터를 저장할 수 없습니다.

내부 임시 테이블과 관련된 구성 파라미터는 클러스터의 라이터 및 리더 인스턴스와 다르게 적용됩니다.

- 리더 인스턴스의 경우 Aurora MySQL은 항상 `TempTable` 스토리지 엔진을 사용합니다.
- DB 인스턴스 메모리 크기에 관계없이 라이터 인스턴스와 리더 인스턴스 모두에 대해 `temptable_max_mmap`의 기본값 크기는 1GiB입니다. 라이터 및 리더 인스턴스 둘 다에서 이 값을 조정할 수 있습니다.
- `temptable_max_mmap`을 `0`으로 설정하면 라이터 인스턴스에서 메모리 매핑된 임시 파일 사용이 비활성화됩니다.
- 리더 인스턴스에서는 `temptable_max_mmap`을 `0`으로 설정할 수 없습니다.

> **Note**
>
> temptable_use_mmap 파라미터를 사용하지 않는 것이 좋습니다. 이 파라미터는 사용이 중단되었으며 향후 MySQL 릴리스에서 지원이 제거될 예정입니다.

내부 인 메모리 임시 테이블의 크기 제한

내부(암묵적) 임시 테이블에 대한 스토리지 엔진에서 설명한 바와 같이 temptable_max_ram 및 temptable_max_mmap 설정을 사용하여 임시 테이블 리소스를 전역적으로 제어할 수 있습니다.

또한 tmp_table_size DB 파라미터를 사용하여 개별 내부 인 메모리 임시 테이블의 크기를 제한할 수 있습니다. 이 제한은 개별 쿼리가 과도한 양의 글로벌 임시 테이블 리소스를 소비하는 것을 방지하기 위한 것이며, 이러한 리소스가 필요한 동시 쿼리의 성능에 영향을 미칠 수 있습니다.

`tmp_table_size` 파라미터는 Aurora MySQL 버전 3의 `MEMORY` 스토리지 엔진이 생성한 임시 테이블의 최대 크기를 정의합니다.

Aurora MySQL 버전 3.04 이상에서는 `tmp_table_size`가 `aurora_tmptable_enable_per_table_limit` DB 파라미터가 ON으로 설정되었을 때 TempTable 스토리지 엔진이 생성한 임시 테이블의 최대 크기도 정의합니다. 이 동작은 기본적으로 비활성화(OFF)되며, 이는 Aurora MySQL 버전 3.03 이하 버전에서와 동일합니다.

- `aurora_tmptable_enable_per_table_limit`이 OFF로 설정되었을 때 `tmp_table_size`는 TempTable 스토리지 엔진이 생성한 내부 인 메모리 임시 테이블에 고려되지 않습니다.

  그러나 글로벌 TempTable 리소스 제한은 계속 적용됩니다. Aurora MySQL은 글로벌 TempTable 리소스 제한에 도달했을 때 다음과 같이 동작합니다.

  - 라이터 DB 인스턴스 - Aurora MySQL이 인 메모리 임시 테이블을 InnoDB 온 디스크 임시 테이블로 자동 변환합니다.
  - 리더 DB 인스턴스 - 쿼리가 오류와 함께 종료됩니다.

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

- `aurora_tmptable_enable_per_table_limit`이 ON인 경우 Aurora MySQL은 `tmp_table_size` 제한에 도달했을 때 다음과 같이 동작합니다.

  - 라이터 DB 인스턴스 - Aurora MySQL이 인 메모리 임시 테이블을 InnoDB 온 디스크 임시 테이블로 자동 변환합니다.
  - 리더 DB 인스턴스 - 쿼리가 오류와 함께 종료됩니다.

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

  이 경우 글로벌 TempTable 리소스 제한과 테이블당 제한이 모두 적용됩니다.

> **Note**
>
> `aurora_tmptable_enable_per_table_limit` 파라미터는 [internal_tmp_mem_storage_engine]이 MEMORY로 설정된 경우 아무런 효과가 없습니다. 이 경우 인 메모리 임시 테이블의 최대 크기는 [tmp_table_size] 또는 [max_heap_table_size] 값 중 더 작은 것으로 정의됩니다.

다음 예시는 라이터 및 리더 DB 인스턴스에 대한 `aurora_tmptable_enable_per_table_limit` 파라미터의 동작을 보여줍니다.

Example **aurora_tmptable_enable_per_table_limit**이 **OFF**로 설정된 경우 라이터 DB 인스턴스

인 메모리 임시 테이블이 InnoDB 온 디스크 임시 테이블로 변환되지 않습니다.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+------------------------------------------
+---------------------+---------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+------------------------------------------
+---------------------+---------------------+
|                  0 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+------------------------------------------
+---------------------+---------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
```

```
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit**이 **ON**으로 설정된 경우 라이터 DB 인스턴스

인 메모리 임시 테이블이 InnoDB 온 디스크 임시 테이블로 변환됩니다.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+------------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+------------------------------------------
+------------------+
|                  0 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+------------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
```

```
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit**이 **OFF**로 설정된 경우 리더 DB 인스턴스

쿼리가 오류 없이 완료되는 이유는 tmp_table_size가 적용되지 않으며 글로벌 TempTable 리소스 제한에 도달하지 않았기 때문입니다.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+------------------------------------------
+--------------------+---------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+------------------------------------------
+--------------------+---------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+------------------------------------------
+--------------------+---------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
```

```
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example **aurora_tmptable_enable_per_table_limit**이 **OFF**로 설정된 경우 리더 DB 인스턴스

이 쿼리는 aurora_tmptable_enable_per_table_limit이 OFF로 설정된 경우 글로벌 TempTable 리소스 제한에 도달합니다. 리더 인스턴스에 오류가 있는 상태로 쿼리가 종료됩니다.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+------------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+------------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+------------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example **aurora_tmptable_enable_per_table_limit**이 **ON**로 설정된 경우 리더 DB 인스턴스

tmp_table_size 제한에 도달하면 오류와 함께 쿼리가 종료됩니다.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
```

```
+---------------------+------------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+---------------------+------------------+-----------------------------------------
+------------------+
|                   1 | 3.04.0           |                                        1 |
      16777216 |
+---------------------+------------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

Aurora 복제본의 내부 임시 테이블에 대한 완전성 문제 완화

임시 테이블의 크기 제한 문제를 방지하려면 temptable_max_ram 및 temptable_max_mmap 파라미터를 워크로드의 요구 사항에 맞출 수 있는 결합된 값으로 설정합니다.

temptable_max_ram 파라미터 값을 설정할 때는 주의해야 합니다. 값을 너무 높게 설정하면 데이터베이스 인스턴스에서 사용 가능한 메모리가 줄어들어 메모리 부족 상태가 발생할 수 있습니다. DB 인스턴스의 평균 여유 메모리를 모니터링합니다. 그런 다음, temptable_max_ram에 대한 적절한 값을 결정하여 인스턴스에 합리적인 수준의 여유 메모리가 계속해서 남아 있도록 합니다. 자세한 내용은 Amazon Aurora의 여유 메모리 부족 단원을 참조하십시오.

로컬 스토리지의 크기와 임시 테이블 공간 사용량을 모니터링하는 것도 중요합니다. FreeLocalStorage에 설명된 Amazon Aurora에 대한 Amazon CloudWatch 지표 Amazon CloudWatch 지표를 사용하여 특정 DB 인스턴스에 사용할 수 있는 임시 스토리지를 모니터링할 수 있습니다.

> **Note**
>
> 이 프로시저는 aurora_tmptable_enable_per_table_limit 파라미터가 ON으로 설정된 경우에는 작동하지 않습니다. 자세한 내용은 내부 인 메모리 임시 테이블의 크기 제한 단원을 참조하십시오.

Example 1

임시 테이블의 누적 크기가 20GiB까지 늘어난다는 점은 알고 있지만, 메모리 내 임시 테이블을 2GiB로 설정하고 디스크에서 최대 20GiB까지 확장하려고 합니다.

이 경우 `temptable_max_ram`을 **`2,147,483,648`**로, `temptable_max_mmap`를 **`21,474,836,480`**으로 설정합니다. 이러한 값은 바이트 단위입니다.

이러한 파라미터 설정을 사용하면 임시 테이블을 누적해서 총 22GiB로 확장할 수 있습니다.

Example 2

현재 인스턴스 크기는 16xlarge 이상이고, 필요한 임시 테이블의 전체 크기를 알 수 없습니다. 메모리에서 최대 4GiB를, 디스크에서 사용 가능한 최대 스토리지 크기를 사용할 수 있기를 원합니다.

이 경우 `temptable_max_ram`을 **`4,294,967,296`**으로, `temptable_max_mmap`를 **`1,099,511,627,776`**으로 설정합니다. 이러한 값은 바이트 단위입니다.

여기서 `temptable_max_mmap`을 16xlarge Aurora DB 인스턴스의 최대 로컬 스토리지인 1.2TiB보다 작은 1TiB로 설정합니다.

인스턴스 크기가 이보다 작은 경우 사용 가능한 로컬 스토리지를 채우지 않도록 `temptable_max_mmap`의 값을 조정합니다. 예를 들어, 2xlarge 인스턴스에는 160GiB의 로컬 스토리지만 사용할 수 있습니다. 이 경우 값을 160GiB 미만으로 설정하는 것이 좋습니다. DB 인스턴스 크기에 사용할 수 있는 로컬 스토리지에 대한 자세한 내용은 [Aurora MySQL에 대한 임시 스토리지 한도](#) 섹션을 참조하세요.

Aurora MySQL DB 인스턴스에서 temptable_max_mmap 파라미터 최적화

Aurora MySQL의 `temptable_max_mmap` 파라미터는 온디스크 InnoDB 임시 테이블로 오버플로하거나(라이터 DB 인스턴스) 오류를 발생시키기(리더 DB 인스턴스) 전에 메모리 매핑 파일에서 사용할 수 있는 최대 로컬 디스크 공간을 제어합니다. 이 DB 인스턴스 파라미터를 올바르게 설정하면 DB 인스턴스의 성능을 최적화하는 데 도움이 될 수 있습니다.

사전 조건

1. 성능 스키마가 활성화되어 있는지 확인합니다. 다음 SQL 명령을 실행하여 확인할 수 있습니다.

```
SELECT @@performance_schema;
```

출력 값이 `1`이면 활성화되었음을 나타냅니다.

2. 임시 테이블 메모리 계측이 활성화되어 있는지 확인합니다. 다음 SQL 명령을 실행하여 확인할 수 있습니다.

```
SELECT name, enabled FROM performance_schema.setup_instruments WHERE name LIKE
 '%memory%temptable%';
```

   enabled 열에는 관련 임시 테이블 메모리 계측 항목에 대해 YES가 표시됩니다.

임시 테이블 사용량 모니터링

temptable_max_mmap의 초기 값을 설정할 때는 사용 중인 DB 인스턴스 클래스의 로컬 스토리지 크기의 80%로 시작하는 것이 좋습니다. 이렇게 하면 임시 테이블이 효율적으로 작동할 수 있는 충분한 디스크 공간을 확보하는 동시에 인스턴스에서 다른 디스크를 사용할 수 있는 공간을 확보할 수 있습니다.

DB 인스턴스 클래스의 로컬 스토리지 크기를 찾으려면 Aurora MySQL에 대한 임시 스토리지 한도 섹션을 참조하세요.

예를 들어 db.r5.large DB 인스턴스 클래스를 사용하는 경우 로컬 스토리지 크기는 32GiB입니다. 이 경우 처음에 temptable_max_mmap 파라미터를 32GiB의 80%, 즉 25.6GiB로 설정합니다.

초기 temptable_max_mmap 값을 설정한 후 Aurora MySQL 인스턴스에서 피크 워크로드를 실행합니다. 다음 SQL 쿼리를 사용하여 현재 및 높은 임시 테이블 디스크 사용량을 모니터링합니다.

```
SELECT event_name, current_count, current_alloc, current_avg_alloc, high_count,
 high_alloc, high_avg_alloc
FROM sys.memory_global_by_current_bytes WHERE event_name LIKE 'memory/temptable/%';
```

이 쿼리는 다음 정보를 검색합니다.

- event_name - 임시 테이블 메모리 또는 디스크 사용 이벤트의 이름입니다.
- current_count - 할당된 현재 임시 테이블 메모리 또는 디스크 블록 수입니다.
- current_alloc - 임시 테이블에 할당된 현재 메모리 또는 디스크의 양입니다.
- current_avg_alloc - 임시 테이블 메모리 또는 디스크 블록의 현재 평균 크기입니다.
- high_count - 할당된 임시 테이블 메모리 또는 디스크 블록의 가장 많은 수입니다.
- high_alloc - 임시 테이블에 할당된 메모리 또는 디스크의 가장 많은 양입니다.
- high_avg_alloc - 임시 테이블 메모리 또는 디스크 블록의 가장 큰 평균 크기입니다.

이 설정을 사용하여 테이블이 가득 참 오류와 함께 쿼리가 실패할 경우 임시 테이블 작업을 위해 워크로드에 더 많은 디스크 공간이 필요함을 나타냅니다. 이 경우 로컬 스토리지 공간이 더 많은 인스턴스 크기로 DB 인스턴스 크기를 늘리는 것이 좋습니다.

최적의 **temptable_max_mmap** 값 설정

다음 프로시저에 따라 temptable_max_mmap 파라미터에 적합한 크기를 모니터링하고 설정합니다.

1. 이전 쿼리의 출력을 검토하고 high_alloc 열에 표시된 대로 임시 테이블 디스크 피크 사용량을 식별합니다.
2. 임시 테이블 디스크 피크 사용량에 따라 Aurora MySQL DB 인스턴스의 DB 파라미터 그룹에서 temptable_max_mmap 파라미터를 조정합니다.

   향후 성장을 수용하기 위해 값을 임시 테이블 디스크 피크 사용량보다 약간 높게 설정합니다.
3. DB 인스턴스에 파라미터 그룹 변경 사항을 적용합니다.
4. 피크 워크로드 중에 임시 테이블 디스크 사용량을 다시 모니터링하여 새 temptable_max_mmap 값이 적절한지 확인합니다.
5. 필요에 따라 이전 단계를 반복하여 temptable_max_mmap 파라미터를 미세 조정합니다.

리더 DB 인스턴스의 사용자 생성(명시적) 임시 테이블

CREATE TABLE 문에서 TEMPORARY 키워드를 사용하여 명시적 임시 테이블을 생성할 수 있습니다. Aurora DB 클러스터의 라이터 DB 인스턴스에서 명시적 임시 테이블이 지원됩니다. 리더 DB 인스턴스에서 명시적 임시 테이블을 사용할 수도 있지만, 이 테이블에서 InnoDB 스토리지 엔진을 사용할 수 없습니다.

Aurora 리더 DB 인스턴스에서 명시적 임시 테이블을 생성하는 동안 오류가 발생하지 않게 하려면 모든 CREATE TEMPORARY TABLE 문을 다음 두 방법 중 하나 또는 둘 다를 이용해 실행해야 합니다.

- ENGINE=InnoDB 절은 지정하지 마십시오.
- SQL 모드를 NO_ENGINE_SUBSTITUTION으로 설정하지 마십시오.

임시 테이블 생성 오류 및 완화

수신하는 오류는 일반 CREATE TEMPORARY TABLE 문 또는 변형 CREATE TEMPORARY TABLE AS SELECT를 사용하는지에 따라 다릅니다. 다음 예에서는 서로 다른 형식의 오류를 보여줍니다.

이 임시 테이블 동작은 읽기 전용 인스턴스에만 적용됩니다. 이 첫 번째 예에서는 세션이 연결된 인스턴스의 종류임을 확인합니다.

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

일반 CREATE TEMPORARY TABLE 문에서 명령문은 NO_ENGINE_SUBSTITUTION SQL 모드가 켜져 있는 경우 실패합니다. NO_ENGINE_SUBSTITUTION이 해제되면(기본값) 적절한 엔진 교체가 이루어지며, 임시 테이블 생성이 성공합니다.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

CREATE TEMPORARY TABLE AS SELECT 문에서 명령문은 NO_ENGINE_SUBSTITUTION SQL 모드가 켜져 있는 경우 실패합니다. NO_ENGINE_SUBSTITUTION이 해제되면(기본값) 적절한 엔진 교체가 이루어지며, 임시 테이블 생성이 성공합니다.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
```

```
+--------+--------------------------------------------------------------+
| Table | Create Table                                                 |
+--------+--------------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------+--------------------------------------------------------------+
1 row in set (0.00 sec)
```

Aurora MySQL 버전 3에서 임시 테이블의 스토리지 측면 및 성능 영향에 대한 자세한 내용은 Amazon RDS for MySQL 및 Amazon Aurora MySQL의 TempTable 스토리지 엔진 사용 블로그 게시물을 참조하세요.

Aurora MySQL 버전 2와 Aurora MySQL 버전 3의 비교

Aurora MySQL 버전 2 클러스터를 버전 3으로 업그레이드하는 경우 주의해야 할 변경 사항에 대해 알아보려면 다음을 참조하세요.

주제

원자성 데이터 정의 언어(DDL) 지원

MySQL 5.7에서 8.0으로의 가장 큰 변경 사항 중 하나는 원자성 데이터 딕셔너리의 도입입니다. MySQL 8.0 이전에 MySQL 데이터 딕셔너리는 파일 기반 접근 방식을 사용하여 테이블 정의(.frm), 트리거(.trg) 및 함수와 같은 메타데이터를 스토리지 엔진의 메타데이터(예: InnoDB)와 별도로 저장했습니다. 이로 인해 DDL 작업 중에 예기치 않은 일이 발생하여 파일 기반 및 스토리지 엔진 메타데이터가 동기화되지 않을 경우 테이블이 '분리'될 위험을 비롯한 몇 가지 문제가 발생했습니다.

이를 해결하기 위해 MySQL 8.0은 모든 메타데이터를 `mysql` 스키마의 내부 InnoDB 테이블 세트에 저장하는 원자성 데이터 딕셔너리를 도입했습니다. 이 새로운 아키텍처는 트랜잭션 ACID 규정을 준수하

는 방식으로 데이터베이스 메타데이터를 관리하여 이전 파일 기반 접근 방식의 '원자성 DDL' 문제를 해결합니다. 원자성 데이터 딕셔너리에 대한 자세한 내용은 MySQL 참조 설명서의 파일 기반 메타데이터 스토리지 제거 및 원자성 데이터 정의 문 지원을 참조하세요.

이러한 아키텍처 변경으로 인해 Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 때 다음을 고려해야 합니다.

- 버전 3으로 업그레이드하는 동안 버전 2의 파일 기반 메타데이터를 새 데이터 딕셔너리 테이블로 마이그레이션해야 합니다. 마이그레이션되는 데이터베이스 객체 수에 따라 시간이 다소 걸릴 수 있습니다.
- 또한 이 변경 사항으로 인해 MySQL 5.7에서 8.0으로 업그레이드하기 전에 해결해야 할 몇 가지 새로운 비호환성도 발생했습니다. 예를 들어, 8.0에는 기존 데이터베이스 객체 이름과 충돌할 수 있는 일부 새로운 예약 키워드가 있습니다.

엔진을 업그레이드하기 전에 이러한 비호환성을 식별하는 데 도움이 되도록 Aurora MySQL은 일련의 업그레이드 호환성 검사(사전 확인)를 실행하여 데이터 딕셔너리 업그레이드를 수행하기 전에 데이터베이스 딕셔너리에 비호환 객체가 있는지 확인합니다. 사전 확인에 대한 자세한 설명은 Aurora MySQL에 대한 메이저 버전 업그레이드 사전 확인 섹션을 참조하세요.

Aurora MySQL 버전 2와 3의 기능 차이점

다음 Amazon Aurora MySQL 기능은 Aurora MySQL for MySQL 5.7에서 지원되지만, 이러한 기능은 현재 Aurora MySQL for MySQL 8.0에서 지원되지 않습니다.

- Aurora Serverless v1 클러스터를 위한 Aurora MySQL 버전 3을 사용할 수 없습니다. Aurora MySQL 버전 3은 Aurora Serverless v2와 호환됩니다.
- 랩 모드는 Aurora MySQL 버전 3에는 적용되지 않습니다. Aurora MySQL 버전 3에는 랩 모드 기능이 없습니다. 인스턴트 DDL은 이전에 랩 모드에서 사용 가능했던 빠른 온라인 DDL 기능을 대체합니다. 예시는 인스턴트 DDL(Aurora MySQL 버전 3)을 확인하세요.
- 쿼리 캐시는 커뮤니티 MySQL 8.0 및 Aurora MySQL 버전 3에서도 제거됩니다.
- Aurora MySQL 버전 3은 커뮤니티 MySQL 해시 조인 기능과 호환됩니다. Aurora MySQL 버전 2에서 해시 조인의 Aurora별 구현은 사용되지 않습니다. Aurora 병렬 쿼리에서 해시 조인 사용에 대한 자세한 내용은 병렬 쿼리 클러스터에 대한 해시 조인 설정 및 Aurora MySQL 힌트 섹션을 참조하세요. 해시 조인에 대한 일반적인 사용 정보는 MySQL 참조 설명서의 해시 조인 최적화를 참조하세요.
- Aurora MySQL 버전 2에서 더 이상 사용되지 않는 `mysql.lambda_async` 저장 프로시저는 버전 3에서 제거됩니다. 버전 3의 경우 대신 비동기 함수 `lambda_async`을 사용합니다.

- Aurora MySQL 버전 3의 기본 문자 집합은 `utf8mb4`입니다. Aurora MySQL 버전 2의 기본 문자 집합은 `latin1`이었습니다. 이 문자 집합에 대한 자세한 내용은 MySQL 참조 설명서의 utf8mb4 문자 집합(4바이트 UTF-8 유니코드 인코딩)을 참조하세요.

일부 Aurora MySQL 기능은 AWS 리전 및 DB 엔진 버전의 특정 조합에서 사용할 수 있습니다. 세부 정보는 Amazon Aurora에서 AWS 리전 및 Aurora DB 엔진별 지원 기능을 참조하세요.

인스턴스 클래스 지원

Aurora MySQL 버전 3은 Aurora MySQL 버전 2와 다른 인스턴스 클래스 집합을 지원합니다.

- 대형 인스턴스의 경우 `db.r5`, `db.r6g` 및 `db.x2g`와 같은 최신 인스턴스 클래스를 사용할 수 있습니다.
- 소형 인스턴스의 경우 `db.t3` 및 `db.t4g`와 같은 최신 인스턴스 클래스를 사용할 수 있습니다.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 개발 및 테스트에 T 인스턴스 클래스 사용 섹션을 참조하세요.

Aurora MySQL 버전 2의 다음 인스턴스 클래스는 Aurora MySQL 버전 3에서 사용할 수 없습니다.

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Aurora MySQL DB 인스턴스를 생성하는 CLI 문을 관리 스크립트에서 확인하세요. Aurora MySQL 버전 3에서 사용할 수 없는 인스턴스 클래스 이름을 하드코딩합니다. 필요한 경우 인스턴스 클래스 이름을 Aurora MySQL 버전 3에서 지원하는 이름으로 수정합니다.

> **Tip**
>
> Aurora MySQL 버전 및 AWS 리전의 특정 조합에 사용할 수 있는 인스턴스 클래스를 확인하려면 `describe-orderable-db-instance-options` AWS CLI 명령을 사용하세요.

Aurora 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

Aurora MySQL 버전 3에 대한 파라미터 변경 사항

Aurora MySQL 버전 3에는 새로운 클러스터 수준 및 인스턴스 수준 구성 파라미터가 포함되어 있습니다. Aurora MySQL 버전 3은 또한 Aurora MySQL 버전 2에 있던 일부 파라미터를 제거합니다. 일부 파라미터 이름은 포괄적 언어에 대한 이니셔티브의 결과로 변경됩니다. 이전 버전과의 호환성을 위해 이전 이름이나 새 이름을 사용하여 파라미터 값을 검색할 수 있습니다. 그러나 새 이름을 사용하여 사용자 지정 파라미터 그룹에서 파라미터 값을 지정해야 합니다.

Aurora MySQL 버전 3에서는 `lower_case_table_names` 파라미터가 클러스터가 생성될 때 영구적으로 설정됩니다. 이 옵션에 기본값이 아닌 값을 사용하는 경우 업그레이드하기 전에 Aurora MySQL 버전 3 사용자 지정 파라미터 그룹을 설정합니다. 그런 다음, 클러스터 생성 또는 스냅샷 복원 작업 중에 파라미터 그룹을 지정합니다.

> **Note**
>
> Aurora MySQL 기반 Aurora Global Database 사용 시 `lower_case_table_names` 파라미터가 활성화 되어 있는 경우 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수 없습니다. 대신 스냅샷 복원 방법을 사용하세요.

Aurora MySQL 버전 3에서는 `CONNECTION_ADMIN` 권한이 있는 사용자에게 `init_connect` 및 `read_only` 파라미터가 적용되지 않습니다. Aurora 마스터 사용자도 마찬가지입니다. 자세한 내용은 역할 기반 권한 모델 단원을 참조하십시오.

Aurora MySQL 클러스터 파라미터 전체 목록은 클러스터 수준 파라미터 섹션을 참조하세요. 이 테이블에서는 Aurora MySQL 버전 2 및 3의 모든 파라미터를 다룹니다. 이 테이블에는 Aurora MySQL 버전 3에서 새로운 파라미터 또는 Aurora MySQL 버전 3에서 제거된 파라미터를 보여주는 메모가 포함되어 있습니다.

Aurora MySQL 인스턴스 파라미터 전체 목록은 인스턴스 수준 파라미터 섹션을 참조하세요. 이 테이블에서는 Aurora MySQL 버전 2 및 3의 모든 파라미터를 다룹니다. 이 테이블에는 Aurora MySQL 버전 3에서 새로운 파라미터 또는 Aurora MySQL 버전 3에서 제거된 파라미터를 보여주는 메모가 포함되어 있습니다. 또한 이전 버전에서는 수정할 수 있었지만 Aurora MySQL 버전 3에서는 수정할 수 없는 파라미터를 보여주는 메모도 포함되어 있습니다.

변경된 파라미터 이름에 대한 자세한 내용은 Aurora MySQL 버전 3에 대한 포괄적 언어 변경 섹션을 참조하세요.

상태 변수

Aurora MySQL에 적용되지 않는 상태 변수에 대한 자세한 내용은 Aurora MySQL에 적용되지 않는 MySQL 상태 변수 섹션을 참조하세요.

Aurora MySQL 버전 3에 대한 포괄적 언어 변경

Aurora MySQL 버전 3은 MySQL 커뮤니티 에디션의 버전 8.0.23과 호환됩니다. Aurora MySQL 버전 3에는 포괄적 언어에 대한 키워드 및 시스템 스키마와 관련된 MySQL 8.0.26의 변경도 포함되어 있습니다. 예를 들어 `SHOW REPLICA STATUS` 명령이 이제 `SHOW SLAVE STATUS` 대신 기본 설정됩니다.

다음 Amazon CloudWatch 지표는 Aurora MySQL 버전 3에서 새 이름을 갖습니다.

Aurora MySQL 버전 3에서는 새 지표 이름만 사용할 수 있습니다. Aurora MySQL 버전 3으로 업그레이드할 때 지표 이름에 의존하는 경보 또는 기타 자동화를 업데이트해야 합니다.

이전 이름	새 이름	
ForwardingMasterDMLLatency	ForwardingWriterDMLLatency	
ForwardingMasterOpenSessions	ForwardingWriterOpenSessions	
AuroraDMLRejectedMasterFull	AuroraDMLRejectedWriterFull	
ForwardingMasterDMLThroughput	ForwardingWriterDMLThroughput	

다음 상태 변수는 Aurora MySQL 버전 3에서 새 이름을 갖습니다.

호환성을 위해 초기 Aurora MySQL 버전 3 릴리스에서 두 이름 중 하나를 사용할 수 있습니다. 향후 릴리스에서 기존 상태 변수 이름이 제거되게 됩니다.

제거할 이름	새 이름 또는 기본 이름	
Aurora_fwd_master_dml_stmt_duration	Aurora_fwd_writer_dml_stmt_duration	

제거할 이름	새 이름 또는 기본 이름
Aurora_fwd_master_dml_stmt_count	Aurora_fwd_writer_dml_stmt_count
Aurora_fwd_master_select_stmt_duration	Aurora_fwd_writer_select_stmt_duration
Aurora_fwd_master_select_stmt_count	Aurora_fwd_writer_select_stmt_count
Aurora_fwd_master_errors_session_timeout	Aurora_fwd_writer_errors_session_timeout
Aurora_fwd_master_open_sessions	Aurora_fwd_writer_open_sessions
Aurora_fwd_master_errors_session_limit	Aurora_fwd_writer_errors_session_limit
Aurora_fwd_master_errors_rpc_timeout	Aurora_fwd_writer_errors_rpc_timeout

다음 구성 파라미터는 Aurora MySQL 버전 3에서 새 이름을 갖습니다.

호환성을 위해 초기 Aurora MySQL 버전 3 릴리스에서 두 이름 중 하나를 사용하여 mysql에서 파라미터 값을 확인할 수 있습니다. 사용자 정의 파라미터 그룹의 값을 수정할 때는 새 이름만 사용 수 있습니다. 향후 릴리스에서 기존 파라미터 이름이 제거되게 됩니다.

제거할 이름	새 이름 또는 기본 이름
aurora_fwd_master_idle_timeout	aurora_fwd_writer_idle_timeout

제거할 이름	새 이름 또는 기본 이름
aurora_fwd_master_max_connections_pct	aurora_fwd_writer_max_connections_pct
master_verify_checksum	source_verify_checksum
sync_master_info	sync_source_info
init_slave	init_replica
rpl_stop_slave_timeout	rpl_stop_replica_timeout
log_slow_slave_statements	log_slow_replica_statements
slave_max_allowed_packet	replica_max_allowed_packet
slave_compressed_protocol	replica_compressed_protocol
slave_exec_mode	replica_exec_mode
slave_type_conversions	replica_type_conversions
slave_sql_verify_checksum	replica_sql_verify_checksum
slave_parallel_type	replica_parallel_type
slave_preserve_commit_order	replica_preserve_commit_order
log_slave_updates	log_replica_updates

제거할 이름	새 이름 또는 기본 이름
slave_allow_batching	replica_allow_batching
slave_load_tmpdir	replica_load_tmpdir
slave_net_timeout	replica_net_timeout
sql_slave_skip_counter	sql_replica_skip_counter
slave_skip_errors	replica_skip_errors
slave_checkpoint_period	replica_checkpoint_period
slave_checkpoint_group	replica_checkpoint_group
slave_transaction_retries	replica_transaction_retries
slave_parallel_workers	replica_parallel_workers
slave_pending_jobs_size_max	replica_pending_jobs_size_max
pseudo_slave_mode	pseudo_replica_mode

다음 저장 프로시저는 Aurora MySQL 버전 3에서 새 이름을 갖습니다.

호환성을 위해 초기 Aurora MySQL 버전 3 릴리스에서 두 이름 중 하나를 사용할 수 있습니다. 향후 릴리스에서 기존 프로시저 이름이 제거되게 됩니다.

제거할 이름	새 이름 또는 기본 이름	
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position	
mysql.rds_set_external_master	mysql.rds_set_external_source	
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position	
mysql.rds_reset_external_master	mysql.rds_reset_external_source	
mysql.rds_next_master_log	mysql.rds_next_source_log	

AUTO_INCREMENT 값

Aurora MySQL 버전 3에서 Aurora에서는 각 DB 인스턴스를 다시 시작할 때 각 테이블의 AUTO_INCREMENT 값을 보존합니다. Aurora MySQL 버전 2에서는 다시 시작한 후에는 AUTO_INCREMENT 값이 보존되지 않았습니다.

스냅샷에서 복원하고, 특정 시점으로 복구를 수행하고, 클러스터를 복제하여 새 클러스터를 설정하는 경우 AUTO_INCREMENT 값이 보존되지 않습니다. 이런 경우 AUTO_INCREMENT 값은 스냅샷이 생성된 시점의 테이블에서 가장 큰 열 값을 기준으로 한 값으로 초기화됩니다. 이 동작은 이런 작업 중에 AUTO_INCREMENT 값이 보존되는 RDS for MySQL 8.0의 동작과 다릅니다.

이진 로그 복제

MySQL 8.0 커뮤니티 에디션에서는 기본적으로 이진 로그 복제를 설정해 둡니다. Aurora MySQL 버전 3에서는 기본적으로 이진 로그 복제를 해제해 둡니다.

> **Tip**
>
> Aurora 기본 제공 복제 기능으로 고가용성 요구 사항이 충족되면 이진 로그 복제를 해제해 둘 수 있습니다. 이렇게 하면 이진 로그 복제의 성능 오버헤드를 방지할 수 있습니다. 또한 이진 로그 복제를 관리하는 데 필요한 관련 모니터링 및 문제 해결을 방지할 수 있습니다.

Aurora는 MySQL 5.7 호환 소스에서 Aurora MySQL 버전 3으로 이진 로그 복제를 지원합니다. 소스 시스템은 Aurora MySQL DB 클러스터, RDS for MySQL DB 인스턴스 또는 온프레미스 MySQL 인스턴스일 수 있습니다.

커뮤니티 MySQL과 마찬가지로 Aurora MySQL은 특정 버전을 실행하는 소스에서 동일한 주 버전 또는 하나의 주 버전 이상을 실행하는 대상으로 복제를 지원합니다. 예를 들어 MySQL 5.6 호환 시스템에서 Aurora MySQL 버전 3으로 복제는 지원되지 않습니다. Aurora MySQL 버전 3에서 MySQL 5.7 호환 또는 MySQL 5.6 호환 시스템으로 복제는 지원되지 않습니다. 이진 로그 복제 사용에 대한 자세한 내용은 [Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제](#) 섹션을 참조하세요.

Aurora MySQL 버전 3에는 필터링된 복제와 같은 커뮤니티 MySQL 8.0에서 이진 로그 복제에 대한 개선이 포함됩니다. 커뮤니티 MySQL 8.0 개선 사항에 대한 자세한 내용은 MySQL 참조 설명서의 [서버가 복제 필터링 규칙을 평가하는 방법](#)을 참조하세요.

이진 로그 복제에 대한 트랜잭션 압축

이진 로그 압축에 대한 사용 정보는 MySQL 참조 설명서의 [이진 로그 트랜잭션 압축](#)을 참조하세요.

Aurora MySQL 버전 3의 이진 로그 압축에는 다음 제한 사항이 적용됩니다.

- 이진 로그 데이터가 허용되는 최대 패킷 크기보다 큰 트랜잭션은 압축되지 않습니다. 이는 Aurora MySQL 이진 로그 압축 설정이 켜져 있는지와 관계없이 해당됩니다. 이러한 트랜잭션은 압축되지 않고 복제됩니다.
- 아직 MySQL 8.0을 지원하지 않는 변경 데이터 캡처(CDC)의 커넥터를 사용하는 경우 이 기능을 사용할 수 없습니다. 이진 로그 압축을 사용하여 타사 커넥터를 철저히 테스트하는 것이 좋습니다. 또한 CDC에 binlog 복제를 사용하는 시스템에서 binlog 압축을 켜기 전에 테스트하는 것이 좋습니다.

Aurora MySQL 버전 3과 MySQL 8.0 커뮤니티 에디션 비교

다음 정보를 사용하여 다른 MySQL 8.0 호환 시스템에서 Aurora MySQL 버전 3으로 변환하는 경우 알아야 할 변경 사항에 대해 알아볼 수 있습니다.

일반적으로 Aurora MySQL 버전 3은 커뮤니티 MySQL 8.0.23의 기능 세트를 지원합니다. MySQL 8.0 커뮤니티 에디션의 일부 새로운 기능은 Aurora MySQL에는 적용되지 않습니다. 이러한 기능 중 일부는 Aurora 스토리지 아키텍처와 같은 Aurora의 일부 측면과 호환되지 않습니다. Amazon RDS 관리 서비스가 동등한 기능을 제공하기 때문에 다른 기능은 필요하지 않습니다. 커뮤니티 MySQL 8.0의 다음 기능은 Aurora MySQL 버전 3에서 지원되지 않거나 다르게 작동합니다.

모든 Aurora MySQL 버전 3 릴리스에 대한 릴리스 정보는 Aurora MySQL 릴리스 정보의 Amazon Aurora MySQL 버전 3에 대한 데이터베이스 엔진 업데이트를 참조하세요.

주제

Aurora MySQL 버전 3에서 MySQL 8.0 기능을 사용할 수 없음

커뮤니티 MySQL 8.0의 다음 기능은 Aurora MySQL 버전 3에서 사용할 수 없거나 다르게 작동합니다.

- Aurora MySQL에서는 Resource Groups 및 관련 SQL 문을 지원하지 않습니다.
- Aurora MySQL은 사용자 정의 테이블스페이스 실행 취소 및 관련 SQL 문(예: `CREATE UNDO TABLESPACE`, `ALTER UNDO TABLESPACE ... SET INACTIVE`, `DROP UNDO TABLESPACE`)을 지원하지 않습니다.
- Aurora MySQL은 3.06 미만의 Aurora MySQL 버전에서 테이블스페이스 자르기 실행 취소를 지원하지 않습니다. Aurora MySQL 버전 3.06 이상에서는 테이블스페이스 자르기 자동 실행 취소가 지원됩니다.
- MySQL 플러그인의 설정은 수정할 수 없습니다.
- X 플러그인은 지원되지 않습니다.
- 다중 소스 복제는 지원되지 않습니다.

역할 기반 권한 모델

Aurora MySQL 버전 3에서는 `mysql` 데이터베이스의 테이블을 직접 수정할 수 없습니다. 특히 `mysql.user` 테이블에 삽입하여 사용자를 설정할 수 없습니다. 대신 SQL 문을 사용하여 역할 기반 권한을 부여합니다. 또한 `mysql` 데이터베이스에 저장 프로시저를 비롯한 다른 종류의 객체를 생성할 수

없습니다. 여전히 mysql 테이블을 쿼리할 수 있습니다. 이진 로그 복제를 사용하는 경우 원본 클러스터의 mysql 테이블에 대한 직접 변경은 대상 클러스터로 복제되지 않습니다.

경우에 따라 애플리케이션에서 바로 가기를 사용하여 mysql 테이블에 삽입함으로써 사용자 또는 다른 객체를 만들 수 있습니다. 그렇다면 애플리케이션 코드를 변경하여 CREATE USER와 같은 해당 명령문을 사용합니다. 애플리케이션에서 mysql 데이터베이스에 저장 프로시저 또는 다른 객체를 생성하는 경우 다른 데이터베이스를 대신 사용합니다.

외부 MySQL 데이터베이스에서 마이그레이션하는 동안 데이터베이스 사용자의 메타데이터를 내보내려면 mysqldump 대신 MySQL Shell 명령을 사용할 수 있습니다. 자세한 내용은 Instance Dump Utility, Schema Dump Utility, and Table Dump Utility를 참조하세요.

많은 사용자 또는 애플리케이션에 대한 권한 관리를 단순화하려면 CREATE ROLE 문을 사용하여 권한 집합이 있는 역할을 만들 수 있습니다. 그런 다음, GRANT 및 SET ROLE 문과 current_role 함수를 사용하여 사용자 또는 애플리케이션에 역할을 할당하고 현재 역할을 전환하며 어떤 역할이 유효한지 확인할 수 있습니다. MySQL 8.0의 역할 기반 권한 시스템에 대한 자세한 내용은 MySQL 참조 설명서의 역할 사용을 참조하세요.

> **Important**
>
> 애플리케이션에서 직접 마스터 사용자를 사용하지 않는 것이 좋습니다. 대신에 애플리케이션에 필요한 최소 권한으로 생성한 데이터베이스 사용자를 사용하는 모범 사례를 준수하십시오.

주제

- rds_superuser_role
- 이진 로그 복제에 대한 권한 검사 사용자
- 다른 AWS 서비스에 액세스하기 위한 역할

rds_superuser_role

Aurora MySQL 버전 3에는 다음과 같은 모든 권한이 있는 특수 역할이 포함됩니다. 이 역할의 이름은 rds_superuser_role입니다. 각 클러스터의 기본 관리 사용자에게는 이미 이 역할이 부여되었습니다. rds_superuser_role 역할에는 모든 데이터베이스 객체에 대한 다음과 같은 권한이 포함됩니다.

- ALTER

- APPLICATION_PASSWORD_ADMIN
- ALTER ROUTINE
- CONNECTION_ADMIN
- CREATE
- CREATE ROLE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE
- FLUSH_OPTIMIZER_COSTS(Aurora MySQL 버전 3.09 이상)
- FLUSH_STATUS(Aurora MySQL 버전 3.09 이상)
- FLUSH_TABLES(Aurora MySQL 버전 3.09 이상)
- FLUSH_USER_RESOURCES(Aurora MySQL 버전 3.09 이상)
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT

- SHOW DATABASES
- SHOW_ROUTINE(Aurora MySQL 버전 3.04 이상)
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

역할 정의에는 WITH GRANT OPTION이 포함되므로 관리 사용자가 다른 사용자에게 해당 역할을 부여할 수 있습니다. 특히 관리자는 Aurora MySQL 클러스터를 대상으로 사용하여 이진 로그 복제를 수행하는 데 필요한 모든 권한을 부여해야 합니다.

> **Tip**
>
> 권한의 세부 정보를 모두 확인하려면 다음 명령문을 입력합니다.
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

이진 로그 복제에 대한 권한 검사 사용자

Aurora MySQL 버전 3에는 이진 로그(binlog) 복제에 대한 권한 검사 사용자인 rdsrepladmin_priv_checks_user가 포함되어 있습니다. 이 사용자는 rds_superuser_role의 권한 외에도 replication_applier 권한을 가집니다.

mysql.rds_start_replication 저장 절차를 직접적으로 호출하여 binlog 복제를 켜면 rdsrepladmin_priv_checks_user가 생성됩니다.

rdsrepladmin_priv_checks_user@localhost 사용자는 예약된 사용자입니다. 수정하지 마세요.

다른 AWS 서비스에 액세스하기 위한 역할

Aurora MySQL 버전 3에는 다른 AWS 서비스에 액세스하는 데 사용할 수 있는 역할이 포함됩니다. 이러한 역할을 권한 부여에 대한 대안으로 설정할 수 있습니다. 예를 들어 GRANT INVOKE LAMBDA ON *.* TO *user* 대신 GRANT AWS_LAMBDA_ACCESS TO *user*를 지정합니다. 다른 AWS 서비스에

대한 액세스 절차는 Amazon Aurora MySQL을 다른 AWS 서비스와 통합 섹션을 참조하세요. Aurora MySQL 버전 3에는 다른 AWS 서비스에 대한 액세스와 관련된 다음 역할이 포함됩니다.

- AWS_LAMBDA_ACCESS – INVOKE LAMBDA 권한에 대한 대안입니다. 사용 정보는 Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 단원을 참조하세요.
- AWS_LOAD_S3_ACCESS – LOAD FROM S3 권한에 대한 대안입니다. 사용 정보는 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 단원을 참조하세요.
- AWS_SELECT_S3_ACCESS – SELECT INTO S3 권한에 대한 대안입니다. 사용 정보는 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 단원을 참조하세요.
- AWS_COMPREHEND_ACCESS – INVOKE COMPREHEND 권한에 대한 대안입니다. 사용 정보는 데이터베이스 사용자에게 Aurora 기계 학습에 대한 액세스 권한 부여 단원을 참조하세요.
- AWS_SAGEMAKER_ACCESS – INVOKE SAGEMAKER 권한에 대한 대안입니다. 사용 정보는 데이터베이스 사용자에게 Aurora 기계 학습에 대한 액세스 권한 부여 단원을 참조하세요.
- AWS_BEDROCK_ACCESS – Amazon Bedrock에는 유사한 INVOKE 권한이 없습니다. 사용 정보는 데이터베이스 사용자에게 Aurora 기계 학습에 대한 액세스 권한 부여 단원을 참조하세요.

Aurora MySQL 버전 3에서 역할을 사용하여 액세스 권한을 부여하는 경우 SET ROLE *role_name* 또는 SET ROLE ALL 문을 사용하여 해당 역할을 활성화합니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다. 적합한 역할 이름을 AWS_SELECT_S3_ACCESS로 대체합니다.

```
# Grant role to user.

mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
```

```
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-----------------------------------------------------+
| CURRENT_ROLE()                                      |
+-----------------------------------------------------+
| `AWS_SELECT_S3_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-----------------------------------------------------+
```

데이터베이스 서버 ID 찾기

데이터베이스 서버 ID(`server_id`)는 이진 로깅(binlog) 복제에 필요합니다. 서버 ID를 찾는 방법은 Aurora MySQL과 커뮤니티 MySQL이 다릅니다.

커뮤니티 MySQL에서 서버 ID는 서버에 로그인한 상태에서 다음 구문을 사용하여 얻는 숫자입니다.

```
mysql> select @@server_id;

+-------------+
| @@server_id |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

Aurora MySQL에서 서버 ID는 DB 인스턴스에 로그인한 상태에서 다음 구문을 사용하여 얻는 DB 인스턴스 ID입니다.

```
mysql> select @@aurora_server_id;

+------------------------+
| @@aurora_server_id     |
+------------------------+
| mydbcluster-instance-2 |
+------------------------+
1 row in set (0.00 sec)
```

Binlog 복제에 대한 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 섹션을 참조하세요.

인증

커뮤니티 MySQL 8.0에서 기본 인증 플러그인은 caching_sha2_password입니다. Aurora MySQL 버전 3은 여전히 mysql_native_password 플러그인을 사용합니다. default_authentication_plugin 설정은 변경할 수 없습니다. 그러나 새 사용자를 생성하고 현재 사용자를 변경할 수 있으며 개별 암호는 새 인증 플러그인을 사용합니다. 다음은 한 예입니다.

```
mysql> CREATE USER 'testnewsha'@'%' IDENTIFIED WITH caching_sha2_password BY
 'aNewShaPassword';
Query OK, 0 rows affected (0.74 sec)
```

Aurora MySQL 버전 3으로 업그레이드

Aurora MySQL 버전 2에서 버전 3으로 데이터베이스를 업그레이드하는 내용은 DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드 섹션을 참고하시기 바랍니다.

Aurora MySQL 버전 2는 MySQL 5.7과 호환

이 주제에서는 Aurora MySQL 버전 2와 MySQL 5.7 Community Edition 간의 차이점에 대해 설명합니다.

Important

2024년 10월 31일에 Aurora MySQL 버전 2의 표준 지원이 종료되었습니다. 자세한 내용은 Amazon Aurora MySQL 호환 버전 2 표준 지원 종료 준비 단원을 참조하십시오.

Aurora MySQL 버전 2에서 지원되지 않는 기능

다음 기능은 MySQL 5.7에서 지원되지만 현재 Aurora MySQL 버전 2에서는 지원되지 않습니다.

- CREATE TABLESPACE SQL 문
- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼 풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제

- 온라인 버퍼 풀 크기 조정
- 암호 유효성 검사 플러그인 - 플러그인을 설치할 수 있지만, 지원되지 않습니다. 플러그인을 사용자 지정할 수 없습니다.
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- X 프로토콜

이러한 기능에 대한 자세한 내용은 MySQL 5.7 설명서를 참조하십시오.

Aurora MySQL 버전 2의 임시 테이블스페이스 동작

MySQL 5.7에서는 임시 테이블스페이스가 자동 확장되며 온디스크 임시 테이블을 수용하기 위해 필요에 따라 크기가 증가합니다. 임시 테이블이 삭제되면 여유 공간을 새 임시 테이블에 재사용할 수 있지만 임시 테이블스페이스는 확장된 크기로 유지되며 축소되지 않습니다. 엔진이 재시작되면 임시 테이블스페이스가 삭제되고 다시 생성됩니다.

Aurora MySQL 버전 2에서는 다음 동작이 적용됩니다.

- 버전 2.10 이상에서 생성된 새 Aurora MySQL DB 클러스터의 경우 데이터베이스를 다시 시작하면 임시 테이블스페이스가 제거되고 다시 생성됩니다. 이렇게 하면 동적 크기 조정 기능을 통해 스토리지 공간을 회수할 수 있습니다.
- 다음으로 업그레이드된 기존 Aurora MySQL DB 클러스터의 경우:
  - 버전 2.10 이상 - 데이터베이스를 다시 시작하면 임시 테이블스페이스가 제거되고 다시 생성됩니다. 이렇게 하면 동적 크기 조정 기능을 통해 스토리지 공간을 회수할 수 있습니다.
  - 버전 2.09 - 데이터베이스를 다시 시작할 때 임시 테이블 공간이 제거되지 않습니다.

다음 쿼리를 사용하여 Aurora MySQL 버전 2 DB 클러스터의 임시 테이블스페이스 크기를 확인할 수 있습니다.

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

자세한 내용은 MySQL 설명서에서 [The Temporary Tablespace](임시 테이블스페이스)를 참조하세요.

온디스크 임시 테이블에 대한 스토리지 엔진

Aurora MySQL 버전 2는 인스턴스의 역할에 따라 온디스크 내부 임시 테이블에 대해 서로 다른 스토리지 엔진을 사용합니다.

- 라이터 인스턴스에서 온디스크 임시 테이블은 기본적으로 InnoDB 스토리지 엔진을 사용합니다. Aurora 클러스터 볼륨의 임시 테이블스페이스에 저장됩니다.

  DB 파라미터의 `internal_tmp_disk_storage_engine` 값을 수정하여 라이터 인스턴스에서 이 동작을 변경할 수 있습니다. 자세한 내용은 [인스턴스 수준 파라미터](#) 단원을 참조하십시오.
- 리더 인스턴스에서 온디스크 임시 테이블은 로컬 스토리지를 사용하는 MyISAM 스토리지 엔진을 사용합니다. 읽기 전용 인스턴스가 Aurora 클러스터 볼륨에 데이터를 저장할 수 없기 때문입니다.

Amazon Aurora MySQL를 사용한 보안

Amazon Aurora MySQL 보안은 다음과 같이 세 가지 수준에서 관리됩니다.

- Aurora MySQL DB 클러스터 및 DB 인스턴스에서 Amazon RDS 관리 작업을 수행할 수 있는 사용자를 제어하려면 AWS Identity and Access Management(IAM)를 사용합니다. IAM 자격 증명을 사용하여 AWS에 연결할 때, AWS 계정은 Amazon RDS 관리 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책을 보유하고 있어야 합니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 단원을 참조하세요.

  IAM을 사용하여 Amazon RDS 콘솔에 액세스하는 경우 먼저 IAM 사용자 자격 증명으로 AWS Management Console에 로그인합니다. 그런 다음 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔로 이동합니다.
- Aurora MySQL DB 클러스터는 Amazon VPC 서비스를 기반으로 Virtual Private Cloud(VPC)에 생성해야 합니다. Aurora MySQL DB 클러스터에서 DB 인스턴스의 엔드포인트 및 포트에 연결할 수 있는 디바이스 또는 Amazon EC2 인스턴스를 제어하려면 VPC 보안 그룹을 사용합니다. 전송 계층 보안(TLS)을 사용하여 이러한 엔드포인트 및 포트 연결을 만들 수 있습니다. 그 밖에도 기업의 방화벽 규칙을 통해 기업에서 이용하는 디바이스의 DB 인스턴스 연결 여부를 제어하는 것도 가능합니다. VPC에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 단원을 참조하세요.

  지원되는 VPC 테넌시는 Aurora MySQL DB 클러스터에서 사용하는 DB 인스턴스 클래스에 따라 다릅니다. `default` VPC 테넌시로 VPC가 공유된 하드웨어에서 실행됩니다. 하지만 `dedicated` VPC 테넌시일 때는 VPC는 전용 하드웨어 인스턴스에서 실행됩니다. 버스트 가능 성능 DB 인스턴스 클래스는 기본 VPC 테넌시만 지원합니다. 버스트 가능 성능 DB 인스턴스 클래스에는 db.t2, db.t3 및 db.t4g DB 인스턴스 클래스가 포함됩니다. 다른 모든 Aurora MySQL DB 인스턴스 클래스는 기본 및 전용 VPC 테넌시를 모두 지원합니다.

  > **Note**
  >
  > T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 개발 및 테스트에 T 인스턴스 클래스 사용 섹션을 참조하세요.

  인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하세요. `default` 및 `dedicated` VPC 테넌시에 대한 자세한 정보는 Amazon Elastic Compute Cloud 사용 설명서의 전용 인스턴스를 참조하십시오.

- Amazon Aurora MySQL DB 클러스터에 대한 로그인 및 권한을 인증하기 위해서는 다음 접근 방식 중 하나를 따르거나 두 방식을 조합할 수 있습니다.
  - 독립형 MySQL 인스턴스와 동일한 접근법을 사용할 수 있습니다.

    `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE` 및 `SET PASSWORD` 등의 명령은 온프레미스 데이터베이스에서 작동하는 것과 마찬가지로 작동하며, 데이터베이스 스키마 테이블을 직접 수정할 때도 동일합니다. 자세한 내용은 MySQL 설명서의 [액세스 제어 및 계정 관리](#) 단원을 참조하십시오.
  - 또한 IAM 데이터베이스 인증을 사용할 수도 있습니다.

    IAM 데이터베이스 인증의 경우, IAM 사용자 또는 IAM 역할 및 인증 토큰을 이용해 DB 클러스터에 인증합니다. 인증 토큰은 서명 버전 4 서명 프로세스를 통해 생성하는 고유 값입니다. IAM 데이터베이스 인증을 사용하면 동일한 자격 증명을 사용해 AWS 리소스 및 데이터베이스에 대한 액세스를 제어할 수 있습니다. 자세한 내용은 [IAM 데이터베이스 인증](#) 섹션을 참조하세요.

> **Note**
>
> 자세한 내용은 [Amazon Aurora의 보안](#) 단원을 참조하십시오.

다음 섹션에서 Aurora MySQL 및 Aurora MySQL DB 클러스터와의 TLS 연결에 대한 사용자 권한 관련 정보를 참조하세요.

주제

Amazon Aurora MySQL을 사용한 마스터 사용자 권한

Amazon Aurora MySQL DB 인스턴스를 생성할 때 마스터 사용자는 마스터 사용자 계정 권한에 나열된 기본 권한을 갖습니다.

DB 클러스터를 생성할 때는 각 DB 클러스터의 관리 서비스를 제공하기 위해 `admin` 및 `rdsadmin` 사용자가 만들어집니다. `rdsadmin` 계정을 삭제하려고 하거나, 계정 이름 또는 암호를 변경하려고 하거나, 계정 권한을 변경하려고 하면 오류가 발생합니다.

Aurora MySQL 버전 2 DB 클러스터에서는 DB 클러스터를 생성할 때 `admin` 및 `rdsadmin` 사용자가 만들어집니다. Aurora MySQL 버전 3 DB 클러스터에서는 `admin`, `rdsadmin`, 및 `rds_superuser_role` 사용자가 만들어집니다.

> **Important**
>
> 애플리케이션에서 직접 마스터 사용자를 사용하지 않는 것이 좋습니다. 대신에 애플리케이션에 필요한 최소 권한으로 생성한 데이터베이스 사용자를 사용하는 모범 사례를 준수하십시오.

Aurora MySQL DB 클러스터의 관리를 위해 기본 `kill` 및 `kill_query` 명령은 사용이 제한됩니다. 대신, Amazon RDS 명령으로 `rds_kill` 및 `rds_kill_query`를 사용하여 Aurora MySQL DB 인스턴스의 사용자 세션이나 쿼리를 종료할 수 있습니다.

> **Note**
>
> 중국(닝샤) 리전에서는 데이터베이스 인스턴스와 스냅샷의 암호화가 지원되지 않습니다.

Aurora MySQL DB 클러스터에 대한 TLS 연결

Amazon Aurora MySQL DB 클러스터는 RDS for MySQL DB 인스턴스와 동일한 프로세스 및 퍼블릭 키를 사용하여 애플리케이션의 전송 계층 보안(TLS) 연결을 지원합니다.

Amazon RDS가 TLS 인증서를 생성한 후 Amazon RDS가 인스턴스를 프로비저닝할 때 DB 인스턴스에 인증서를 설치합니다. 인증 기관이 서명하는 SSL 인증서에는 TLS 인증서에는 스푸핑 공격으로부터 보호해주는 TLS 인증서를 위한 일반 이름(CN)으로 DB 인스턴스 엔드포인트가 포함되어 있습니다. 그 결과 클라이언트가 Subject Alternative Names(SAN)를 지원할 경우 TLS를 이용한 DB 클러스터 연결의 유일한 방법은 DB 클러스터 엔드포인트를 이용하는 것입니다. 그렇지 않으면 라이터 인스턴스의 인스턴스 엔드포인트를 사용해야 합니다.

인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화](#) 단원을 참조하십시오.

TLS가 포함된 SAN을 지원하는 클라이언트로 AWS 드라이버를 사용하는 것이 좋습니다. AWS JDBC 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 [Amazon Web Services (AWS) JDBC Driver GitHub repository](#)를 참조하세요.

주제

- Aurora MySQL DB 클러스터에 대한 TLS 연결 요구
- Aurora MySQL에 지원되는 TLS 버전
- Aurora MySQL DB 클러스터 연결을 위한 암호 그룹 구성
- Aurora MySQL DB 클러스터에 대한 연결 암호화

Aurora MySQL DB 클러스터에 대한 TLS 연결 요구

`require_secure_transport` DB 클러스터 파라미터를 사용하여 Aurora MySQL DB 클러스터에 대한 모든 사용자 연결에서 TLS를 사용하도록 요구할 수 있습니다. 기본적으로 `require_secure_transport` 파라미터는 `OFF`로 설정됩니다. `require_secure_transport` 파라미터를 `ON`으로 설정하면 해당 DB 클러스터에 대한 연결 시 TLS를 요구합니다.

`require_secure_transport` 파라미터 값은 DB 클러스터의 파라미터 그룹을 업데이트하여 설정할 수 있습니다. 변경 사항을 적용하기 위해 DB 클러스터를 재부팅할 필요가 없습니다. 파라미터 그룹에 대한 자세한 정보는 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.

> **Note**
>
> `require_secure_transport` 파라미터는 Aurora MySQL 버전 2와 3에서만 사용할 수 있습니다. 사용자 지정 DB 클러스터 파라미터 그룹에서 이 파라미터를 설정할 수 있습니다. DB 인스턴스 파라미터 그룹에서는 이 파라미터를 사용할 수 없습니다.

DB 클러스터에 대해 `require_secure_transport` 파라미터를 `ON`으로 설정하면 암호화된 연결을 설정할 수 있는 경우 데이터베이스 클라이언트가 인스턴스에 연결할 수 있습니다. 그렇지 않으면 다음과 유사한 오류 메시지가 클라이언트에 반환됩니다.

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

Aurora MySQL에 지원되는 TLS 버전

Aurora MySQL은 전송 계층 보안(TLS) 버전 1.0, 1.1, 1.2, 1.3을 지원하지 않습니다. Aurora MySQL 버전 3.04.0부터는 TLS 1.3 프로토콜을 사용하여 연결을 보호할 수 있습니다. 다음 표는 Aurora MySQL 버전에 대한 TLS 지원을 보여줍니다.

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	기본값
Aurora MySQL 버전 2	지원	지원	지원	지원되지 않음	지원되는 모든 TLS 버전
Aurora MySQL 버전 3(3.04.0 미만)	지원	지원	지원	지원되지 않음	지원되는 모든 TLS 버전
Aurora MySQL 버전 3(3.04.0 이상)	지원되지 않음	지원되지 않음	지원	지원	지원되는 모든 TLS 버전

Important

버전 2 및 3.04.0 미만 버전의 Aurora MySQL 클러스터에 사용자 지정 파라미터 그룹을 사용하는 경우 TLS 1.0 및 1.1은 안전성이 떨어지므로 TLS 1.2를 사용하는 것이 좋습니다. MySQL 8.0.26 Community Edition과 Aurora MySQL 3.03 및 해당 마이너 버전에서는 TLS 버전 1.1 및 1.0에 대한 지원이 중단되었습니다.
MySQL 8.0.28의 Community Edition 및 호환되는 Aurora MySQL 버전 3.04.0 이상은 TLS 1.1 및 TLS 1.0을 지원하지 않습니다. Aurora MySQL 버전 3.04.0 이상을 사용하는 경우 사용자 지정 파라미터 그룹에서 TLS 프로토콜을 1.0 및 1.1로 설정하지 마세요.
Aurora MySQL 버전 3.04.0 이상의 경우 기본 설정은 TLS 1.3 및 TLS 1.2입니다.

`tls_version` DB 클러스터 파라미터를 사용하여 허용되는 프로토콜 버전을 나타낼 수 있습니다. 대부분의 클라이언트 도구 또는 데이터베이스 드라이버에 대해 유사한 클라이언트 파라미터가 있습니다. 일부 이전 클라이언트는 최신 TLS 버전을 지원하지 않을 수 있습니다. 기본적으로 DB 클러스터는 서버와 클라이언트 구성 모두에서 허용되는 가장 높은 TLS 프로토콜 버전을 사용하려고 합니다.

`tls_version` DB 클러스터 파라미터를 다음 값 중 하나로 설정합니다.

- TLSv1.3
- TLSv1.2
- TLSv1.1
- TLSv1

또한 tls_version 파라미터를 쉼표로 구분된 목록의 문자열로 설정할 수 있습니다. TLS 1.2 및 TLS 1.0 프로토콜을 모두 사용하려면 tls_version 파라미터는 가장 낮은 프로토콜부터 가장 높은 프로토콜까지 모든 프로토콜을 포함해야 합니다. 이 경우 tls_version은 다음과 같이 설정됩니다.

```
tls_version=TLSv1,TLSv1.1,TLSv1.2
```

DB 클러스터 파라미터 그룹에서 파라미터 수정에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요. AWS CLI를 클릭하여 tls_version DB 클러스터 파라미터를 수정하려면 ApplyMethod를 pending-reboot로 설정해야 합니다. 애플리케이션 메서드가 pending-reboot이면 파라미터 그룹과 연결된 DB 클러스터를 중지하고 다시 시작한 후에 파라미터 변경 내용이 적용됩니다.

Aurora MySQL DB 클러스터 연결을 위한 암호 그룹 구성

구성 가능한 암호 그룹을 사용하면 데이터베이스 연결의 보안을 더 잘 제어할 수 있습니다. 데이터베이스에 대한 클라이언트 TLS 연결을 보호하도록 허용할 암호 그룹 목록을 지정할 수 있습니다. 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. 이렇게 하면 안전하지 않거나 사용되지 않는 암호의 사용을 방지할 수 있습니다.

구성 가능한 암호 그룹은 Aurora MySQL 버전 3 및 Aurora MySQL 버전 2에서 지원됩니다. 연결을 암호화하는 데 허용되는 TLS 1.2, TLS 1.1, TLS 1.0 암호 목록을 지정하려면 ssl_cipher 클러스터 파라미터를 수정합니다. AWS Management Console, AWS CLI 또는 RDS API를 사용하는 클러스터 파라미터 그룹의 ssl_cipher 파라미터를 설정합니다.

ssl_cipher 파라미터를 TLS 버전에 대한 쉼표로 구분된 암호 값의 문자열로 설정합니다. 클라이언트 애플리케이션의 경우 데이터베이스에 연결할 때 --ssl-cipher 옵션을 사용하여 암호화된 연결에 사용할 암호를 지정할 수 있습니다. 데이터베이스 연결에 대한 자세한 내용은 Amazon Aurora MySQL DB 클러스터에 연결 섹션을 참조하세요.

Aurora MySQL 버전 3.04.0 이상부터 TLS 1.3 암호 그룹을 지정할 수 있습니다. 허용되는 TLS 1.3 암호 그룹을 지정하려면 파라미터 그룹에서 tls_ciphersuites 파라미터를 수정합니다. TLS 1.3에서는 사용되는 키 교환 메커니즘과 인증서를 제거하는 이름 지정 규칙의 변경으로 인해 사용 가능한 암호 그

릅 수가 감소했습니다. `tls_ciphersuites`를 TLS 1.3에 대한 쉼표로 구분된 암호 값의 문자열로 설정합니다.

다음 표에는 각 암호에 대해 TLS 암호화 프로토콜 및 유효한 Aurora MySQL 엔진 버전과 함께 지원되는 암호가 나와 있습니다.

암호	암호화 프로토콜	지원되는 Aurora MySQL 버전
`DHE-RSA-AES128-SHA`	TLS 1.0	3.01.0 이상, 모든 2.11.0 미만
`DHE-RSA-AES128-SHA256`	TLS 1.2	3.01.0 이상, 모든 2.11.0 미만
`DHE-RSA-AES128-GCM-SHA256`	TLS 1.2	3.01.0 이상, 모든 2.11.0 미만
`DHE-RSA-AES256-SHA`	TLS 1.0	3.03.0 이하, 2.11.0 미만의 모든 버전
`DHE-RSA-AES256-SHA256`	TLS 1.2	3.01.0 이상, 모든 2.11.0 미만
`DHE-RSA-AES256-GCM-SHA384`	TLS 1.2	3.01.0 이상, 모든 2.11.0 미만
`ECDHE-RSA-AES128-SHA`	TLS 1.0	3.01.0 이상, 2.09.3 이상, 2.10.2 이상
`ECDHE-RSA-AES128-SHA256`	TLS 1.2	3.01.0 이상, 2.09.3 이상, 2.10.2 이상
`ECDHE-RSA-AES128-GCM-SHA256`	TLS 1.2	3.01.0 이상, 2.09.3 이상, 2.10.2 이상
`ECDHE-RSA-AES256-SHA`	TLS 1.0	3.01.0 이상, 2.09.3 이상, 2.10.2 이상
`ECDHE-RSA-AES256-SHA384`	TLS 1.2	3.01.0 이상, 2.09.3 이상, 2.10.2 이상

암호	암호화 프로토콜	지원되는 Aurora MySQL 버전
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 이상, 2.09.3 이상, 2.10.2 이상
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 이상
TLS_AES_256_GCM_SHA384	TLS 1.3	3.04.0 이상
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	3.04.0 이상

Note

DHE-RSA 암호는 2.11.0 이전의 Aurora MySQL 버전에서만 지원됩니다. 버전 2.11.0 이상은 ECDHE 암호만 지원합니다.

DB 클러스터 파라미터 그룹에서 파라미터 수정에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요. CLI를 사용하여 `ssl_cipher` DB 클러스터 파라미터를 수정하려면 `ApplyMethod`를 `pending-reboot`로 설정해야 합니다. 애플리케이션 메서드가 `pending-reboot`이면 파라미터 그룹과 연결된 DB 클러스터를 중지하고 다시 시작한 후에 파라미터 변경 내용이 적용됩니다.

describe-engine-default-cluster-parameters CLI 명령을 사용하여 특정 파라미터 그룹 제품군에 대해 현재 지원되는 암호 그룹을 결정할 수도 있습니다. 다음 예에서는 Aurora MySQL 버전 2에 대한 `ssl_cipher` 클러스터 파라미터에 허용되는 값을 가져오는 방법을 보여줍니다.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-
```

```
AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-
RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
       ...some output truncated...
```

암호에 대한 자세한 내용은 MySQL 설명서의 ssl_cipher 변수를 참조하세요. 암호 그룹 형식에 대한 자세한 내용은 OpenSSL 웹사이트에서 openssl-ciphers 목록 형식 및 openssl-ciphers 문자열 형식 문서를 참조하세요.

Aurora MySQL DB 클러스터에 대한 연결 암호화

기본 `mysql` 클라이언트를 사용하여 연결을 암호화하려면 `--ssl-ca` 파라미터를 사용하여 mysql 클라이언트를 실행하고 다음과 같은 퍼블릭 키 등을 참조합니다.

MySQL 5.7 및 8.0의 경우:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

MySQL 5.6의 경우:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

*`full_path_to_CA_certificate`*를 인증 기관(CA) 인증서의 전체 경로로 바꿉니다. 인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.

특정 사용자 계정에 대한 TLS 연결을 요구할 수 있습니다. 예를 들면 MySQL 버전에 따라 다음 명령문 중 하나를 사용하여 사용자 계정 `encrypted_user`에 대한 TLS 연결을 요구할 수 있습니다.

MySQL 5.7 및 8.0의 경우:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

MySQL 5.6의 경우:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

RDS 프록시를 사용하는 경우 일반적인 클러스터 엔드포인트 대신 프록시 엔드포인트에 연결합니다. Aurora DB 클러스터에 직접 연결하는 것과 같은 방법으로 프록시 연결에 SSL/TLS를 필수 또는 선택 사항으로 지정할 수 있습니다. RDS 프록시 사용에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

> **Note**
>
> MySQL과의 TLS 연결에 대한 자세한 정보는 MySQL 설명서를 참조하세요.

새 TLS 인증서를 사용하여 Aurora MySQL DB 클러스터에 연결할 애플리케이션 업데이트

2023년 1월 13일부터 Amazon RDS는 전송 계층 보안(TLS)을 사용하여 Aurora DB 클러스터에 연결하기 위한 용도의 새 인증 기관(CA) 인증서를 게시했습니다. 아래에서 새 인증서를 사용하기 위해 애플리케이션을 업데이트하는 방법에 관한 정보를 찾으실 수 있습니다.

이 주제는 클라이언트 애플리케이션에서 TLS를 사용하여 DB 클러스터에 연결하는지 여부를 판단하는데 도움이 됩니다. SSL/TLS를 사용해 연결한다면 이 애플리케이션에서 연결 시 인증서 확인이 필요한지 여부를 추가로 확인할 수 있습니다.

> **Note**
>
> 어떤 애플리케이션은 서버에서 인증서를 성공적으로 확인할 수 있는 경우에만 Aurora MySQL DB 클러스터에 연결하도록 구성되어 있습니다.
> 이러한 애플리케이션의 경우 클라이언트 애플리케이션 트러스트 스토어를 업데이트하여 새 CA 인증서를 포함해야 합니다.

클라이언트 애플리케이션 트러스트 스토어에서 CA 인증서를 업데이트한 후에는 DB 클러스터에서 인증서를 교환할 수 있습니다. 이 절차를 프로덕션 환경에서 구현하기 전에 개발 또는 스테이징 환경에서 테스트해볼 것을 적극 권장합니다.

인증서 교환에 대한 자세한 내용은 SSL/TLS 인증서 교체 단원을 참조하십시오. 인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오. Aurora MySQL DB 클러스터에서 TLS를 사용하는 방법에 대한 자세한 내용은 Aurora MySQL DB 클러스터에 대한 TLS 연결 섹션을 참조하세요.

주제

애플리케이션에서 TLS를 사용하여 Aurora MySQL DB 클러스터에 연결하는지 여부 확인

Aurora MySQL 버전 2(MySQL 5.7과 호환됨)를 사용 중이고 성능 스키마가 활성화되어 있는 경우 다음 쿼리를 실행하여 연결 시 TLS를 사용하는지 여부를 확인합니다. 성능 스키마 활성화에 대한 자세한 내용은 MySQL 설명서의 성능 스키마 빠른 시작을 참조하십시오.

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

이 샘플 출력에서는 사용자 본인의 세션(admin)과 webapp1로 로그인된 애플리케이션에서 모두 TLS를 사용 중임을 알 수 있습니다.

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

클라이언트에서 연결 시 인증서 확인이 필요한지 여부 확인

JDBC 클라이언트 및 MySQL 클라이언트에서 연결 시 인증서 확인이 필요한지 여부를 확인할 수 있습니다.

JDBC

아래의 MySQL 커넥터/J 8.0 관련 예시에서는 애플리케이션의 JDBC 연결 속성을 확인하여 성공적인 연결을 위해 유효한 인증서가 필요한지 여부를 판단하는 한 가지 방법을 보여줍니다. MySQL에 대한 모든 JDBC 연결 옵션에 대한 자세한 내용은 MySQL 문서의 구성 속성을 참조하십시오.

MySQL 커넥터/J 8.0을 사용 중인 경우 다음 예제와 같이 연결 속성에서 `sslMode`가 `VERIFY_CA` 또는 `VERIFY_IDENTITY`로 설정되어 있다면 TLS 연결 시 서버 CA 인증서에 대한 확인이 필요합니다.

```
Properties properties = new Properties();
```

```
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> 데이터베이스에 연결할 때 TLS를 사용하도록 애플리케이션을 명시적으로 구성하지 않았더라도 MySQL Java Connector v5.1.38 이상 또는 MySQL Java Connector v8.0.9 이상을 사용하여 데이터베이스에 연결하는 경우, 이러한 클라이언트 드라이버는 기본적으로 TLS를 사용합니다. 또한 TLS 사용 시 부분 인증서 확인을 수행하고 데이터베이스 서버 인증서가 만료되면 연결에 실패합니다.

MySQL

MySQL 클라이언트에 관한 다음 예시에서는 스크립트의 MySQL 연결을 확인하여 성공적인 연결을 위해 유효한 인증서가 필요한지 여부를 판단하는 두 가지 방법을 보여줍니다. MySQL 클라이언트의 모든 연결 옵션에 대한 자세한 내용은 MySQL 문서의 [암호화된 연결을 위한 클라이언트 측 구성]을 참조하십시오.

MySQL 5.7 또는 MySQL 8.0 클라이언트를 사용 중인 경우 다음 예제와 같이 `--ssl-mode` 옵션에 대해 `VERIFY_CA` 또는 `VERIFY_IDENTITY`를 지정했다면 TLS 연결 시 서버 CA 인증서에 대한 확인이 필요합니다.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

MySQL 5.6 클라이언트를 사용 중인 경우 다음 예시와 같이 `--ssl-verify-server-cert` 옵션을 지정했다면 SSL 연결 시 서버 CA 인증서에 대한 확인이 필요합니다.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

애플리케이션 트러스트 스토어 업데이트

MySQL 애플리케이션을 위한 트러스트 스토어 업데이트에 관한 자세한 내용은 MySQL 문서의 [SSL 인증서 설치] 단원을 참조하십시오.

> **Note**
>
> 트러스트 스토어를 업데이트할 때 새 인증서를 추가할 뿐 아니라 이전 인증서를 유지할 수도 있습니다.

JDBC를 위한 애플리케이션 트러스트 스토어 업데이트

TLS 연결을 위해 JDBC를 사용하는 애플리케이션에 대해 트러스트 스토어를 업데이트할 수 있습니다.

루트 인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화](#) 단원을 참조하십시오.

인증서를 가져오는 샘플 스크립트는 [트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트](#) 섹션을 참조하세요.

애플리케이션에서 mysql JDBC를 사용 중인 경우 애플리케이션에서 다음 속성을 설정하십시오.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> 보안 모범 사례로 여기에 표시된 프롬프트 이외의 암호를 지정하는 것이 좋습니다.

애플리케이션을 시작할 때 다음 속성을 설정하십시오.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

TLS 연결 설정을 위한 Java 코드 예제

다음 코드 예시에서는 JDBC를 사용하여 서버 인증서를 확인하는 SSL 연결을 설정하는 방법을 보여줍니다.

```
public class MySQLSSLTest {

        private static final String DB_USER = "user name";
```

```
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

Important

데이터베이스 연결에서 TLS를 사용함을 확인하고 애플리케이션 트러스트 스토어를 업데이트한 후에는 데이터베이스에서 rds-ca-rsa2048-g1 인증서를 사용하도록 업데이트할 수 있습니다. 지침은 DB 인스턴스를 수정하여 CA 인증서 업데이트의 3단계를 참조하십시오.

Aurora MySQL에 Kerberos 인증 사용

사용자가 Aurora MySQL DB 클러스터에 접속하려고 할 때 Kerberos 인증을 사용하여 사용자를 인증할 수 있습니다. 이를 위해서는 Kerberos 인증을 위한 AWS Directory Service for Microsoft Active Directory를 사용하도록 DB 클러스터를 구성해야 합니다. AWS Directory Service for Microsoft Active Directory는 AWS Managed Microsoft AD라고도 불립니다. AWS Directory Service에서 사용할 수 있는 기능입니다. 자세한 내용은 AWS Directory Service 관리 가이드의 AWS Directory Service란 무엇입니까?를 참조하세요.

시작하려면 사용자 자격 증명을 저장할 AWS Managed Microsoft AD 디렉터리를 만듭니다. 그런 다음 Aurora MySQL DB 클러스터에 Active Directory의 도메인 및 기타 정보를 제공합니다. Aurora MySQL DB 클러스터에 대해 사용자가 인증될 때 AWS Managed Microsoft AD 디렉터리에 인증 요청이 전달됩니다.

모든 자격 증명을 동일한 디렉터리에 보관하면 시간과 노력을 절약할 수 있습니다. 이 접근 방식의 경우, 여러 DB 클러스터에 대한 자격 증명을 보관하고 관리할 수 있는 중앙 집중식 공간이 있습니다. 디렉터리를 사용하면 전체 보안 프로필을 향상할 수도 있습니다.

자체 온프레미스 Microsoft Active Directory에서 자격 증명에 액세스할 수도 있습니다. 이렇게 하려면 AWS Managed Microsoft AD 디렉터리가 온프레미스 Microsoft Active Directory를 신뢰하도록 신뢰 도메인 관계를 만듭니다. 그러면 사용자가 온프레미스 네트워크에서 워크로드에 액세스할 때와 동일한 Windows SSO(Single Sign-On) 환경을 사용하여 Aurora MySQL DB 클러스터에 액세스할 수 있습니다.

데이터베이스는 Kerberos 또는 AWS Identity and Access Management(IAM) 인증을 사용하거나, Kerberos 인증과 IAM 인증을 모두 사용할 수 있습니다. 하지만 Kerberos 및 IAM 인증은 서로 다른 인증 방법을 제공하므로, 특정 사용자는 두 가지 인증 방법 중 하나만 사용하여 데이터베이스에 로그인해야 하며 둘 다 사용할 수는 없습니다. IAM 인증에 관한 자세한 내용은 IAM 데이터베이스 인증 단원을 참조하십시오.

목차

Aurora MySQL DB 클러스터에 대한 Kerberos 인증 개요

Aurora MySQL DB 클러스터에 대해 Kerberos 인증을 설정하려면 아래의 일반 단계를 완료합니다. 이러한 단계는 다음에 자세히 설명되어 있습니다.

1. AWS Managed Microsoft AD를 사용하여 AWS Managed Microsoft AD 디렉터리를 생성합니다. AWS Management Console, AWS CLI 또는 AWS Directory Service를 사용하여 디렉터리를 생성할 수 있습니다. 자세한 내용은 AWS Directory Service 관리 가이드의 AWS Managed Microsoft AD 디렉터리 생성을 참조하세요.
2. 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 AWS Identity and Access Management(IAM) 역할을 생성합니다. 이 역할을 사용하여 Amazon Aurora에서 디렉터리를 호출할 수 있습니다.

   역할이 액세스를 허용하려면 AWS 리전에서 AWS 계정의 AWS Security Token Service(AWS STS) 엔드포인트를 활성화해야 합니다. AWS STS 엔드포인트는 기본적으로 모든 AWS 리전에서 활성화되어 있으므로 별도의 조치 없이 사용할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 리전에서 AWS STS 활성화 및 비활성화를 참조하세요.

3. Microsoft Active Directory 도구를 사용하여 AWS Managed Microsoft AD 디렉터리에서 사용자를 만들고 구성합니다. Active Directory에서 사용자를 생성하는 방법에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD에서 사용자 및 그룹 관리를 참조하세요.
4. Aurora MySQL DB 클러스터를 생성하거나 수정합니다. 생성 요청에 CLI 또는 RDS API를 사용하는 경우 `Domain` 파라미터를 사용해 도메인 식별자를 지정합니다. 디렉터리를 만들 때 생성된 `d-*` 식별자와 생성한 IAM 역할의 이름을 사용합니다.

   Kerberos 인증을 사용하도록 기존 Aurora MySQL DB 클러스터를 수정하는 경우 DB 클러스터에 대해 도메인 및 IAM 역할 파라미터를 설정합니다. 도메인 디렉터리와 동일한 VPC에서 DB 클러스터를 찾습니다.
5. Amazon RDS 기본 사용자 자격 증명을 사용하여 Aurora MySQL DB 클러스터에 연결합니다. 6단계: Kerberos 인증을 사용하는 Aurora MySQL 사용자 생성의 지침에 따라 Aurora MySQL 내에 데이터베이스 사용자를 생성합니다.

   이 방법으로 생성한 사용자는 Kerberos 인증을 사용하여 Aurora MySQL DB 클러스터에 로그인할 수 있습니다. 자세한 내용은 Kerberos 인증을 사용하여 Aurora MySQL에 연결 단원을 참조하십시오.

온프레미스 또는 자체 호스팅된 Microsoft Active Directory로 Kerberos 인증을 사용하려면 포리스트 트러스트를 생성합니다. 포리스트 트러스트는 두 도메인 그룹 간의 트러스트 관계입니다. 단방향 또는 양방향 트러스트가 가능합니다. AWS Directory Service를 사용하여 포리스트 신뢰를 설정하는 방법에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 신뢰 관계를 생성해야 하는 경우를 참조하세요.

Aurora MySQL에 대한 Kerberos 인증의 제한 사항

Aurora MySQL에 대한 Kerberos 인증에는 다음과 같은 제한 사항이 적용됩니다.

- Kerberos 인증은 Aurora MySQL 버전 3.03 이상에서 지원됩니다.

  AWS 리전 지원에 대한 자세한 내용은 Aurora MySQL을 사용하는 Kerberos 인증 단원을 참조하세요.
- Aurora MySQL에서 Kerberos 인증을 사용하려면 MySQL 클라이언트 또는 커넥터가 Unix 플랫폼에서는 버전 8.0.26 이상, Windows에서는 8.0.27 이상을 사용해야 합니다. 그렇지 않으면 클라이언트 측`authentication_kerberos_client` 플러그인을 사용할 수 없으며 인증할 수 없습니다.

- AWS Managed Microsoft AD는 Aurora MySQL에서만 지원됩니다. 하지만 Aurora MySQL DB 클러스터에 조인하여 동일한 AWS 리전의 서로 다른 계정이 소유한 Managed Microsoft AD 도메인을 공유할 수 있습니다.

  자체 온프레미스 Active Directory를 사용할 수도 있습니다. 자세한 내용은 2단계: (선택 사항) 온프레미스 Active Directory에 대한 신뢰 생성 단원을 참조하십시오.
- Kerberos를 사용하여 MySQL 클라이언트나 Windows 운영 체제의 드라이버에서 Aurora MySQL 클러스터에 연결하는 사용자를 인증하려는 경우, 기본적으로 데이터베이스 사용자 이름의 대소문자가 Active Directory에 있는 사용자의 대소문자와 일치해야 합니다. 예를 들어 Active Directory의 사용자가 `Admin`으로 표시될 경우 데이터베이스 사용자 이름도 `Admin`이어야 합니다.

  하지만 이제 `authentication_kerberos` 플러그인으로 대소문자를 구분하지 않는 사용자 이름 비교를 사용할 수 있습니다. 자세한 내용은 8단계: (선택 사항) 대소문자를 구분하지 않는 사용자 이름 비교 구성 단원을 참조하십시오.
- 기능을 켠 후에는 리더 DB 인스턴스를 재부팅하여 `authentication_kerberos` 플러그인을 설치해야 합니다.
- `authentication_kerberos` 플러그인을 지원하지 않는 DB 인스턴스로 복제할 경우 복제가 실패할 수 있습니다.
- Aurora 글로벌 데이터베이스에서 Kerberos 인증을 사용하려면 글로벌 데이터베이스의 모든 DB 클러스터에 대해 인증을 구성해야 합니다.
- 도메인 이름은 62자 미만이어야 합니다.
- Kerberos 인증을 켠 후에 DB 클러스터 포트를 수정하면 안 됩니다. 포트를 수정하면 Kerberos 인증이 더 이상 작동하지 않습니다.

Aurora MySQL DB 클러스터에 대해 Kerberos 인증 설정

AWS Managed Microsoft AD를 사용하여 Aurora MySQL DB 클러스터에 대해 Kerberos 인증을 설정할 수 있습니다. Kerberos 인증을 설정하려면 다음 단계를 수행하십시오.

주제

- [6단계: Kerberos 인증을 사용하는 Aurora MySQL 사용자 생성]
- [7단계: MySQL 클라이언트 구성]
- [8단계: (선택 사항) 대소문자를 구분하지 않는 사용자 이름 비교 구성]

1단계: AWS Managed Microsoft AD를 사용하여 디렉터리 생성

AWS Directory Service는 AWS 클라우드에서 완전 관리형 Microsoft Active Directory를 생성합니다. AWS Managed Microsoft AD 디렉터리를 생성할 때 AWS Directory Service에서 두 개의 도메인 컨트롤러 및 Domain Name System(DNS) 서버가 자동으로 생성됩니다. 디렉터리 서버는 VPC 내 다른 서브넷에서 생성됩니다. 이러한 중복으로 인해 장애가 발생해도 디렉터리에 액세스할 수 있습니다.

AWS Managed Microsoft AD 디렉터리를 생성하는 경우 AWS Directory Service에서 다음 작업이 자동으로 수행됩니다.

- VPC 내에서 Active Directory를 설정합니다.
- 사용자 이름 `Admin`과 지정된 암호를 사용하여 디렉터리 관리자 계정을 생성합니다. 이 계정을 사용하여 디렉터리를 관리할 수 있습니다.

> **Note**
>
> 이 암호를 저장하십시오. AWS Directory Service에서는 저장되지 않습니다. 재설정은 가능하지만 검색은 불가능합니다.

- 디렉터리 컨트롤러에 대한 보안 그룹을 만듭니다.

AWS Managed Microsoft AD를 시작하면 AWS에서 모든 디렉터리의 객체를 포함하는 OU(조직 단위)를 생성합니다. 이 OU는 디렉터리를 만들 때 입력한 NetBIOS 이름으로 되어 있습니다. 이 OU는 AWS가 소유하고 관리하는 도메인 루트 내에 있습니다.

`Admin` 디렉터리를 사용하여 생성한 AWS Managed Microsoft AD 계정은 다음을 포함한 OU의 가장 일반적인 관리 활동에 대한 권한이 있습니다.

- 사용자 생성, 업데이트 또는 삭제
- 도메인(예: 파일 또는 인쇄 서버)에 리소스를 추가한 다음 OU 내의 사용자에 해당 리소스에 대한 권한 할당
- 추가 OU 및 컨테이너 생성
- 권한 위임

- Active Directory 휴지통에서 삭제된 객체 복원
- Active Directory 웹 서비스에서 AD 및 DNS Windows PowerShell 모듈 실행

또한 `Admin` 계정은 다음 도메인 차원 활동을 수행할 권한이 있습니다.

- DNS 구성 관리(레코드, 영역 및 전달자 추가, 제거 또는 업데이트)
- DNS 이벤트 로그 보기
- 보안 이벤트 로그 보기

AWS Managed Microsoft AD으로 디렉터리를 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/directoryservicev2/에서 AWS Directory Service 콘솔을 엽니다.
2. 탐색 창에서 디렉터리를 선택한 후 디렉터리 설정을 선택합니다.
3. AWS Managed Microsoft AD를 선택합니다. AWS Managed Microsoft AD는 현재 Amazon RDS와 함께 사용할 수 있는 유일한 옵션입니다.
4. 다음 정보를 입력합니다.

   디렉터리 DNS 이름

      디렉터리를 위한 정규화된 이름(예: **`corp.example.com`**)입니다.

   디렉터리 NetBIOS 이름

      디렉터리의 짧은 이름(예: **`CORP`**)입니다.

   디렉터리 설명

      (선택 사항) 디렉터리에 대한 설명입니다.

   관리자 암호

      디렉터리 관리자의 암호입니다. 디렉터리 생성 프로세스에서는 사용자 이름 Admin과 이 암호를 사용하여 관리자 계정을 생성합니다.

      디렉터리 관리자 암호는 "admin"이라는 단어를 포함할 수 없습니다. 암호는 대소문자를 구분하며 길이가 8~64자 사이여야 합니다. 또한 다음 네 범주 중 세 개에 해당하는 문자를 1자 이상 포함해야 합니다.

      - 소문자(a–z)

- 대문자(A–Z)
- 숫자(0–9)
- 영숫자 외의 특수 문자(~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

[Confirm Password]

관리자 암호가 다시 입력됩니다.

5. 다음을 선택합니다.
6. 네트워킹 섹션에 다음 정보를 입력하고 다음을 선택합니다.

VPC

디렉터리에 대한 VPC입니다. 이 동일한 VPC에서 Aurora MySQL 클러스터를 생성합니다.

서브넷

디렉터리 서버에 대한 서브넷입니다. 두 서브넷이 서로 다른 가용 영역에 있어야 합니다.

7. 디렉터리 정보를 검토하고 필요한 사항을 변경합니다. 정보가 올바르면 디렉터리 생성을 선택합니다.

디렉터리를 생성하는 데 몇 분 정도 걸립니다. 디렉터리가 성공적으로 생성되면 상태 값이 활성으로 변경됩니다.

디렉터리에 대한 정보를 보려면 디렉터리 목록에서 해당 디렉터리 이름을 선택합니다. Aurora MySQL DB 클러스터를 생성하거나 수정할 때 이 값이 필요하므로 디렉터리 ID 값을 기록해 두십시오.

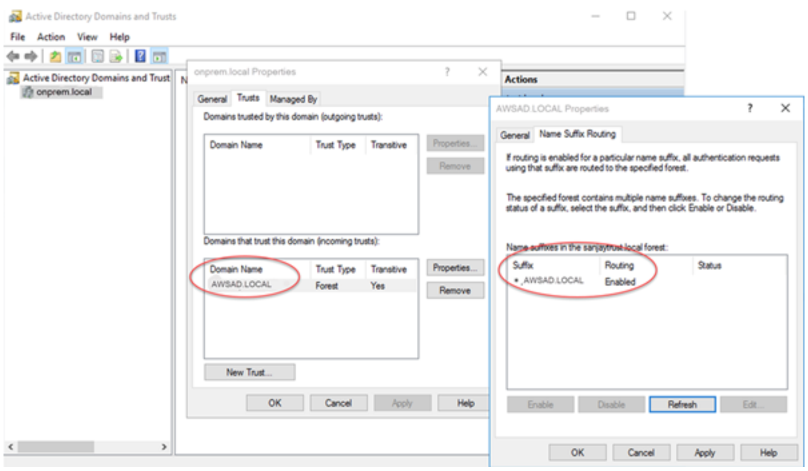
2단계: (선택 사항) 온프레미스 Active Directory에 대한 신뢰 생성

자체 온프레미스 Microsoft Active Directory를 사용하지 않으려는 경우 3단계: Amazon Aurora에서 사용할 IAM 역할 만들기로 건너뜁니다.

온프레미스 Active Directory로 Kerberos 인증을 사용하려면 온프레미스 Microsoft Active Directory와 AWS Managed Microsoft AD에서 만든 1단계: AWS Managed Microsoft AD를 사용하여 디렉터리 생성 디렉터리 간에 forest trust를 사용하여 신뢰 도메인 관계를 생성해야 합니다. 신뢰는 AWS Managed Microsoft AD 디렉터리가 온프레미스 Microsoft Active Directory를 신뢰하는 단방향일 수도 있고, 두 Active Directory가 서로를 신뢰하는 양방향일 수도 있습니다. AWS Directory Service를 사용하여 신뢰를 설정하는 방법에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 신뢰 관계를 생성해야 하는 경우를 참조하세요.

> **Note**
>
> 온프레미스 Microsoft Active Directory를 사용하는 경우:
>
> - Windows 클라이언트가 Aurora 사용자 지정 엔드포인트를 사용하여 연결할 수 없습니다. 자세한 내용은 Amazon Aurora 엔드포인트 연결를 참조하세요.
> - 글로벌 데이터베이스의 경우:
>   - Windows 클라이언트가 글로벌 데이터베이스의 기본 AWS 리전에 있는 인스턴스 엔드포인트 또는 클러스터 엔드포인트만 사용하여 연결할 수 있습니다.
>   - Windows 클라이언트가 보조 AWS 리전의 클러스터 엔드포인트를 사용하여 연결할 수 없습니다.

온프레미스 Microsoft Active Directory 도메인 이름에 새로 만든 신뢰 관계에 해당하는 DNS 접미사 라우팅이 포함되어 있는지 확인합니다. 다음 스크린샷은 예를 보여줍니다.



3단계: Amazon Aurora에서 사용할 IAM 역할 만들기

AWS Directory Service를 호출하는 Amazon Aurora의 경우 계정에 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 AWS Identity and Access Management(IAM) 역할이 필요합니다. 이 역할을 사용하여 Aurora에서 AWS Directory Service를 자동으로 호출할 수 있습니다.

AWS Management Console을 사용하여 DB 클러스터를 생성할 경우 `iam:CreateRole` 권한이 있으면 콘솔에서 이 역할을 자동으로 생성합니다. 이 경우 역할 이름은 `rds-directoryservice-kerberos-access-role`입니다. 그렇지 않으면 IAM 역할을 수동으로 생성해야 합니다.

이 IAM 역할을 생성할 때 `Directory Service`를 선택하고 여기에 AWS 관리형 정책인 `AmazonRDSDirectoryServiceAccess`를 연결합니다.

서비스에 대한 IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성]을 참조하세요.

선택 사항으로 관리형 IAM 정책인 `AmazonRDSDirectoryServiceAccess`를 사용하는 대신 필요한 권한으로 정책을 생성할 수 있습니다. 이 경우 IAM 역할에 다음과 같은 IAM 신뢰 정책이 있어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

또한 역할에는 다음과 같은 IAM 역할 정책도 있어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
```

```
            "ds:UnauthorizeApplication",
            "ds:GetAuthorizedApplicationDetails"
          ],
        "Effect": "Allow",
        "Resource": "*"
        }
      ]
    }
```

4단계: 사용자 생성 및 구성

Active Directory Users and Computers 도구를 사용하여 사용자를 생성할 수 있습니다. 이 도구는 Active Directory Domain Services 및 Active Directory Lightweight Directory Services 도구에 포함되어 있습니다. 사용자는 디렉터리에 액세스할 수 있는 개별 사용자 또는 개체를 나타냅니다.

AWS Directory Service 디렉터리에서 사용자를 생성하려면 AWS Directory Service 디렉터리에 조인된 Microsoft Windows를 기반으로 한 온프레미스 또는 Amazon EC2 인스턴스를 사용합니다. 사용자를 생성할 수 있는 권한을 가진 인스턴스에 로그인해야 합니다. 자세한 내용은 AWS Managed Microsoft AD Directory Service 관리 가이드에서 AWS의 사용자 및 그룹 관리를 참조하세요.

5단계: Aurora MySQL DB 클러스터 생성 또는 수정

디렉터리에서 사용할 Aurora MySQL DB 클러스터를 생성하거나 수정합니다. 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 디렉터리에 연결할 수 있습니다. 이 태스크를 다음 중 한 가지 방법으로 수행할 수 있습니다.

- 콘솔, create-db-cluster CLI 명령 또는 CreateDBCluster RDS API 작업을 사용하여 새 Aurora MySQL DB 클러스터를 생성합니다.

  지침은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.
- 콘솔, modify-db-cluster CLI 명령 또는 ModifyDBCluster RDS API 작업을 사용하여 기존 Aurora MySQL DB 클러스터를 수정합니다.

  지침은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.
- 콘솔, restore-db-cluster-from-snapshot CLI 명령 또는 RestoreDBClusterFromSnapshot RDS API 작업을 사용하여 DB 스냅샷에서 Aurora MySQL DB 클러스터를 복원합니다.

  지침은 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.

- 콘솔, restore-db-cluster-to-point-in-time CLI 명령 또는 RestoreDBClusterToPointInTime RDS API 작업을 사용하여 Aurora MySQL DB 클러스터를 특정 시점으로 복원합니다.

  지침은 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요.

Kerberos 인증은 VPC의 Aurora MySQL DB 클러스터에 대해서만 지원됩니다. DB 클러스터는 디렉터리와 동일한 VPC 또는 다른 VPC에 있을 수 있습니다. DB 클러스터의 VPC에는 디렉터리와의 아웃바운드 통신을 허용하는 VPC 보안 그룹이 있어야 합니다.

콘솔

콘솔을 사용하여 DB 클러스터를 생성, 수정 또는 복원하는 경우 [데이터베이스 인증(Database authentication)] 섹션에서 [Kerberos 인증(Kerberos authentication)]을 선택합니다. 디렉터리 찾아보기를 선택한 다음 디렉터리를 선택하거나 새 디렉터리 생성을 선택합니다.

AWS CLI

콘솔, AWS CLI 또는 RDS API를 사용할 경우 DB 클러스터를 디렉터리에 연결합니다. DB 클러스터에서 생성된 도메인 디렉터리를 사용하려면 다음과 같은 파라미터가 필요합니다.

- `--domain` 파라미터의 경우 디렉터리를 만들 때 생성된 도메인 식별자("d-*" 식별자)를 사용하세요.
- `--domain-iam-role-name` 파라미터의 경우 귀하가 생성한, 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 역할을 사용하십시오.

예를 들어, 다음 CLI 명령은 디렉터리를 사용하도록 DB 클러스터를 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> DB 클러스터를 수정하여 Kerberos 인증을 켜는 경우 변경 후 DB 클러스터를 재부팅합니다.

6단계: Kerberos 인증을 사용하는 Aurora MySQL 사용자 생성

DB 클러스터는 AWS Managed Microsoft AD 도메인에 조인됩니다. 따라서 도메인 내 Active Directory 사용자에서 Aurora MySQL 사용자를 생성할 수 있습니다. 데이터베이스 권한은 이러한 사용자에서 부여 및 취소되는 표준 Aurora MySQL 권한을 통해 관리됩니다.

Active Directory 사용자가 Aurora MySQL에서 인증을 하도록 허용할 수 있습니다. 이렇게 하려면 먼저 다른 DB 클러스터의 경우와 마찬가지로 Amazon RDS 기본 사용자 자격 증명을 사용하여 MySQL DB 인스턴스에 연결합니다. 로그인한 후에는 다음과 같이 Aurora MySQL 내에서 Kerberos 인증을 활용하여 외부에서 인증된 사용자를 생성합니다.

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- *user_name*을 사용자 이름으로 바꿉니다. 이제 도메인의 사용자(사람 및 애플리케이션)는 Kerberos 인증을 사용하여 도메인이 조인된 클라이언트 컴퓨터의 DB 클러스터에 연결할 수 있습니다.
- *host_name*을 호스트 이름으로 바꿉니다. %를 와일드카드로 사용할 수 있습니다. 호스트 이름에 특정 IP 주소를 사용할 수도 있습니다.
- *realm_name*을 도메인의 디렉터리 영역 이름으로 바꿉니다. 영역 이름은 일반적으로 DNS 도메인 이름과 동일하며 `CORP.EXAMPLE.COM`처럼 대문자로 표시됩니다. 영역이란 같은 Kerberos 키 배포 센터를 사용하는 시스템 그룹입니다.

다음 예시에서는 `Admin`이라는 이름의 데이터베이스 사용자를 생성합니다. 이 사용자는 영역 이름이 `MYSQL.LOCAL`인 Active Directory에 대해 인증합니다.

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

기존 Aurora MySQL 로그인 수정

다음 구문을 사용하여 Kerberos 인증을 사용하도록 기존 Aurora MySQL 로그인을 수정할 수도 있습니다.

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

7단계: MySQL 클라이언트 구성

MySQL 클라이언트를 구성하려면 다음 단계를 수행합니다.

1. 도메인을 가리키도록 `krb5.conf` 파일(또는 동등한 파일)을 생성합니다.
2. 클라이언트 호스트와 AWS Directory Service 간에 트래픽이 흐를 수 있는지 확인합니다. Netcat과 같은 네트워크 유틸리티를 사용하여 다음을 수행합니다.
   - 포트 53의 DNS를 통한 트래픽을 확인합니다.
   - 포트 53 및 AWS Directory Service용 포트 88 및 464를 포함하는 Kerberos의 TCP/UDP를 통한 트래픽을 확인합니다.
3. 데이터베이스 포트를 통해 클라이언트 호스트와 DB 인스턴스 간에 트래픽이 흐를 수 있는지 확인합니다. 예를 들어, `mysql`을 사용하여 데이터베이스에 연결하고 액세스합니다.

다음은 AWS Managed Microsoft AD의 샘플 `krb5.conf` 콘텐츠입니다.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

다음은 온프레미스 Microsoft Active Directory의 샘플 `krb5.conf` 콘텐츠입니다.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
```

```
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

8단계: (선택 사항) 대소문자를 구분하지 않는 사용자 이름 비교 구성

기본적으로 MySQL 데이터베이스 사용자 이름의 대소문자는 Active Directory 로그인 시 이용하는 사용자 이름의 대소문자와 일치해야 합니다. 하지만 이제 `authentication_kerberos` 플러그인으로 대소문자를 구분하지 않는 사용자 이름 비교를 사용할 수 있습니다. 이렇게 하려면 `authentication_kerberos_caseins_cmp` DB 클러스터 파라미터를 `true`로 설정합니다.

대소문자를 구분하지 않는 사용자 이름 비교를 사용하려면

1. 사용자 지정 DB 클러스터 파라미터 그룹을 만듭니다. Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성의 절차를 따르십시오.
2. 새 파라미터 그룹을 편집하여 `authentication_kerberos_caseins_cmp` 값을 `true`로 설정합니다. Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정의 절차를 따르십시오.
3. DB 클러스터 파라미터 그룹을 Aurora MySQL DB 클러스터와 연결합니다. Amazon Aurora에서 DB 클러스터 파라미터 그룹과 DB 클러스터 연결의 절차를 따르십시오.
4. DB 클러스터를 재부팅합니다.

Kerberos 인증을 사용하여 Aurora MySQL에 연결

오류를 방지하려면 Unix 플랫폼에서는 버전 8.0.26 이상, Windows에서는 8.0.27 이상의 MySQL 클라이언트를 사용합니다.

Aurora MySQL Kerberos 로그인을 사용하여 DB 클러스터에 연결

Kerberos 인증을 사용하여 Aurora MySQL에 연결하려면 6단계: Kerberos 인증을 사용하는 Aurora MySQL 사용자 생성의 지침에 따라 생성한 데이터베이스 사용자로 로그인해야 합니다.

명령 프롬프트에서 Aurora MySQL DB 클러스터와 연결된 엔드포인트 중 하나에 연결합니다. 암호를 입력하라는 메시지가 표시되면 해당 사용자 이름과 연결된 Kerberos 암호를 입력합니다.

Kerberos로 인증할 때 티켓 부여 티켓(TGT)이 없으면 새로 생성됩니다. `authentication_kerberos` 플러그인은 TGT를 사용하여 서비스 티켓을 가져오고, 이 티켓은 Aurora MySQL 데이터베이스 서버에 제공됩니다.

MySQL 클라이언트를 사용하면 Windows 또는 Unix를 사용하는 Kerberos 인증을 통해 Aurora MySQL에 연결할 수 있습니다.

Unix

다음 방법 중 하나를 사용하여 연결할 수 있습니다.

- TGT를 수동으로 가져옵니다. 이 경우 암호를 MySQL 클라이언트에 제공할 필요가 없습니다.
- Active Directory 로그인을 위한 암호를 MySQL 클라이언트에 직접 입력합니다.

클라이언트 측 플러그인은 MySQL 클라이언트 버전 8.0.26 이상을 위한 Unix 플랫폼에서 지원됩니다.

TGT를 수동으로 가져와서 연결하려면

1. 명령줄 인터페이스에서 다음 명령을 사용하여 TGT를 가져옵니다.

```
kinit user_name
```

2. 다음 `mysql` 명령을 사용하여 DB 클러스터의 DB 인스턴스 엔드포인트에 로그인합니다.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> DB 인스턴스에서 keytab을 교체하면 인증이 실패할 수 있습니다. 이 경우 `kinit`를 다시 실행하여 새 TGT를 가져옵니다.

직접 연결하려면

1. 명령줄 인터페이스에서 다음`mysql` 명령을 사용하여 DB 클러스터의 DB 인스턴스 엔드포인트에 로그인합니다.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. Active Directory 사용자용 암호를 입력합니다.

Windows

Windows에서는 일반적으로 로그인 시 인증이 수행되므로 Aurora MySQL DB 클러스터에 연결하기 위해 TGT를 수동으로 가져올 필요가 없습니다. 데이터베이스 사용자 이름의 대소문자는 Active Directory에 있는 사용자 이름의 대소문자와 일치해야 합니다. 예를 들어 Active Directory의 사용자가 Admin으로 표시될 경우 데이터베이스 사용자 이름도 Admin이어야 합니다.

클라이언트 측 플러그인은 MySQL 클라이언트 버전 8.0.27 이상을 위한 Windows 플랫폼에서 지원됩니다.

직접 연결하려면

- 명령줄 인터페이스에서 다음mysql 명령을 사용하여 DB 클러스터의 DB 인스턴스 엔드포인트에 로그인합니다.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

Aurora를 글로벌 데이터베이스를 사용하는 Kerberos 인증

Aurora MySQL에 대한 Kerberos 인증이 Aurora 글로벌 데이터베이스에 대해 지원됩니다. 기본 DB 클러스터의 Active Directory를 사용하여 보조 DB 클러스터의 사용자를 인증하려면 Active Directory를 보조 AWS 리전에 복제합니다. 기본 클러스터와 동일한 도메인 ID를 사용하여 보조 클러스터에서 Kerberos 인증을 설정합니다. AWS Managed Microsoft AD 복제는 Active Directory의 Enterprise 버전에서만 지원됩니다. 자세한 내용은 AWS Directory Service 관리 안내서의 다중 리전 복제를 참조하세요.

RDS for MySQL에서 Aurora MySQL로 마이그레이션

Kerberos 인증을 활성화한 상태에서 RDS for MySQL에서 Aurora MySQL로 마이그레이션한 후에는 auth_pam 플러그인을 사용하여 생성된 사용자가 authentication_kerberos 플러그인을 사용하도록 수정합니다. 예:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

티켓 캐싱 방지

MySQL 클라이언트 애플리케이션이 시작될 때 유효한 TGT가 존재하지 않을 경우 애플리케이션은 TGT를 가져와 캐시할 수 있습니다. TGT가 캐시되는 것을 방지하려면 /etc/krb5.conf 파일에 구성 파라미터를 설정합니다.

> **Note**
> 이 구성은 Windows가 아닌 Unix를 실행하는 클라이언트 호스트에만 적용됩니다.

TGT 캐싱을 방지하려면

- 다음과 같이 [appdefaults] 섹션을 /etc/krb5.conf에 추가합니다.

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

Kerberos 인증에 대한 로깅

AUTHENTICATION_KERBEROS_CLIENT_LOG 환경 변수는 Kerberos 인증의 로깅 수준을 설정합니다. 로그를 클라이언트 측 디버깅에 사용할 수 있습니다.

허용되는 값은 1~5입니다. 로그 메시지는 표준 오류 출력에 기록됩니다. 다음 표에는 각 로깅 수준이 설명되어 있습니다.

로깅 수준	설명
1 또는 설정되지 않음	로깅 없음
2	오류 메시지
3	오류 및 경고 메시지
4	오류, 경고, 정보 메시지
5	오류, 경고, 정보, 디버그 메시지

도메인에서 DB 클러스터 관리

AWS CLI 또는 RDS API를 사용하여 DB 클러스터 및 DB 클러스터와 관리형 Active Directory의 관계를 관리할 수 있습니다. 예를 들어 Active Directory를 연결하여 Kerberos 인증을 활성화하고 Active

Directory의 연결을 해제하여 Kerberos 인증을 끌 수 있습니다. 또한 한 Active Directory에서 외부 인증할 DB 클러스터를 다른 도메인으로 이동시킬 수 있습니다.

예를 들어 Amazon RDS API를 사용하여 다음을 수행할 수 있습니다.

- 실패한 멤버십에 대해 Kerberos 인증 켜기를 다시 시도하려면 `ModifyDBInstance` API 작업을 사용하여 현재 멤버십의 디렉터리 ID를 지정합니다.
- 멤버십에 대한 IAM 역할 이름을 업데이트하려면 `ModifyDBInstance` API 작업을 사용하고 현재 멤버십의 디렉터리 ID 및 새 IAM 역할을 지정합니다.
- DB 클러스터에서 Kerberos 인증을 끄려면 `ModifyDBInstance` API 작업을 사용하여 none을 도메인 파라미터로 지정합니다.
- 한 도메인에서 다른 도메인으로 DB 클러스터를 이동하려면 `ModifyDBInstance` API 작업을 사용하여 새 도메인의 도메인 식별자를 도메인 파라미터로 지정합니다.
- 각 DB 클러스터의 멤버십을 나열하려면 `DescribeDBInstances` API 작업을 사용합니다.

도메인 멤버십 이해

DB 클러스터를 생성하거나 수정하고 나면 해당 클러스터는 도메인의 멤버가 됩니다. [describe-db-clusters](#) CLI 명령을 실행하여 DB 클러스터에 대한 도메인 멤버십의 상태를 확인할 수 있습니다. DB 클러스터의 상태는 다음 중 한 가지가 될 수 있습니다.

- `kerberos-enabled` - DB 클러스터의 Kerberos 인증이 켜져 있습니다.
- `enabling-kerberos` - AWS가 이 DB 클러스터에 대한 Kerberos 인증 켜기를 진행 중입니다.
- `pending-enable-kerberos` - 이 DB 클러스터에 대한 Kerberos 인증 켜기가 보류 중입니다.
- `pending-maintenance-enable-kerberos` - AWS는 예약된 다음 유지 관리 기간 동안 DB 인스턴스에 대한 Kerberos 인증을 켜려 합니다.
- `pending-disable-kerberos` - 이 DB 클러스터에 대한 Kerberos 인증 끄기가 보류 중입니다.
- `pending-maintenance-disable-kerberos` - AWS는 예약된 다음 유지 관리 기간 동안 DB 클러스터에 대한 Kerberos 인증을 끄려 합니다.
- `enable-kerberos-failed` - 구성 문제로 인해 AWS가 DB 클러스터에 대해 Kerberos 인증을 켜지 못했습니다. DB 클러스터 modify 명령을 다시 실행하기 전에 구성을 확인하고 수정합니다.
- `disabling-kerberos` - AWS가 이 DB 클러스터에 대한 Kerberos 인증 끄기를 진행 중입니다.

네트워크 연결 문제 또는 잘못된 IAM 역할로 인해 Kerberos 인증 켜기 요청이 실패할 수 있습니다. 예를 들어 DB 클러스터를 생성하거나 기존 DB 클러스터를 수정하는데 Kerberos 인증을 켜려는 시도가

실패한다고 가정합니다. 이 경우 modify 명령을 다시 실행하거나 새로 생성된 DB 클러스터를 수정하여 도메인에 조인합니다.

Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

기존 데이터베이스에서 Amazon Aurora MySQL DB 클러스터로 데이터를 마이그레이션하기 위한 여러 가지 옵션이 있습니다. 마이그레이션할 데이터베이스와 데이터 크기에 따라서도 마이그레이션 옵션이 달라집니다.

마이그레이션에는 물리적 마이그레이션과 논리적 마이그레이션이라는 두 가지 유형이 있습니다. 물리적 마이그레이션이란 데이터베이스 파일의 물리적 사본을 사용하여 데이터베이스를 마이그레이션함을 뜻합니다. 논리적 마이그레이션은 삽입, 업데이트, 삭제 같은 논리적인 데이터베이스 변경을 적용하여 마이그레이션이 이루어짐을 뜻합니다.

물리적 마이그레이션의 장점은 다음과 같습니다.

- 물리적 마이그레이션은 특히 대규모 데이터베이스의 경우, 논리적 마이그레이션보다 빠릅니다.
- 물리적 마이그레이션을 위해 백업을 생성할 때 데이터베이스 성능이 저하되지 않습니다.
- 물리적 마이그레이션은 복잡한 데이터베이스 구성 요소를 포함하여 원본 데이터베이스의 모든 것을 마이그레이션할 수 있습니다.

물리적 마이그레이션의 제한은 다음과 같습니다.

- `innodb_page_size` 파라미터를 기본값(16KB)으로 설정해야 합니다.
- `innodb_data_file_path` 파라미터는 기본 데이터 파일 이름 `"ibdata1:12M:autoextend"`를 사용하는 데이터 파일 하나만 사용하여 구성해야 합니다. 두 개의 데이터 파일이 있거나 다른 이름의 데이터 파일이 있는 데이터베이스는 이 방법을 사용하여 마이그레이션할 수 없습니다.

  다음은 허용되지 않는 파일 이름의 예입니다. `"innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend"` 및 `"innodb_data_file_path=ibdata01:50M:autoextend"`.
- `innodb_log_files_in_group` 파라미터를 기본값(2)으로 설정해야 합니다.

논리적 마이그레이션의 장점은 다음과 같습니다.

- 특정 테이블이나 테이블의 일부와 같은 데이터베이스 하위 집합을 마이그레이션할 수 있습니다.
- 물리적 스토리지 구조에 상관없이 데이터를 마이그레이션할 수 있습니다.

논리적 마이그레이션의 제한은 다음과 같습니다.

- 논리적 마이그레이션은 일반적으로 물리적 마이그레이션보다 느립니다.

- 복잡한 데이터베이스 구성 요소는 논리적 마이그레이션 프로세스의 속도를 늦출 수 있습니다. 경우에 따라 복잡한 데이터베이스 구성 요소가 논리적 마이그레이션을 차단할 수도 있습니다.

다음 표에서는 사용자의 옵션과 각 옵션에 따른 마이그레이션 유형을 설명합니다.

마이그레이션 원본	[Migration type]	솔루션
RDS for MySQL DB 인스턴스	물리적	먼저 MySQL DB 인스턴스의 Aurora MySQL 읽기 전용 복제본을 생성하여 RDS for MySQL DB 인스턴스에서 마이그레이션할 수 있습니다. MySQL DB 인스턴스와 Aurora MySQL 읽기 전용 복제본 간 복제 지연이 0이라면 클라이언트 애플리케이션이 Aurora 읽기 전용 복제본에서 데이터를 가져오다가 읽기 또는 쓰기 작업이 있을 때는 복제를 중단하고 Aurora MySQL 읽기 전용 복제본을 독립형 Aurora MySQL DB 클러스터로 사용하도록 지정할 수 있습니다. 자세한 내용은 단원을 참조하십시오[Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션]().
RDS for MySQL DB 스냅샷	물리적	RDS for MySQL DB 스냅샷의 데이터를 Amazon Aurora MySQL DB 클러스터로 직접 마이그레이션할 수 있습니다. 세부 정보는 [RDS for MySQL 스냅샷을 Aurora로 마이그레이션]()을 참조하세요.
Amazon RDS 외부의 MySQL 데이터베이스	논리적	`mysqldump` 유틸리티로 데이터 덤프를 생성하고 해당 데이터를 기존 Amazon Aurora MySQL DB 클러스터로 가져올 수 있습니다. 세부 정보는 [mysqldump를 사용하여 MySQL에서 Amazon Aurora MySQL로 논리적 마이그레이션]()을 참조하세요. 외부 MySQL 데이터베이스에서 마이그레이션하는 동안 데이터베이스 사용자의

마이그레이션 원본	[Migration type]	솔루션
		메타데이터를 내보내려면 `mysqldump` 대신 MySQL Shell 명령도 사용할 수 있습니다. 자세한 내용은 Instance Dump Utility, Schema Dump Utility, and Table Dump Utility를 참조하세요. **Note** mysqlpump 유틸리티는 MySQL 8.0.34부터 더 이상 사용되지 않습니다.
Amazon RDS 외부의 MySQL 데이터베이스	물리적	데이터베이스에서 Amazon Simple Storage Service(Amazon S3) 버킷으로 백업 파일을 복사한 후 해당 파일에서 Amazon Aurora MySQL DB 클러스터를 복원할 수 있습니다. `mysqldump` 를 사용하여 데이터를 마이그레이션하는 것보다 이 방법이 훨씬 더 빠를 것입니다. 자세한 내용은 단원을 참조하십시오Percona XtraBackup과 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션
Amazon RDS 외부의 MySQL 데이터베이스	논리적	데이터베이스의 데이터를 텍스트 파일로 저장하고 해당 파일을 Amazon S3 버킷에 복사할 수 있습니다. 그런 다음 `LOAD DATA FROM S3` MySQL 명령을 사용하여 기존 Aurora MySQL DB 클러스터에 해당 데이터를 로드할 수 있습니다. 자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 단원을 참조하십시오.

마이그레이션 원본	[Migration type]	솔루션
MySQL과 호환되지 않는 데이터베이스	논리적	AWS Database Migration Service(AWS DMS)을 사용하여 MySQL 비호환성 데이터베이스의 데이터를 마이그레이션할 수 있습니다. AWS DMS에 대한 자세한 내용은 AWS Database Migration Service란 무엇입니까?를 참조하세요.

Note

Amazon RDS 외부의 MySQL 데이터베이스를 마이그레이션하는 경우, 표에 설명된 마이그레이션 옵션은 데이터베이스가 InnoDB 또는 MyISAM 테이블스페이스를 지원하는 경우에만 지원됩니다.

Aurora MySQL로 마이그레이션하는 MySQL 데이터베이스가 `memcached`를 사용하는 경우 마이그레이션 전에 `memcached`를 제거하세요.

8.0.11, 8.0.13 및 8.0.15 등의 일부 이전 MySQL 8.0 버전에서는 Aurora MySQL 버전 3.05 이상으로 마이그레이션할 수 없습니다. 마이그레이션하기 전에 MySQL 버전 8.0.28로 업그레이드하는 것이 좋습니다.

외부 MySQL 데이터베이스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션

InnoDB 또는 MyISAM 테이블스페이스를 지원하는 데이터베이스인 경우 다음과 같은 옵션으로 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션할 수 있습니다.

- `mysqldump` 유틸리티로 데이터 덤프를 생성하고 해당 데이터를 기존 Amazon Aurora MySQL DB 클러스터로 가져올 수 있습니다. 자세한 내용은 mysqldump를 사용하여 MySQL에서 Amazon Aurora MySQL로 논리적 마이그레이션 단원을 참조하십시오.
- 데이터베이스에서 Amazon S3 버킷으로 전체 및 증분 백업 파일을 복사한 다음, 해당 파일에서 Amazon Aurora MySQL DB 클러스터로 복원할 수 있습니다. `mysqldump`를 사용하여 데이터를 마이그레이션하는 것보다 이 방법이 훨씬 더 빠를 것입니다. 자세한 내용은 Percona XtraBackup과 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션 단원을 참조하십시오.

주제

Percona XtraBackup과 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션

소스 MySQL 버전 5.7 또는 8.0 데이터베이스에서 Amazon S3 버킷으로 전체 및 증분 백업 파일을 복사할 수 있습니다. 그런 다음 해당 파일에서 동일한 메이저 DB 엔진 버전을 사용하여 Amazon Aurora MySQL DB 클러스터에 복원할 수 있습니다.

`mysqldump`를 사용하여 데이터를 마이그레이션하는 것보다 이 옵션이 훨씬 더 빠를 것입니다. `mysqldump`를 사용하면 명령을 모두 다시 실행하여 소스 데이터베이스의 데이터와 스키마를 새 Aurora MySQL DB 클러스터에 다시 만들기 때문입니다. Aurora MySQL에서는 원본 MySQL 데이터 파일을 복사하는 즉시 해당 파일을 Aurora MySQL DB 클러스터의 데이터로 사용할 수 있습니다.

또한 마이그레이션 프로세스가 진행되는 동안 이진수 로그 복제본을 사용하여 가동 중지 시간을 최소화할 수 있습니다. 이진수 로그 복제를 사용하면, 외부 MySQL 데이터베이스는 데이터가 Aurora MySQL DB 클러스터로 마이그레이션 되는 동안 트랜젝션에 개방된 상태를 유지합니다. Aurora MySQL DB 클러스터를 생성한 후, 이진수 로그 복제를 사용하여 Aurora MySQL DB 클러스터와 백업 이후 발생한 트랜젝션을 동기화합니다. Aurora MySQL DB 클러스터가 MySQL 데이터베이스를 따라잡았을 때, 새 트랜젝션에 대한 Aurora MySQL DB 클러스터로 완전히 전환해 마이그레이션을 완료합니다. 자세한 내용은 복제를 사용하여 Amazon Aurora MySQL DB 클러스터를 MySQL 데이터베이스와 동기화 섹션을 참조하세요.

목차

제한 사항 및 고려 사항

Amazon S3 버킷에서 Amazon Aurora MySQL DB 클러스터로 복원할 때 적용되는 제한 사항 및 고려 사항은 다음과 같습니다.

- 데이터는 기존 DB 클러스터가 아닌 새 DB 클러스터로만 마이그레이션할 수 있습니다.
- Percona XtraBackup을 사용하여 S3에 데이터를 백업해야 합니다. 자세한 내용은 Percona XtraBackup 설치 섹션을 참조하세요.
- Amazon S3 버킷과 Aurora MySQL DB 클러스터는 같은 AWS 리전에 있어야 합니다.
- 다음의 경우에는 복원할 수 없습니다.
  - Amazon S3로 DB 클러스터 스냅샷을 내보냅니다. DB 클러스터 스냅샷 내보내기에서 S3 버킷으로 데이터를 마이그레이션할 수 없습니다.

  - 암호화된 소스 데이터베이스에서는 마이그레이션되는 데이터를 암호화할 수 있습니다. 또한 데이터를 마이그레이션 프로세스 동안 암호화하지 않은 상태로 유지할 수도 있습니다.
  - MySQL 5.5 또는 5.6 데이터베이스
- Percona Server for MySQL은 `mysql` 스키마에 `compression_dictionary*` 테이블을 포함할 수 있으므로 소스 데이터베이스로 지원되지 않습니다.
- Aurora Serverless DB 클러스터로 복원할 수 없습니다.
- 메이저 버전과 마이너 버전에서 역방향 마이그레이션은 지원되지 않습니다. 예를 들어 MySQL 버전 8.0에서 Aurora MySQL 버전 2(MySQL 5.7과 호환), MySQL 버전 8.0.32에서 MySQL 커뮤니티 버전 8.0.26과 호환되는 Aurora MySQL 버전 3.03으로 마이그레이션할 수 없습니다.
- 8.0.11, 8.0.13 및 8.0.15 등의 일부 이전 MySQL 8.0 버전에서는 Aurora MySQL 버전 3.05 이상으로 마이그레이션할 수 없습니다. 마이그레이션하기 전에 MySQL 버전 8.0.28로 업그레이드하는 것이 좋습니다.
- Amazon S3에서 가져오기는 db.t2.micro DB 인스턴스 클래스에서 지원되지 않습니다. 그러나 다른 DB 인스턴스 클래스로 복원한 다음 나중에 DB 인스턴스 클래스를 변경할 수 있습니다. DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.
- Amazon S3는 S3 버킷에 업로드되는 파일 크기를 5TB로 제한합니다. 백업 파일이 5TB를 초과하면 해당 백업 파일을 더 작은 크기의 파일들로 나누어야 합니다.
- Amazon RDS는 S3 버킷에 업로드되는 파일의 개수를 100만 개로 제한합니다. 모든 전체 및 증분 백업을 포함하여 데이터베이스에 대한 백업 데이터가 100만 개의 파일을 초과하는 경우, Gzip(.gz), tar(.tar.gz) 또는 Percona xbstream(.xbstream) 파일을 사용하여 전체 및 증분 백업 파일을 S3 버킷에 저장합니다. Percona Xtrabackup 8.0은 압축에 Percona xbstream만 지원합니다.
- DB 클러스터를 생성할 때는 각 DB 클러스터의 관리 서비스를 위해 `rdsadmin` 사용자가 만들어집니다. 이 경우는 RDS에서 예약된 사용자이므로 다음과 같은 제한 사항이 적용됩니다.
  - `'rdsadmin'@'localhost'`를 정의자로 사용하는 함수, 프로시저, 뷰, 이벤트 및 트리거는 가져올 수 없습니다. 자세한 내용은 'rdsadmin'@'localhost'를 정의자로 사용하여 저장된 객체 및 Amazon Aurora MySQL을 사용한 마스터 사용자 권한(을)를 참조하세요.
  - Aurora MySQL DB 클러스터가 생성되면 지원되는 최대 권한을 지닌 마스터 사용자가 생성됩니다. 백업에서 복원하는 동안, 가져오는 사용자에게 할당된 지원되지 않는 권한은 가져오는 동안 자동으로 제거됩니다.

    이로 인해 영향을 받을 수 있는 사용자를 식별하려면 지원되지 않는 권한을 지닌 사용자 계정 단원을 참조하세요. Aurora MySQL에서 지원되는 권한에 대한 자세한 내용은 역할 기반 권한 모델 단원을 참조하세요.

- Aurora MySQL 버전 3의 경우 동적 권한을 가져올 수 없습니다. Aurora가 지원하는 동적 권한은 마이그레이션 후에 가져올 수 있습니다. 자세한 내용은 Aurora MySQL 버전 3의 동적 권한 섹션을 참조하세요.
- `mysql` 스키마에서 사용자가 생성한 테이블은 마이그레이션되지 않습니다.
- `innodb_data_file_path` 파라미터는 기본 데이터 파일 이름 `ibdata1:12M:autoextend`를 사용하는 데이터 파일 하나만 사용하여 구성해야 합니다. 두 개의 데이터 파일이 있거나 다른 이름의 데이터 파일이 있는 데이터베이스는 이 방법을 사용하여 마이그레이션할 수 없습니다.

  `innodb_data_file_path=ibdata1:50M`, `ibdata2:50M:autoextend`, `innodb_data_file_path=ibdata01:50M:autoextend` 파일 이름은 허용되지 않는 예입니다.
- 기본 MySQL 데이터 디렉터리 외부에서 정의된 테이블이 있는 원본 데이터베이스에서 마이그레이션할 수 없습니다.
- 이 방법을 사용할 때 압축 백업에 지원되는 최대 크기는 현재 64TiB로 제한됩니다. 압축 백업의 경우 압축되지 않은 공간 관련 요구 사항을 고려하여 이 제한 수준이 낮게 적용됩니다. 이 경우 지원되는 최대 백업 크기는 (`64 TiB – compressed backup size`)입니다.
- Aurora MySQL은 MySQL 및 기타 외부 구성 요소 및 플러그인 가져오기를 지원하지 않습니다.
- Aurora MySQL이 데이터베이스에서 모든 것을 복원하는 것은 아닙니다. 소스 MySQL 데이터베이스에서 다음 항목의 데이터베이스 스키마와 값을 저장한 다음, 복원한 Aurora MySQL DB 클러스터가 생성되면 여기에 추가하는 것이 좋습니다.
  - 사용자 계정
  - Functions
  - 저장 프로시저
  - 시간대 정보. 시간대 정보는 Aurora MySQL DB 클러스터의 로컬 운영 체제에서 로드됩니다. 자세한 내용은 Amazon Aurora DB 클러스터의 현지 시간대 섹션을 참조하세요.

시작하기 전 준비 사항

데이터를 Amazon S3 버킷에 복사하고 해당 파일에서 DB 클러스터로 복원하려면 먼저 다음 내용을 따라야 합니다.

- 로컬 서버에 Percona XtraBackup을 설치합니다.
- Aurora MySQL이 Amazon S3 버킷에 대신 액세스하도록 허용합니다.

Percona XtraBackup 설치

Amazon Aurora는 Percona XtraBackup을 사용하여 만든 파일로 DB 클러스터를 복원할 수 있습니다. Percona XtraBackup은 Software Downloads - Percona에서 설치할 수 있습니다.

MySQL 5.7 마이그레이션의 경우 Percona XtraBackup 2.4를 사용합니다.

MySQL 8.0 마이그레이션의 경우 Percona XtraBackup 8.0을 사용합니다. Percona XtraBackup 버전이 소스 데이터베이스의 엔진 버전과 호환되는지 확인하시기 바랍니다.

필수 권한

MySQL 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션하려면 몇 가지 권한이 필요합니다.

- Aurora에서 Amazon S3 버킷에 새 클러스터를 생성하도록 요청하는 사용자는 AWS 계정의 버킷을 나열할 권한이 있어야 합니다. AWS Identity and Access Management(IAM) 정책을 사용하여 사용자에게 이 권한을 부여합니다.
- Aurora는 Amazon Aurora MySQL DB 클러스터를 생성하기 위해 사용된 파일을 저장하는 Amazon S3 버킷에 대신 액세스할 수 있는 권한을 요구합니다. IAM 서비스 역할을 사용하여 Aurora에 필요한 권한을 부여합니다.
- 요청한 사용자에게는 AWS 계정의 IAM 역할을 나열할 권한도 있어야 합니다.
- 요청한 사용자가 IAM 서비스 역할을 만들거나 Aurora에 IAM 서비스 역할 생성(콘솔 사용)을 요청하기 위해서는 AWS 계정의 IAM 역할을 만들 권한이 그 사용자에게 있어야 합니다.
- 마이그레이션 프로세스 동안 데이터를 암호화할 계획이면, 마이그레이션을 수행할 사용자에 대한 IAM 정책을 업데이트, 백업 암호화에 사용하는 AWS KMS keys에 대한 RDS 액세스 권한을 부여합니다. 지침은 AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 섹션을 참조하세요.

예를 들어, 다음 IAM 정책은 콘솔을 사용하여 IAM 역할을 나열하고, IAM 역할을 만들고, 해당 계정의 Amazon S3 버킷을 나열하고, KMS 키를 나열하는 데 필요한 최소한의 권한을 사용자에게 부여합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

또한 사용자가 IAM 역할을 Amazon S3 버킷과 연결하려는 경우, IAM 사용자에게 해당 IAM 역할에 대한 `iam:PassRole` 권한이 있어야 합니다. 관리자는 이 권한으로 사용자가 Amazon S3 버킷에 연결할 수 있는 IAM 역할을 제한하게 됩니다.

예를 들어 다음 IAM 정책은 사용자가 `S3Access`라는 역할을 Amazon S3 버킷과 연결할 수 있도록 허용합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

IAM 사용자 권한에 대한 자세한 내용은 정책을 사용하여 액세스 관리 단원을 참조하십시오.

IAM 서비스 역할 생성

새 역할 생성 옵션을 선택하여 AWS Management Console에서 역할을 생성할 수 있습니다(이 주제 후반부에서 설명). 이 옵션을 선택하고 새 역할의 이름을 지정하면 Aurora는 Aurora가 그 이름으로 Amazon S3 버킷에 액세스하는 데 필요한 IAM 서비스 역할을 생성합니다.

또는 다음 절차에 따라 수동으로 역할을 만들 수도 있습니다.

Aurora가 Amazon S3에 액세스할 수 있도록 IAM 역할을 만들려면

1. Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성의 단계를 수행합니다.
2. Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성의 단계를 수행합니다.
3. IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결의 단계를 수행합니다.

Amazon Aurora MySQL DB 클러스터로 복원할 파일 백업

Percona XtraBackup을 사용하여 MySQL 데이터베이스 파일의 전체 백업을 만들고 백업 파일을 Amazon S3 버킷으로 업로드할 수 있습니다. 또는 이미 Percona XtraBackup을 사용하여 MySQL 데이터베이스 파일을 백업 중인 경우 기존의 전체 및 증분 백업 디렉터리 및 파일을 Amazon S3 버킷으로 업로드할 수 있습니다.

주제

- Percona XtraBackup을 사용한 전체 백업 생성
- Percona XtraBackup을 사용한 증분 백업 사용
- 백업 고려 사항

Percona XtraBackup을 사용한 전체 백업 생성

Amazon S3에서 복원하여 Aurora MySQL DB 클러스터를 생성할 수 있는 MySQL 데이터베이스 파일의 전체 백업을 만들려면 Percona XtraBackup 유틸리티(`xtrabackup`)를 사용하여 데이터베이스를 백업합니다.

예를 들어, 다음 명령을 실행하면 MySQL 데이터베이스 백업을 만들고 `/on-premises/s3-restore/backup` 폴더에 백업 파일을 저장합니다.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/
s3-restore/backup>
```

백업을 파일 하나로 압축하려면(필요하면 분할 가능) `--stream` 옵션을 사용하여 다음 형식 중 하나로 백업을 저장하면 됩니다.

- Gzip(.gz)
- tar(.tar)
- Percona xbstream(.xbstream)

다음 명령을 실행하면 Gzip 파일 여러 개로 된 MySQL 데이터베이스 백업이 만들어집니다.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

다음 명령을 실행하면 tar 파일 여러 개로 된 MySQL 데이터베이스 백업이 만들어집니다.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

다음 명령을 실행하면 xbstream 파일 여러 개로 된 MySQL 데이터베이스 백업이 만들어집니다.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> 다음 오류가 표시되는 경우 명령에서 파일 형식이 혼합된 것이 원인일 수 있습니다.
>
> ```
> ERROR:/bin/tar: This does not look like a tar archive
> ```

Percona XtraBackup 유틸리티를 사용하여 MySQL 데이터베이스를 백업한 뒤에는 백업 디렉터리 및 파일을 Amazon S3 버킷으로 복사할 수 있습니다.

파일을 만들고 Amazon S3 버킷에 업로드하는 방법에 대한 자세한 내용은 Amazon S3 시작 안내서의 [Amazon Simple Storage Service 시작하기]()를 참조하십시오.

Percona XtraBackup을 사용한 증분 백업 사용

Amazon Aurora MySQL은 Percona XtraBackup을 사용하여 만든 전체 및 증분 백업을 모두 지원합니다. 이미 Percona XtraBackup을 사용하여 MySQL 데이터베이스 파일의 전체 및 증분 백업을 수행 중인 경우 전체 백업을 만들고 백업 파일을 Amazon S3로 업로드할 필요가 없습니다. 대신, 전체 및 증분 백업에 대한 기존 백업 디렉터리 및 파일을 Amazon S3 버킷으로 복사하여 시간을 크게 절약할 수 있습니다. 자세한 내용을 알아보려면 Percona 웹 사이트에서 Create an incremental backup을 참조하세요.

기존의 전체 및 증분 백업 파일을 Amazon S3 버킷으로 복사할 때 기본 디렉터리의 콘텐츠를 반복적으로 복사해야 합니다. 이러한 콘텐츠에는 전체 백업이 포함되며 모든 증분 백업 디렉터리 및 파일도 포함됩니다. 이 복사본은 Amazon S3 버킷의 디렉터리 구조를 보관해야 합니다. Aurora는 모든 파일과 디렉터리에서 반복됩니다. Aurora는 각 증분 백업에 포함된 `xtrabackup-checkpoints` 파일을 사용하여 기본 디렉터리를 식별하고 LSN(로그 시퀀스 번호) 범위에 따라 증분 백업을 정렬합니다.

파일을 만들고 Amazon S3 버킷에 업로드하는 방법에 대한 자세한 내용은 Amazon S3 시작 안내서의 Amazon Simple Storage Service 시작하기를 참조하십시오.

백업 고려 사항

Aurora는 Percona XtraBackup을 사용하여 생성되는 부분 백업을 지원하지 않습니다. 데이터베이스의 소스 파일을 백업할 때 `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude` 또는 `--databases-file` 옵션을 사용하여 부분 백업을 만들 수 없습니다.

Percona XtraBackup을 사용한 데이터베이스 백업에 관한 자세한 내용은 Percona 웹 사이트의 Percona XtraBackup - Documentation 및 Work with binary logs를 참조하세요.

Aurora에서는 Percona XtraBackup을 사용한 증분 백업을 지원합니다. 자세한 내용을 알아보려면 Percona 웹 사이트에서 Create an incremental backup을 참조하세요.

Aurora은 파일 이름을 기준으로 백업 파일을 사용합니다. 파일 형식에 따라 백업 파일에 적절한 파일 확장명을 지정해야 합니다—예를 들어, Percona xbstream 형식을 사용하여 저장한 파일에는 `.xbstream`을 지정합니다.

Aurora는 알파벳 순서뿐 아니라 자연수 순서로도 백업 파일을 사용합니다. `split` 명령을 실행할 때는 항상 `xtrabackup` 옵션을 사용하여 적절한 순서로 백업 파일을 작성하고 이름을 붙여야 합니다.

Amazon S3는 Amazon S3 버킷에 업로드되는 파일 크기를 5TB로 제한합니다. 데이터베이스의 백업 데이터가 5TB를 초과하는 경우, `split` 명령을 사용하여 백업 파일을 각각 5TB 미만의 파일 여러 개로 나누어야 합니다.

Aurora는 Amazon S3 버킷에 업로드되는 소스 파일을 100만 개로 제한합니다. 일부 경우, 모든 전체 및 증분 백업을 포함하고 있는 데이터베이스의 백업 데이터가 아주 많은 수의 파일로 구성되어 있을 수 있습니다. 이 경우에는 tarball(.tar.gz) 파일을 사용하여 Amazon S3 버킷에 전체 및 증분 백업 파일을 저장하십시오.

파일을 Amazon S3 버킷으로 업로드할 때, 서버 측 암호화를 사용해 데이터를 암호화할 수 있습니다. 그런 다음 암호화된 파일에서 Amazon Aurora MySQL DB 클러스터를 복원할 수 있습니다. Amazon Aurora MySQL은 다음 서버 측 암호화 유형을 사용하여 암호화된 파일로 DB 클러스터를 복원할 수 있습니다.

- Amazon S3–관리형 키(SSE-S3)를 이용한 서버 측 암호화 – 각 객체가 강력한 멀티 팩터 암호화를 사용하는 고유 키로 암호화 됩니다.
- AWS KMS 관리형 키(SSE-KMS)를 이용한 서버 측 암호화 - SSE-S3와 유사하지만, 스스로 암호화 키를 생성해 관리하는 옵션이 있으며 그 외 다른 점도 있습니다.

파일을 Amazon S3 버킷에 업로드할 때 서버 측 암호화를 사용하는 방법에 대한 자세한 내용은 Amazon S3 개발자 안내서의 [서버 측 암호화를 사용하여 데이터 보호]를 참조하세요.

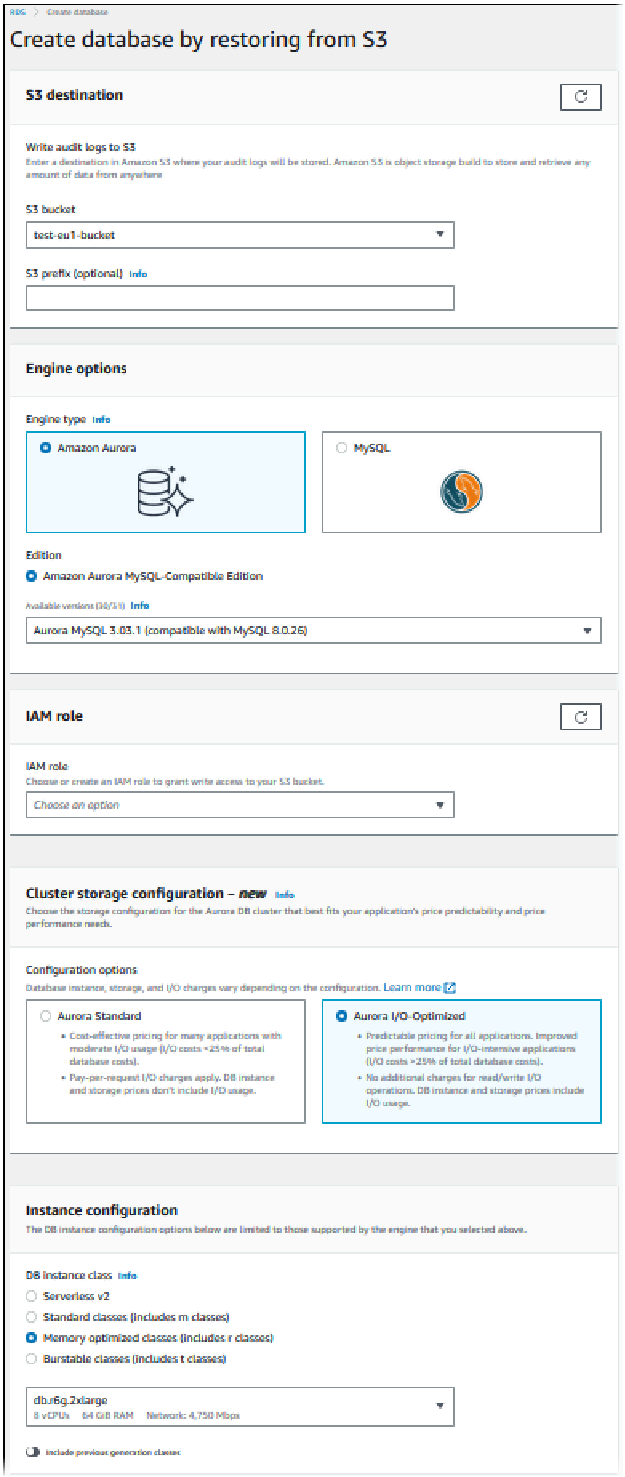
Amazon S3 버킷에서 Amazon Aurora MySQL DB 클러스터 복원

Amazon RDS 콘솔을 사용하여 Amazon S3 버킷의 백업 파일을 복원하여 새 Amazon Aurora MySQL DB 클러스터를 만들 수 있습니다.

Amazon S3 버킷의 파일에서 Amazon Aurora MySQL DB 클러스터를 복원하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 상단에서 DB 클러스터를 생성할 AWS 리전을 선택합니다. 데이터베이스 백업이 포함된 Amazon S3 버킷과 동일한 AWS 리전을 선택합니다.
3. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 S3에서 복원을 선택합니다.
4. S3에서 복원(Restore from S3)을 선택합니다.

   S3에서 복원하여 데이터베이스 생성(Create database by restoring from S3) 페이지가 나타납니다.



5. S3 대상(S3 destination)에서 다음을 수행합니다.

   a. 백업 파일이 포함된 S3 버킷을 선택합니다.

   b. (선택 사항) S3 폴더 경로 접두사에 Amazon S3 버킷에 저장되는 파일의 파일 경로 접두사를 입력합니다.

접두사를 지정하지 않는 경우, RDS는 S3 버킷의 루트 폴더에 있는 모든 파일과 폴더를 사용하여 DB 인스턴스를 만듭니다. 접두사를 지정하는 경우 RDS는 파일의 경로가 지정된 접두사로 시작하는 S3 버킷의 파일과 폴더를 사용하여 DB 인스턴스를 만듭니다.

예를 들어 이름이 backups인 하위 폴더의 S3에 백업 파일을 저장했고 여러 세트의 백업 파일이 각각 자체 디렉터리(gzip_backup1, gzip_backup2 등)에 들어 있다고 가정해봅시다. 이 경우 gzip_backup1 폴더의 파일에서 복원하려면 backups/gzip_backup1의 접두사를 지정합니다.

6. 엔진 옵션(Engine options)에서 다음을 수행합니다.

   a. Engine type(엔진 유형)에서 Amazon Aurora을 선택합니다.

   b. 버전에서 복원된 DB 인스턴스의 Aurora MySQL 엔진 버전을 선택합니다.

7. IAM 역할(IAM role)에서 기존 IAM 역할을 선택할 수 있습니다.

8. 또는 새 역할 생성을 선택하여 새 IAM 역할이 생성되도록 할 수도 있습니다. 이 경우 다음을 수행합니다.

   a. IAM 역할 이름(IAM role name)을 입력합니다.

   b. KMS 키에 대한 액세스 허용(Allow access to KMS key) 여부를 선택합니다.

      • 백업 파일을 암호화하지 않았다면 [No]를 선택합니다.

      • Amazon S3로 백업 파일을 업로드할 때 AES-256(SSE-S3)로 암호화를 했다면 [아니오(No)]를 선택합니다. 이 경우 데이터는 자동으로 복호화됩니다.

      • Amazon S3로 백업 파일을 업로드할 때 AWS KMS(SSE-KMS) 서버 측 암호화로 암호화를 했다면 예를 선택합니다. 그런 다음 AWS KMS key에 대해 올바른 KMS 키를 선택합니다.

        AWS Management Console은 Aurora를 활성화하여 데이터를 복호화하는 IAM 정책을 생성합니다.

      자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용하여 데이터 보호를 참조하십시오.

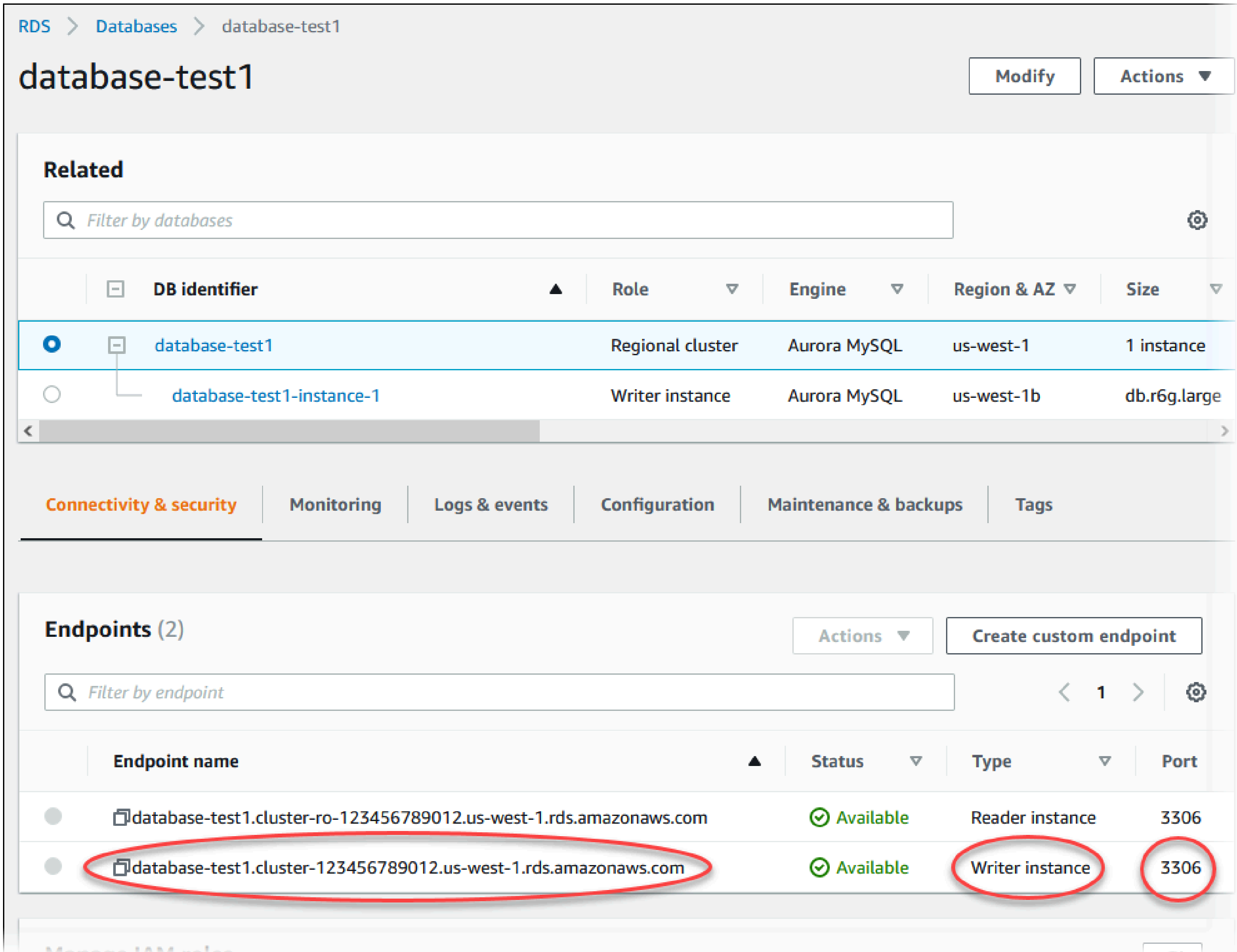
9. DB 클러스터 스토리지 구성, DB 인스턴스 클래스, DB 클러스터 식별자 및 로그인 보안 인증 정보 등 DB 클러스터에 대한 설정을 선택합니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하십시오.

10. 필요에 따라 Aurora MySQL DB 클러스터에 대한 추가 설정을 사용자 지정합니다.

11. 데이터베이스 생성(Database Create) 를 선택하여 Aurora DB 인스턴스를 시작합니다.

Amazon RDS 콘솔의 DB 인스턴스 목록에 새로운 DB 인스턴스가 나타납니다. DB 인스턴스를 만들고 사용할 준비가 될 때까지 DB 인스턴스의 상태는 [creating]입니다. 상태가 [available]로 변경되면 DB 클러스터의 기본 인스턴스에 연결할 수 있습니다. DB 인스턴스 클래스와 할당된 저장소에 따라 새 인스턴스를 사용할 수 있을 때까지 몇 분 정도 걸릴 수 있습니다.

새로 생성된 클러스터를 보려면 Amazon RDS 콘솔에서 Databases(데이터베이스) 보기를 선택하고 DB 클러스터를 선택합니다. 자세한 내용은 Amazon Aurora DB 클러스터 보기 섹션을 참조하세요.



DB 클러스터의 포트와 라이터 엔드포인트를 기록합니다. 쓰기 또는 읽기 작업을 수행하는 애플리케이션은 모두 JDBC 및 ODBC 연결 문자열에 이 DB 클러스터의 라이터 엔드포인트와 포트를 사용합니다.

복제를 사용하여 Amazon Aurora MySQL DB 클러스터를 MySQL 데이터베이스와 동기화

마이그레이션 동안 가동 정지가 거의 없도록 만들기 위해, MySQL 데이터베이스에 커밋된 트랜잭션을 Aurora MySQL DB 클러스터로 복제할 수 있습니다. 복제를 사용하여 DB 클러스터가 마이그레이션 동

안 발생한 MySQL 데이터베이스의 트랜잭션을 따라 잡을 수 있도록 만들 수 있습니다. DB 클러스터가 완전히 따라 잡았을 때, 복제를 중지하고 Aurora MySQL로 마이그레이션을 완료할 수 있습니다.

주제

- 외부 MySQL 데이터베이스와 Aurora MySQL DB 클러스터의 암호화된 복제 구성
- Amazon Aurora MySQL DB 클러스터를 외부 MySQL 데이터베이스와 동기화

외부 MySQL 데이터베이스와 Aurora MySQL DB 클러스터의 암호화된 복제 구성

데이터를 안전하게 복제하기 위해 암호화된 복제를 사용할 수 있습니다.

> **Note**
>
> 암호화된 복제를 사용할 필요가 없다면 이 단계를 건너 뛰고 Amazon Aurora MySQL DB 클러스터를 외부 MySQL 데이터베이스와 동기화의 지시로 이동합니다.

다음은 암호화된 복제를 사용하기 위한 사전 조건입니다.

- 외부 MySQL 소스 데이터베이스에서 SSL(Secure Socket Layer)를 활성화해야 합니다.
- Aurora MySQL DB 클러스터에 대해 클라이언트 키와 클라이언트 인증서를 준비해야 합니다.

암호화 복제 중, Aurora MySQL DB 클러스터는 MySQL 데이터베이스 서버의 클라이언트 역할을 합니다. Aurora MySQL 클라이언트 인증서와 키는 .pem 형식의 파일입니다.

외부 MySQL 데이터베이스와 Aurora MySQL DB 클러스터에 암호화된 복제를 구성하려면,

1. 암호화 복제를 준비해야 합니다.

   - 외부 MySQL 주 데이터베이스에서 SSL을 활성화하지 않았고 클라이언트 키와 클라이언트 인증서가 준비되지 않았다면, MySQL 데이터베이스 서버의 SSL을 활성화하고 필요한 클라이언트 키와 클라이언트 인증서를 생성합니다.
   - 외부 주 데이터베이스에 SSL이 활성화되어 있으면 Aurora MySQL DB 클러스터에 클라이언트 키와 인증서를 제공합니다. 인증서와 키가 없다면, Aurora MySQL DB 클러스터에 대해 새 키와 인증서를 생성합니다. 클라이언트 인증서에 서명하려면, 외부 MySQL 주 데이터베이스의 SSL을 구성할 때 사용하는 인증 기관(CA) 키가 있어야 합니다.

자세한 내용은 MySQL 설명서의 openssl을 사용하여 SSL 인증서 및 키 생성을 참조하십시오.

인증 기관(CA) 인증서, 클라이언트 키, 클라이언트 인증서가 필요합니다.

2. SSL을 사용하여 주 사용자로 Aurora MySQL DB 클러스터에 연결하십시오.

   SSL을 이용한 Aurora MySQL DB 클러스터 연결에 대한 자세한 정보는 Aurora MySQL DB 클러스터에 대한 TLS 연결를 참조하십시오.

3. mysql.rds_import_binlog_ssl_material 저장 프로시저를 실행하여 Aurora MySQL DB 클러스터로 SSL 정보를 가져오십시오.

   `ssl_material_value` 파라미터에서, Aurora MySQL DB 클러스터의 정보를 올바른 JSON 페이로드에 삽입하십시오.

   다음은 Aurora MySQL DB 클러스터에 SSL 정보를 가져오는 예제입니다. .pem 형식 파일은 본문 코드의 길이가 일반적으로 예제의 본문 코드 길이보다 깁니다.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

   자세한 내용은 mysql.rds_import_binlog_ssl_material 및 Aurora MySQL DB 클러스터에 대한 TLS 연결(을)를 참조하세요.

> **Note**
>
> 프로시저를 실행한 후 파일에 암호를 저장합니다. 이 파일을 나중에 삭제하려면 mysql.rds_remove_binlog_ssl_material 저장 프로시저를 실행하면 됩니다.

Amazon Aurora MySQL DB 클러스터를 외부 MySQL 데이터베이스와 동기화

복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 MySQL 데이터베이스를 동기화할 수 있습니다.

복제를 사용하여 Aurora MySQL DB 클러스터와 MySQL 데이터베이스를 동기화하려면,

1. 외부 MySQL 데이터베이스의 /etc/my.cnf 파일에 관련 항목이 있어야 합니다.

   암호화된 복제가 필요 없다면, 이진수 로그(binlogs)를 활성화 시키고 SSL을 비활성화 시켜 외부 MySQL 데이터베이스를 시작합니다. 다음은 암호화된 데이터와 관련된 /etc/my.cnf 파일 항목들입니다.

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   sync_binlog=1
   ```

   암호화된 복제가 필요하다면, SSL과 binlogs를 활성화시켜 외부 MySQL 데이터베이스를 시작합니다. /etc/my.cnf 파일 항목에는 MySQL 데이터베이스 서버의 .pem 파일 위치가 포함되어 있습니다.

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   sync_binlog=1

   # Setup SSL.
   ssl-ca=/home/sslcerts/ca.pem
   ssl-cert=/home/sslcerts/server-cert.pem
   ssl-key=/home/sslcerts/server-key.pem
   ```

다음 명령을 사용하여 SSL이 활성화되었는지 확인할 수 있습니다.

```
mysql> show variables like 'have_ssl';
```

다음과 유사하게 출력되어야 합니다.

```
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
| Variable_name | Value |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
| have_ssl      | YES   |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
1 row in set (0.00 sec)
```

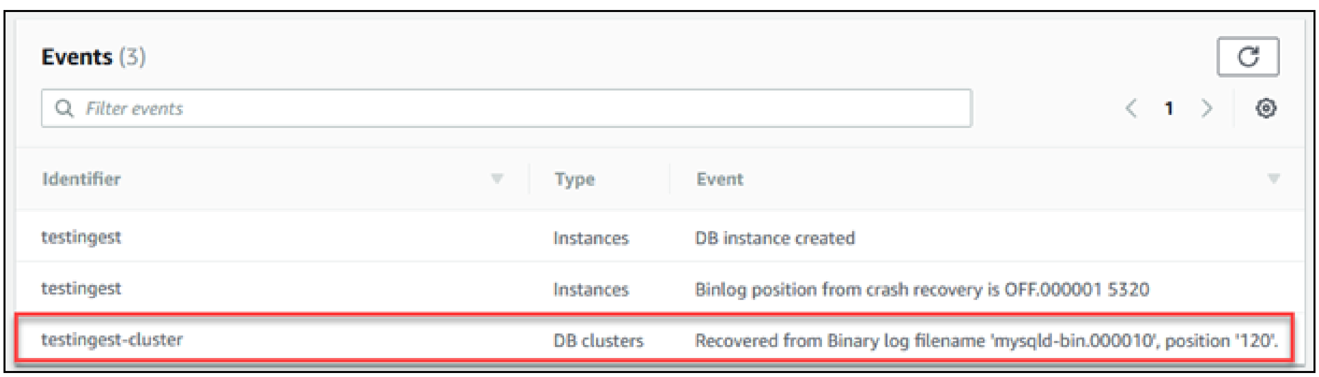
2. 복제에서 시작 이진수 로그 위치를 결정합니다. 다음 단계에서 복제를 시작할 위치를 지정합니다.

   AWS Management Console 사용

   a. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

   b. 탐색 창에서 [Events]를 선택합니다.

   c. [Events] 목록에서 [Recovered from Binary log filename] 이벤트의 위치를 확인합니다.



   AWS CLI 사용

   또한 describe-events AWS CLI 명령을 사용하여 binlog 파일 이름 및 위치를 확인할 수 있습니다. 다음은 `describe-events` 명령의 예시입니다.

```
PROMPT> aws rds describe-events
```

출력에서 binlog 위치를 표시하는 이벤트를 식별합니다.

3. 외부 MySQL 데이터베이스에 연결되어 있는 동안 복제에 사용할 사용자를 만듭니다. 이 계정은 오직 복제용으로만 사용되며 보안 향상을 위해 사용자의 도메인으로 제한되어야 합니다. 다음은 예제입니다.

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>';
```

사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한이 필요합니다. 해당 사용자에게 이 권한을 부여합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

암호화된 복제를 사용해야 한다면, 복제 사용자에게 SSL 연결이 반드시 필요합니다. 예를 들어, 다음 문을 사용하여 사용자 계정 *`<user_name>`*.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> `REQUIRE SSL`이 포함되어 있지 않다면, 복제 연결이 자동으로 암호화되지 않은 연결로 돌아갈 수 있습니다.

4. Amazon RDS 콘솔에서 외부 MySQL 데이터베이스를 호스팅하는 서버의 IP 주소를 Aurora MySQL DB 클러스터용 VPC 보안 그룹에 추가하십시오. VPC 보안 그룹 수정에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [VPC용 보안 그룹](#)을 참조하십시오.

   Aurora MySQL DB 클러스터의 IP 주소에서의 연결을 허용하도록 로컬 네트워크를 구성하여 외부 MySQL 데이터베이스와 통신할 수 있도록 만들어야 할 수도 있습니다. Aurora MySQL DB 클러스터의 IP 주소를 검색하려면 `host` 명령을 사용하십시오.

```
host <db_cluster_endpoint>
```

호스트 이름은 Aurora MySQL DB 클러스터 엔드포인트의 DNS 이름입니다.

5. mysql.rds_reset_external_master(Aurora MySQL 버전 2) 또는 mysql.rds_reset_external_source(Aurora MySQL 버전 3) 저장 프로시저를 실행해 이진수 로그 복제를 활성화합니다. 저장 프로시저에는 다음과 같은 구문이 있습니다.

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);

CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

파라미터에 관한 자세한 내용은 mysql.rds_reset_external_master(Aurora MySQL 버전 2) 및 mysql.rds_reset_external_source(Aurora MySQL 버전 3) 단원을 참조하세요.

`mysql_binary_log_file_name` 및 `mysql_binary_log_file_location`의 경우, 앞에서 확인한 [Recovered from Binary log filename] 이벤트의 위치를 사용합니다.

Aurora MySQL DB 클러스터의 데이터가 암호화되어 있지 않다면, `ssl_encryption` 파라미터를 `0`으로 설정해야 합니다. 데이터가 암호화되어 있으면 `ssl_encryption` 파라미터를 `1`로 설정합니다.

다음은 암호화된 데이터가 있는 Aurora MySQL DB 클러스터에 대해 프로시저를 실행하는 예제입니다.

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

이 저장 프로시저는 Aurora MySQL DB 클러스터가 외부 MySQL 데이터베이스를 연결하고 이진수 로그를 읽을 때 사용하는 파라미터를 설정합니다. 또한 데이터가 암호화되어 있다면, 로컬 디스크에 대한 SSL 인증 기관(CA) 인증서, 클라이언트 인증서, 클라이언트 키를 다운로드합니다.

6. mysql.rds_start_replication 저장 프로시저를 실행해 이진수 로그 복제를 시작합니다.

```
CALL mysql.rds_start_replication;
```

7. Aurora MySQL DB 클러스터가 MySQL 복제 주 데이터베이스보다 얼마나 지연되어 있는지 모니터링하십시오. 이렇게 하려면 Aurora MySQL DB 클러스터에 연결하고 다음 명령을 실행하십시오.

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

명령 출력의 `Seconds Behind Master` 필드는 Aurora MySQL DB 클러스터가 MySQL 주 내용보다 얼마나 지연되었는지 보여줍니다. 이 값이 `0` (0)인 경우, Aurora MySQL DB 클러스터는 주 내용을 따라잡습니다. 그러면 다음 단계로 이동해 복제를 중지할 수 있습니다.

8. MySQL 복제 주 데이터베이스를 연결하고, 복제를 중지합니다. 이렇게 하려면 mysql.rds_stop_replication 저장 프로시저를 실행합니다.

```
CALL mysql.rds_stop_replication;
```

Amazon Aurora MySQL로의 물리적 마이그레이션에 걸리는 시간 단축

다음 데이터베이스 수정 사항을 반영하여 Amazon Aurora MySQL로 데이터베이스를 마이그레이션하는 데 필요한 프로세스를 단축할 수 있습니다.

> **Important**
>
> 이러한 업데이트는 프로덕션 데이터베이스가 아닌 프로덕션 데이터베이스의 복제본에서 수행해야 합니다. 그런 다음, 복제본을 백업하고 Aurora DB 클러스터에 복원하여 프로덕션 데이터베이스에서 서비스 중단이 발생하는 것을 방지할 수 있습니다.

지원되지 않는 테이블 유형

Aurora MySQL은 데이터베이스 테이블에서 InnoDB 엔진만 지원합니다. 데이터베이스에 MyISAM 테이블을 보유하고 있으면 Aurora MySQL로 마이그레이션하기 전에 이 테이블을 변환해야 합니다. 이때, 마이그레이션 절차 중 MyISAM에서 InnoDB로 변환하려면 추가 공간이 필요합니다.

공간이 부족할 가능성을 줄이거나 마이그레이션 프로세스 속도를 높이려면 MyISAM 테이블을 마이그레이션하기 전에 모두 InnoDB 테이블로 변환해야 합니다. 이렇게 변환되는 InnoDB 테이블의 크기는 Aurora MySQL에서 해당 테이블에 요구하는 크기와 동일합니다. MyISAM 테이블을 InnoDB로 변환하는 명령은 다음과 같습니다.

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL에서는 압축 테이블 또는 페이지(즉 `ROW_FORMAT=COMPRESSED` 또는 `COMPRESSION = {"zlib"|"lz4"}`로 만든 테이블)이 지원되지 않습니다.

공간이 부족할 가능성을 줄이거나 마이그레이션 프로세스 속도를 높이려면 ROW_FORMAT을 DEFAULT, COMPACT, DYNAMIC 또는 REDUNDANT로 설정하여 압축된 테이블을 확장합니다. 압축된 페이지의 경우 COMPRESSION="none"을 설정합니다.

자세한 내용은 MySQL 설명서의 [InnoDB 행 형식](https://dev.mysql.com/doc/refman/8.0/en/innodb-row-format.html) 및 [InnoDB 테이블 및 페이지 압축](https://dev.mysql.com/doc/refman/8.0/en/innodb-compression.html)을 참조하세요.

기존 MySQL DB 인스턴스에서 다음 SQL 스크립트를 사용하면 데이터베이스에 있는 MyISAM 테이블 또는 압축된 테이블의 목록을 표시할 수 있습니다.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
```

```
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

지원되지 않는 권한을 지닌 사용자 계정

Aurora MySQL에서 지원되지 않는 권한을 지닌 사용자 계정의 경우, 가져올 때 지원되지 않는 권한은 제외됩니다. 지원되는 권한 목록은 역할 기반 권한 모델 단원을 참조하세요.

소스 데이터베이스에서 다음 SQL 쿼리를 실행하여 지원되지 않는 권한이 있는 사용자 계정을 나열할 수 있습니다.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Aurora MySQL 버전 3의 동적 권한

동적 권한은 가져올 수 없습니다. Aurora MySQL 버전 3에서는 다음의 동적 권한을 지원합니다.

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

다음 예제 스크립트는 Aurora MySQL DB 클러스터의 사용자 계정에 지원되는 동적 권한을 부여합니다.

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

'rdsadmin'@'localhost'를 정의자로 사용하여 저장된 객체

`'rdsadmin'@'localhost'`를 정의자로 사용하는 함수, 프로시저, 뷰, 이벤트 및 트리거는 가져올 수 없습니다.

소스 MySQL 데이터베이스에서 다음 SQL 스크립트를 사용하면 지원되지 않는 정의자가 있는 저장된 객체를 나열할 수 있습니다.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.
```

```
SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

mysqldump를 사용하여 MySQL에서 Amazon Aurora MySQL로 논리적 마이그레이션

Amazon Aurora MySQL은 MySQL 호환 데이터베이스이므로 `mysqldump` 유틸리티를 사용하여 MySQL 또는 MariaDB 데이터베이스에서 기존의 Aurora MySQL DB 클러스터로 데이터를 복사할 수 있습니다.

초대형 MySQL 데이터베이스를 사용하는 방법은 [가동 중지 시간을 단축하여 MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기]를 참조하세요. 소량의 데이터를 갖는 MySQL 데이터베이스는 [MySQL 또는 MariaDB DB 데이터를 MySQL 또는 MariaDB DB 인스턴스로 가져오기]를 참조하세요.

RDS MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터를 마이그레이션(복사)할 수 있습니다.

주제

- RDS for MySQL 스냅샷을 Aurora로 마이그레이션
- Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

> **Note**
>
> Amazon Aurora MySQL는 MySQL과 호환되기 때문에 MySQL 데이터베이스와 Amazon Aurora MySQL DB 클러스터 사이에 복제를 설정하면 MySQL 데이터베이스에서 데이터를 마이그레이션할 수 있습니다. 자세한 내용은 Amazon Aurora를 사용한 복제 단원을 참조하십시오.

RDS for MySQL 스냅샷을 Aurora로 마이그레이션

RDS for MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하면 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 새로운 Aurora MySQL DB 클러스터는 원본 RDS for MySQL DB 인스턴스의 데이터로 채워집니다. DB 스냅샷은 Aurora MySQL과 호환되는 MySQL을 실행하는 Amazon RDS DB 인스턴스에서 생성해야 합니다.

DB 스냅샷을 마이그레이션하는 방법은 수동과 자동이 있습니다. DB 클러스터를 생성한 후에는 옵션으로 Aurora 복제본을 생성할 수도 있습니다.

> **Note**
>
> 또한 소스 RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 RDS for MySQL DB 인스턴스를 Aurora MySQL DB 클러스터로 마이그레이션할 수도 있습니다. 자세한 내용은 Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 단원을 참조하십시오.

8.0.11, 8.0.13 및 8.0.15 등의 일부 이전 MySQL 8.0 버전에서는 Aurora MySQL 버전 3.05 이상으로 마이그레이션할 수 없습니다. 마이그레이션하기 전에 MySQL 버전 8.0.28로 업그레이드하는 것이 좋습니다.

일반적으로 다음 단계를 반드시 따라야 합니다.

1. Aurora MySQL DB 클러스터에 프로비저닝할 공간 크기를 결정합니다. 자세한 내용은 필요한 공간 크기 섹션을 참조하세요.
2. 콘솔을 사용하여 Amazon RDS MySQL 인스턴스가 있는 AWS 리전에 스냅샷을 생성합니다. DB 스냅샷 생성에 대한 자세한 내용은 DB 스냅샷 생성을 참조하십시오.
3. DB 스냅샷이 DB 클러스터와 동일한 AWS 리전에 속하지 않을 때는 Amazon RDS 콘솔을 사용하여 DB 스냅샷을 해당 AWS 리전으로 복사합니다. DB 스냅샷 복사에 대한 자세한 내용은 DB 스냅샷 복사를 참조하십시오.
4. 콘솔을 사용하여 DB 스냅샷을 마이그레이션한 후 원본 MySQL DB 인스턴스와 동일한 데이터베이스로 Aurora MySQL DB 클러스터를 생성하십시오.

Warning

Amazon RDS는 AWS 계정마다 스냅샷 복사본을 한 번에 각 AWS 리전의 스냅샷 복사본 하나로 제한합니다.

필요한 공간 크기

MySQL DB 인스턴스의 스냅샷을 Aurora MySQL DB 클러스터로 마이그레이션할 때는 Aurora에서 마이그레이션 전에 Amazon EBS(Amazon Elastic Block Store) 볼륨을 사용하여 스냅샷의 데이터 형식을 결정합니다. 마이그레이션을 위해 데이터 형식을 결정할 때 공간이 추가로 필요한 경우가 간혹 있습니다.

MyISAM 테이블이 아니거나 압축되지 않은 테이블은 최대 크기가 16TB로 제한됩니다. MyISAM 테이블이 있는 경우 Aurora은 Aurora MySQL과 호환되도록 테이블을 변환하기 위해 볼륨에 추가 공간이 필요합니다. 압축된 테이블이 있으면 Aurora은 이 테이블을 Aurora 클러스터 볼륨에 저장하기 전에 확장할 추가 공간이 필요합니다. 이 추가 공간 요구 사항 때문에 MySQL DB 인스턴스에서 마이그레이션하는 MyISAM 테이블과 압축된 테이블이 크기가 8TB를 넘지 않는지 확인해야 합니다.

데이터를 Amazon Aurora MySQL로 마이그레이션하는 데 필요한 공간 크기 줄이기

데이터를 Amazon Aurora로 마이그레이션하기 전에 데이터베이스 스키마 설정을 변경할 수 있습니다. 이렇게 수정하는 이유는 다음과 같은 경우에 도움이 되기 때문입니다.

- 마이그레이션 프로세스를 가속화하고 싶은 경우
- 프로비저닝에 필요한 공간을 정확히 모르는 경우
- 데이터 마이그레이션을 시도했지만 프로비저닝 공간 부족으로 마이그레이션이 실패한 경우

데이터베이스를 Amazon Aurora로 마이그레이션하는 프로세스를 개선하려면 다음과 같은 설정 변경이 가능합니다.

> **Important**
>
> 이 업데이트는 반드시 프로덕션 인스턴스가 아닌 프로덕션 데이터베이스의 스냅샷에서 새롭게 복구된 DB 인스턴스에 실행해야 합니다. 그런 다음, 새로운 DB 인스턴스의 스냅샷에서 Aurora DB 클러스터로 데이터를 마이그레이션하면 프로덕션 데이터베이스의 서비스 중단을 방지할 수 있습니다.

<table>
<tr><th>테이블 유형</th><th>제한 또는 지침</th></tr>
<tr><td>MyISAM 테이블</td><td>Aurora MySQL는 InnoDB 테이블만 지원합니다. 데이터베이스에 MyISAM 테이블이 있으면 Aurora MySQL로 마이그레이션하기 전에 이 테이블을 변환해야 합니다. 이 때, 마이그레이션 절차 중 MyISAM에서 InnoDB로 변환하려면 추가 공간이 필요합니다.

공간이 부족할 가능성을 줄이거나 마이그레이션 프로세스 속도를 높이려면 MyISAM 테이블을 마이그레이션하기 전에 모두 InnoDB 테이블로 변환해야 합니다. 이렇게 변환되는 InnoDB 테이블의 크기는 Aurora MySQL에서 해당 테이블에 요구하는 크기와 동일합니다. MyISAM 테이블을 InnoDB로 변환하는 명령은 다음과 같습니다.

<code>alter table <schema>.<table_name> engine=innodb, algorithm=copy;</code></td></tr>
<tr><td>압축된 테이블</td><td>Aurora MySQL에서는 압축된 테이블(<code>ROW_FORMAT=COMPRESSED</code> 로 만든 테이블)이 지원되지 않습니다.</td></tr>
</table>

테이블 유형	제한 또는 지침
	공간이 부족할 가능성을 줄이거나 마이그레이션 프로세스 속도를 높이려면 ROW_FORMAT 을 DEFAULT, COMPACT, DYNAMIC 또는 REDUNDANT 로 설정하여 압축된 테이블을 확장합니다. 자세한 내용은 MySQL 설명서의 InnoDB 행 형식을 참조하세요.

기존 MySQL DB 인스턴스에서 다음 SQL 스크립트를 사용하면 데이터베이스에 있는 MyISAM 테이블 또는 압축된 테이블의 목록을 표시할 수 있습니다.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
```

```
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

스크립트를 실행하면 다음 예제와 비슷한 결과를 출력합니다. 이 예제는 MyISAM에서 InnoDB로 변환해야 하는 두 개의 테이블을 보여 줍니다. 그 밖에도 출력 화면에는 각 테이블의 크기도 메가바이트(MB) 단위로 나옵니다.

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

RDS for MySQL DB 스냅샷을 Aurora MySQL DB 클러스터로 마이그레이션

AWS Management Console 또는 AWS CLI를 통해 RDS for MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하여 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 새로운 Aurora MySQL DB 클러스터는 원본 RDS for MySQL DB 인스턴스의 데이터로 채워집니다. DB 스냅샷 생성에 대한 자세한 내용은 DB 스냅샷 생성을 참조하세요.

DB 스냅샷이 데이터를 마이그레이션하려는 AWS 리전에 없는 경우에는 DB 스냅샷을 해당 AWS 리전으로 복사합니다. DB 스냅샷 복사에 대한 자세한 내용은 DB 스냅샷 복사를 참조하세요.

콘솔

AWS Management Console을 사용하여 DB 스냅샷을 마이그레이션할 경우, 콘솔에서 DB 클러스터만 생성하는 데 필요한 작업이 따릅니다.

AWS KMS key을(를) 사용하여 새 Aurora MySQL DB 클러스터가 유휴 상태에서 암호화되도록 선택할 수도 있습니다.

AWS Management Console을 사용하여 MySQL DB 스냅샷을 마이그레이션하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. MySQL DB 인스턴스 또는 스냅샷에서 마이그레이션을 시작하십시오.

   DB 인스턴스에서 마이그레이션을 시작하려면

   1. 탐색 창에서 데이터베이스를 선택한 다음 MySQL DB 인스턴스를 선택합니다.
   2. 작업에서 최근 스냅샷 마이그레이션을 선택합니다.

   스냅샷에서 마이그레이션을 시작하려면

   1. [Snapshots]를 선택합니다.
   2. [Snapshots] 페이지에서 Aurora MySQL DB 클러스터로 마이그레이션하려는 스냅샷을 선택합니다.
   3. [Snapshot Actions]를 선택한 다음 [Migrate Snapshot]을 선택합니다.

   [Migrate Database] 페이지가 표시됩니다.
3. [Migrate Database] 페이지에서 다음과 같이 값을 설정합니다.
   - Migrate to DB Engine: `aurora`를 선택합니다.
   - DB 엔진 버전: Aurora MySQL DB 클러스터에 대해 DB 엔진 버전을 선택합니다.
   - DB 인스턴스 클래스: 데이터베이스에 필요한 스토리지와 용량을 가진 DB 인스턴스 클래스를 선택합니다(예: `db.r3.large`). 데이터베이스의 데이터 용량이 늘어날수록 Aurora 클러스터 볼륨도 자동 확장됩니다. Aurora 클러스터 볼륨 크기는 최대 128 tebibytes (TiB)까지 증가할 수

있습니다. 따라서 현재 스토리지 요구 사항에 맞는 DB 인스턴스 클래스를 선택해야 합니다. 자세한 내용은 Amazon Aurora 스토리지 개요 섹션을 참조하세요.

- [DB 인스턴스 식별자(DB Instance Identifier)]: 선택한 AWS 리전의 계정에 대해 고유한 DB 클러스터 이름을 입력합니다. 이 식별자는 DB 클러스터에 속한 인스턴스의 엔드포인트 주소로 사용됩니다. 선택한 AWS 리전 및 DB 엔진을 포함(예: **aurora-cluster1**)하는 등 이름에 지능적 요소를 추가할 수도 있습니다.

  DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다.

  - 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
  - 각 AWS 리전별로 AWS 계정마다 모든 DB 인스턴스가 고유해야 합니다.
- Virtual Private Cloud(VPC): 기존 VPC가 있는 경우, VPC 식별자(예: `vpc-a464d1c1`)를 선택하여 해당 VPC를 Aurora MySQL DB 클러스터에 사용할 수 있습니다. VPC 생성에 대한 자세한 내용은 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 단원을 참조하세요.

  기존 VPC가 없다면 새 VPC 생성을 선택하여 Aurora에서 VPC를 새로 생성하도록 할 수 있습니다.
- 서브넷 그룹: 기존 서브넷 그룹이 있으면 해당 서브넷 그룹 식별자(예: `gs-subnet-group1`)를 선택하여 Aurora MySQL DB 클러스터에 그 서브넷 그룹을 사용할 수 있습니다.

  기존 서브넷 그룹이 없다면 새 서브넷 그룹 생성을 선택하여 Aurora에서 서브넷 그룹을 새로 생성하도록 할 수 있습니다.
- [Public accessibility]: VPC에 있는 리소스만 DB 클러스터의 인스턴스에 액세스할 수 있도록 하려면 [No]를 선택합니다. 퍼블릭 네트워크에 있는 리소스가 DB 클러스터의 인스턴스에 액세스할 수 있도록 하려면 [Yes]를 선택합니다. 기본값은 [Yes]입니다.

  > **Note**
  >
  > 퍼블릭 서브넷에서는 프로덕션 DB 클러스터가 필요 없을 수도 있습니다. 애플리케이션 서버만 DB 클러스터에 액세스하기 때문입니다. DB 클러스터가 퍼블릭 서브넷에 필요 없는 경우에는 [Publicly Accessible]을 [No]로 설정합니다.
- 가용 영역: Aurora MySQL DB 클러스터의 기본 인스턴스를 호스팅하려면 가용 영역을 선택합니다. Aurora에서 가용 영역을 선택하게 하려면 기본 설정 없음을 선택합니다.

- 데이터베이스 포트: Aurora MySQL DB 클러스터의 인스턴스에 연결할 때 사용할 기본 포트를 입력합니다. 기본값은 3306입니다.

> **Note**
>
> 기업 방화벽 뒤에 있어서 MySQL 기본 포트인 3306 같은 기본 포트에 액세스하지 못할 수도 있습니다. 이런 경우에는 기업 방화벽이 허용하는 포트 값을 입력합니다. 나중에 Aurora MySQL DB 클러스터에 연결할 때도 필요하므로 이 포트 값을 기억해야 합니다.

- 암호화: 새 Aurora MySQL DB 클러스터를 저장 상태에서 암호화하려면 암호화 활성화를 선택합니다. 암호화 사용을 선택한 경우 KMS 키를 AWS KMS key 값으로 선택해야 합니다.

  DB 스냅샷이 암호화되어 있지 않으면 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다.

  DB 스냅샷이 암호화되어 있으면 지정된 암호화 키를 사용하여 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. DB 스냅샷 또는 다른 키에서 사용하는 암호화 키를 지정할 수 있습니다. 암호화된 DB 스냅샷에서 암호화되지 않은 DB 클러스터를 생성할 수 없습니다.
- 마이너 버전 자동 업그레이드: 이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다.

  Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

4. [Migrate]를 선택하여 DB 스냅샷을 마이그레이션합니다.
5. [Instances]를 선택한 다음 화살표 아이콘을 선택하여 DB 클러스터 세부 정보를 표시하고 마이그레이션 진행 상황을 모니터링합니다. 세부 정보 페이지를 보면 DB 클러스터의 기본 인스턴스에 연결하는 데 사용할 클러스터 엔드포인트가 표시됩니다. Aurora MySQL DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

AWS CLI

`restore-db-cluster-from-snapshot` 명령을 다음 파라미터와 함께 사용하여 RDS for MySQL DB 인스턴스의 DB 스냅샷으로 Aurora DB 클러스터를 생성할 수 있습니다.

- `--db-cluster-identifier` - 생성할 DB 클러스터의 이름입니다.
- `--engine aurora-mysql` - MySQL 5.7 호환 또는 8.0 호환 DB 클러스터의 경우.
- `--kms-key-id` - DB 스냅샷 암호화 여부에 따라 선택적으로 DB 클러스터를 암호화하는 데 사용할 수 있는 AWS KMS key.

- DB 스냅샷이 암호화되어 있지 않으면 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 DB 클러스터가 암호화되지 않습니다.
- DB 스냅샷이 암호화되어 있으면 지정된 암호화 키를 사용하여 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 DB 스냅샷용 암호화 키를 사용할 때 DB 클러스터가 유휴 상태에서 암호화됩니다.

> **Note**
>
> 암호화된 DB 스냅샷에서 암호화되지 않은 DB 클러스터를 생성할 수 없습니다.

- `--snapshot-identifier` - 마이그레이션할 DB 스냅샷의 Amazon 리소스 이름(ARN)입니다. Amazon RDS ARN에 대한 자세한 내용은 [Amazon Relational Database Service (Amazon RDS)](#) 단원을 참조하십시오.

[`RestoreDBClusterFromSnapshot`] 명령으로 DB 스냅샷을 마이그레이션하면 이 명령으로 DB 클러스터와 기본 인스턴스가 모두 생성됩니다.

이 예제에서는 ARN이 *`mydbsnapshotARN`*으로 설정된 DB 스냅샷에서 이름이 *`mydbcluster`*인 MySQL 5.7–호환 DB 클러스터를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

이 예제에서는 ARN이 *`mydbsnapshotARN`*으로 설정된 DB 스냅샷에서 이름이 *`mydbcluster`*인 MySQL 5.7–호환 DB 클러스터를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

Aurora는 RDS for MySQL DB 엔진의 바이너리 로그 복제 기능을 사용하여 소스 MySQL DB 인스턴스의 Aurora 읽기 전용 복제본이라고 하는 특수한 유형의 DB 클러스터를 만들 수 있습니다. 소스 RDS for MySQL DB 인스턴스에 적용된 업데이트는 Aurora 읽기 전용 복제본에 비동기 방식으로 복제됩니다.

이 기능을 사용하여 소스 RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성해 RDS for MySQL DB 인스턴스에서 Aurora MySQL DB 클러스터로 마이그레이션하는 것을 권장합니다. RDS for MySQL DB 인스턴스와 Aurora 읽기 전용 복제본 간 복제 지연이 0이라면 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본으로 이동한 다음, 복제를 중단하여 Aurora 읽기 전용 복제본을 독립형 Aurora MySQL DB 클러스터로 생성할 수 있습니다. 마이그레이션에는 상당한 시간이 드는데, 데이터 테비바이트(TiB)당 몇 시간 정도가 소요됩니다.

Aurora를 사용할 수 있는 리전 목록은 AWS 일반 참조 일반 참조에서 [Amazon Aurora](#) 섹션을 참조하세요.

RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하면 Amazon RDS는 소스 RDS for MySQL DB 인스턴스의 DB 스냅샷을 생성합니다(Amazon RDS에 프라이빗 형식이며, 요금이 발생하지 않음). 이어서 Amazon RDS는 DB 스냅샷에서 Aurora 읽기 전용 복제본으로 데이터를 마이그레이션합니다. 데이터가 DB 스냅샷에서 새로운 Aurora MySQL DB 클러스터로 마이그레이션된 후 Amazon RDS는 RDS for MySQL DB 인스턴스와 Aurora MySQL DB 클러스터 간 복제를 시작합니다. RDS for MySQL DB 인스턴스에 InnoDB 외의 스토리지 엔진을 사용하거나 압축된 행 형식을 사용하는 테이블이 포함된 경우, Aurora 읽기 전용 복제본을 만들기 전에 이 테이블들이 InnoDB 스토리지 엔진과 동적 행 형식을 사용하도록 변경함으로써 Aurora 읽기 전용 복제본 생성 과정의 속도를 높일 수 있습니다. MySQL DB 스냅샷을 Aurora MySQL DB 클러스터에 복사하는 과정에 관한 자세한 내용은 [RDS MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션](#) 단원을 참조하십시오.

RDS for MySQL DB 인스턴스 하나에 Aurora 읽기 전용 복제본 하나만 둘 수 있습니다.

> **Note**
>
> Aurora MySQL과 복제 주요 항목인 RDS for MySQL DB 인스턴스의 MySQL 데이터베이스 엔진 버전의 기능 차이로 인해 복제 문제가 발생할 수 있습니다. 오류가 발생하면 [Amazon RDS 커뮤니티 포럼](#) 또는 AWS Support에 문의하여 지원을 받을 수 있습니다.
> RDS for MySQL DB 인스턴스가 리전 간 읽기 전용 복제본의 소스인 경우 Aurora 읽기 전용 복제본을 생성할 수 없습니다.

> 8.0.11, 8.0.13 및 8.0.15 등의 일부 이전 RDS for MySQL 8.0 버전에서는 Aurora MySQL 버전 3.05 이상으로 마이그레이션할 수 없습니다. 마이그레이션하기 전에 RDS for MySQL 버전 8.0.28로 업그레이드하는 것이 좋습니다.

MySQL 읽기 전용 복제본에 대한 자세한 내용은 MariaDB, MySQL 및 PostgreSQL DB 인스턴스의 읽기 전용 복제본 작업을 참조하십시오.

Aurora 읽기 전용 복제본 생성

RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본은 콘솔, AWS CLI 또는 RDS API를 사용하여 생성할 수 있습니다.

콘솔

소스 RDS for MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Aurora 읽기 전용 복제본의 원본으로 사용할 MySQL DB 인스턴스를 선택합니다.
4. 작업에서 Aurora 읽기 전용 복제본 만들기를 선택합니다.
5. 다음 표의 설명대로 Aurora 읽기 전용 복제본에 사용하려는 DB 클러스터 사양을 선택합니다.

옵션	설명
DB 인스턴스 클래스	DB 클러스터의 기본 인스턴스에 대한 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.
다중 AZ 배포	장애 조치 지원을 위해 대상 AWS 리전의 다른 가용 영역에 새 DB 클러스터의 예비 복제본을 만들려면 [다른 영역에 복제본 생성(Create Replica in Different Zone)]을 선택합니다. 다중 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 단원을 참조하십시오.

옵션	설명
DB 인스턴스 식별자	Aurora 읽기 전용 복제본 DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 새 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다. DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다. • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • 개별 AWS 리전별로 각 AWS 계정의 모든 DB 인스턴스에 대해 고유해야 합니다. Aurora 읽기 전용 복제본 DB 클러스터는 원본 DB 인스턴스의 스냅샷으로 만드는 것이기 때문에 Aurora 읽기 전용 복제본의 마스터 사용자 이름과 마스터 암호는 원본 DB 인스턴스의 마스터 사용자 이름 및 마스터 암호와 동일합니다.
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Aurora가 자동으로 VPC를 생성하도록 하려면 Create new VPC(새 VPC 생성)를 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 단원을 참조하십시오.
DB 서브넷 그룹	DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다. Aurora가 자동으로 DB 서브넷 그룹을 생성하도록 하려면 Create new DB subnet group(새 DB 서브넷 그룹 생성)을 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.

옵션	설명
퍼블릭 액세스 가능성	DB 클러스터에 퍼블릭 IP 주소를 할당하려면 Yes를 선택하고, 그렇지 않으면 No를 선택합니다. DB 클러스터의 인스턴스는 퍼블릭과 프라이빗 DB 인스턴스를 모두 혼합하여 사용할 수 있습니다. 모든 사용자의 액세스에서 인스턴스를 숨기는 방법에 대한 자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 단원을 참조하십시오.
가용 영역	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 단원을 참조하세요.
VPC 보안 그룹(방화벽)	Aurora가 자동으로 VPC 보안 그룹을 생성하도록 하려면 새 VPC 보안 그룹 생성을 선택합니다. [Select existing VPC security groups]를 선택하고 하나 이상의 VPC 보안 그룹을 지정하여 DB 클러스터에 대한 네트워크 액세스에 보안을 적용합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.
데이터베이스 포트	애플리케이션과 유틸리티가 데이터베이스에 액세스할 때 사용할 포트를 지정합니다. Aurora MySQL DB 클러스터는 기본 MySQL 포트인 3306으로 지정됩니다. 일부 기업에서는 방화벽으로 이 포트에 대한 연결을 차단합니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 새로운 DB 클러스터에 다른 포트를 선택해야 합니다.
DB 파라미터 그룹	Aurora MySQL DB 클러스터에 대해 DB 파라미터 그룹을 선택합니다. Aurora에 사용할 수 있는 기본 파라미터 그룹이 포함되어 있거나, 직접 파라미터 그룹을 생성할 수 있습니다. DB 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.

옵션	설명
DB 클러스터 파라미터 그룹	Aurora MySQL DB 클러스터에 대한 DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본값으로 사용할 수 있는 DB 클러스터 파라미터 그룹을 제공하며, 자체 DB 클러스터 파라미터 그룹을 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.
암호화	새 Aurora DB 클러스터를 암호화지 않으려면 [Disable encryption]을 선택합니다. 새 Aurora DB 클러스터를 유휴 상태에서 암호화하려면 암호화 활성화를 선택합니다. 암호화 사용을 선택한 경우 KMS 키를 AWS KMS key 값으로 선택해야 합니다. MySQL DB 인스턴스가 암호화되어 있지 않으면 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. MySQL DB 인스턴스가 암호화되어 있으면 지정된 암호화 키를 사용하여 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. MySQL DB 인스턴스 또는 다른 키에서 사용하는 암호화 키를 지정할 수 있습니다. 암호화된 MySQL DB 인스턴스에서 암호화되지 않은 DB 클러스터를 생성할 수 없습니다.
우선순위	DB 클러스터의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스의 결함으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.
백업 보관 기간	Aurora가 데이터베이스 백업 사본을 보관하는 기간을 1~35일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.

옵션	설명
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.
역할 모니터링	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. Aurora가 Amazon CloudWatch Logs와 통신할 수 있도록 생성한 IAM 역할을 선택하거나 Aurora가 `rds-monitoring-role` 이라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.
세부 수준	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
마이너 버전 자동 업그레이드	이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.
유지보수 윈도우	[Select window]를 선택하고 시스템 유지 관리를 실행할 수 있는 주 단위 기간을 지정합니다. 또는 Aurora가 임의로 기간을 지정하도록 하려면 기본 설정 없음을 선택합니다.

6. [Create read replica]를 선택합니다.

AWS CLI

소스 RDS for MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 `create-db-cluster` 및 `create-db-instance` AWS CLI 명령을 사용하여 Aurora MySQL DB 클러스터를 새로 생성합니다. `create-db-cluster` 명령을 호출할 때는 원본 MySQL DB 인스턴스의 Amazon 리소스 이름(ARN)을 식별하는 `--replication-source-identifier` 파라미터를 포함시키십시오.

Amazon RDS ARN에 대한 자세한 내용은 Amazon Relational Database Service (Amazon RDS) 단원을 참조하십시오.

Aurora 읽기 전용 복제본은 원본 MySQL DB 인스턴스와 동일한 마스터 사용자 이름, 마스터 암호, 데이터베이스 이름을 사용하므로 마스터 사용자 이름, 마스터 암호 또는 데이터베이스 이름을 지정하지 마십시오.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

콘솔을 사용하여 Aurora 읽기 전용 복제본을 생성하면 Aurora에서 DB 클러스터 Aurora 읽기 전용 복제본의 기본 인스턴스를 자동으로 생성합니다. AWS CLI를 사용하여 Aurora 읽기 전용 복제본을 생성할 경우 반드시 DB 클러스터용 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.

`create-db-instance` AWS CLI 명령을 다음 파라미터와 함께 사용하여 DB 클러스터의 기본 인스턴스를 생성할 수 있습니다.

- `--db-cluster-identifier`

  DB 클러스터의 이름입니다.
- `--db-instance-class`

  기본 인스턴스에 사용할 DB 인스턴스 클래스의 이름입니다.
- `--db-instance-identifier`

  기본 인스턴스의 이름입니다.

- `--engine aurora`

이 예에서는 *myinstanceclass*에 지정된 DB 인스턴스 클래스를 사용하여 이름이 *myreadreplicacluster*인 DB 클러스터에 대해 이름이 *myreadreplicainstance*인 기본 인스턴스를 생성합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

Windows의 경우:

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS API

소스 RDS for MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 CreateDBCluster 및 CreateDBInstance Amazon RDS API 명령을 사용하여 새 Aurora DB 클러스터와 기본 인스턴스를 생성합니다. Aurora 읽기 전용 복제본은 소스 RDS for MySQL DB 인스턴스와 동일한 마스터 사용자 이름, 마스터 암호, 데이터베이스 이름을 사용하므로 마스터 사용자 이름, 마스터 암호 또는 데이터베이스 이름을 지정하지 않도록 합니다.

CreateDBCluster Amazon RDS API 명령을 다음 파라미터와 함께 사용하여 소스 RDS for MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본의 새 Aurora DB 클러스터를 생성할 수 있습니다.

- `DBClusterIdentifier`

  생성할 DB 클러스터의 이름입니다.
- `DBSubnetGroupName`

  이 DB 클러스터와 연결할 DB 서브넷 그룹의 이름입니다.

- `Engine=aurora`
- `KmsKeyId`

  MySQL DB 인스턴스 암호화 여부에 따라 DB 클러스터를 암호화할 수 있는 AWS KMS key입니다.

  - MySQL DB 인스턴스가 암호화되어 있지 않으면 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 계정용 기본 암호화 키를 사용할 때 DB 클러스터가 유휴 상태에서 암호화됩니다.
  - MySQL DB 인스턴스가 암호화되어 있으면 지정된 암호화 키를 사용하여 유휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 MySQL DB 인스턴스용 암호화 키를 사용할 때 DB 클러스터가 유휴 상태에서 암호화됩니다.

  > **Note**
  >
  > 암호화된 MySQL DB 인스턴스에서 암호화되지 않은 DB 클러스터를 생성할 수 없습니다.

- `ReplicationSourceIdentifier`

  원본 MySQL DB 인스턴스의 Amazon 리소스 이름(ARN)입니다. Amazon RDS ARN에 대한 자세한 내용은 [Amazon Relational Database Service (Amazon RDS)](https://docs.aws.amazon.com/) 단원을 참조하십시오.
- `VpcSecurityGroupIds`

  이 DB 클러스터와 연결할 EC2 VPC 보안 그룹 목록입니다.

이 예에서는 ARN이 *`mysqlmasterARN`*으로 설정되어 있으며, 이름이 *`mysubnetgroup`*인 DB 서브넷 그룹과 이름이 *`mysecuritygroup`*인 VPC 보안 그룹과 연결되어 있는 원본 MySQL DB 인스턴스에서 이름이 *`myreadreplicacluster`*인 DB클러스터를 생성합니다.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
```

```
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

콘솔을 사용하여 Aurora 읽기 전용 복제본을 생성하면 Aurora에서 DB 클러스터 Aurora 읽기 전용 복제본의 기본 인스턴스를 자동으로 생성합니다. AWS CLI를 사용하여 Aurora 읽기 전용 복제본을 생성할 경우 반드시 DB 클러스터용 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.

`CreateDBInstance` Amazon RDS API 명령을 다음 파라미터와 함께 사용하여 DB 클러스터의 기본 인스턴스를 생성할 수 있습니다.

- `DBClusterIdentifier`

  DB 클러스터의 이름입니다.
- `DBInstanceClass`

  기본 인스턴스에 사용할 DB 인스턴스 클래스의 이름입니다.
- `DBInstanceIdentifier`

  기본 인스턴스의 이름입니다.
- `Engine=aurora`

이 예에서는 *myinstanceclass*에 지정된 DB 인스턴스 클래스를 사용하여 이름이 *myreadreplicacluster*인 DB 클러스터에 대해 이름이 *myreadreplicainstance*인 기본 인스턴스를 생성합니다.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
```

```
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
&X-Amz-Date=20140424T194844Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

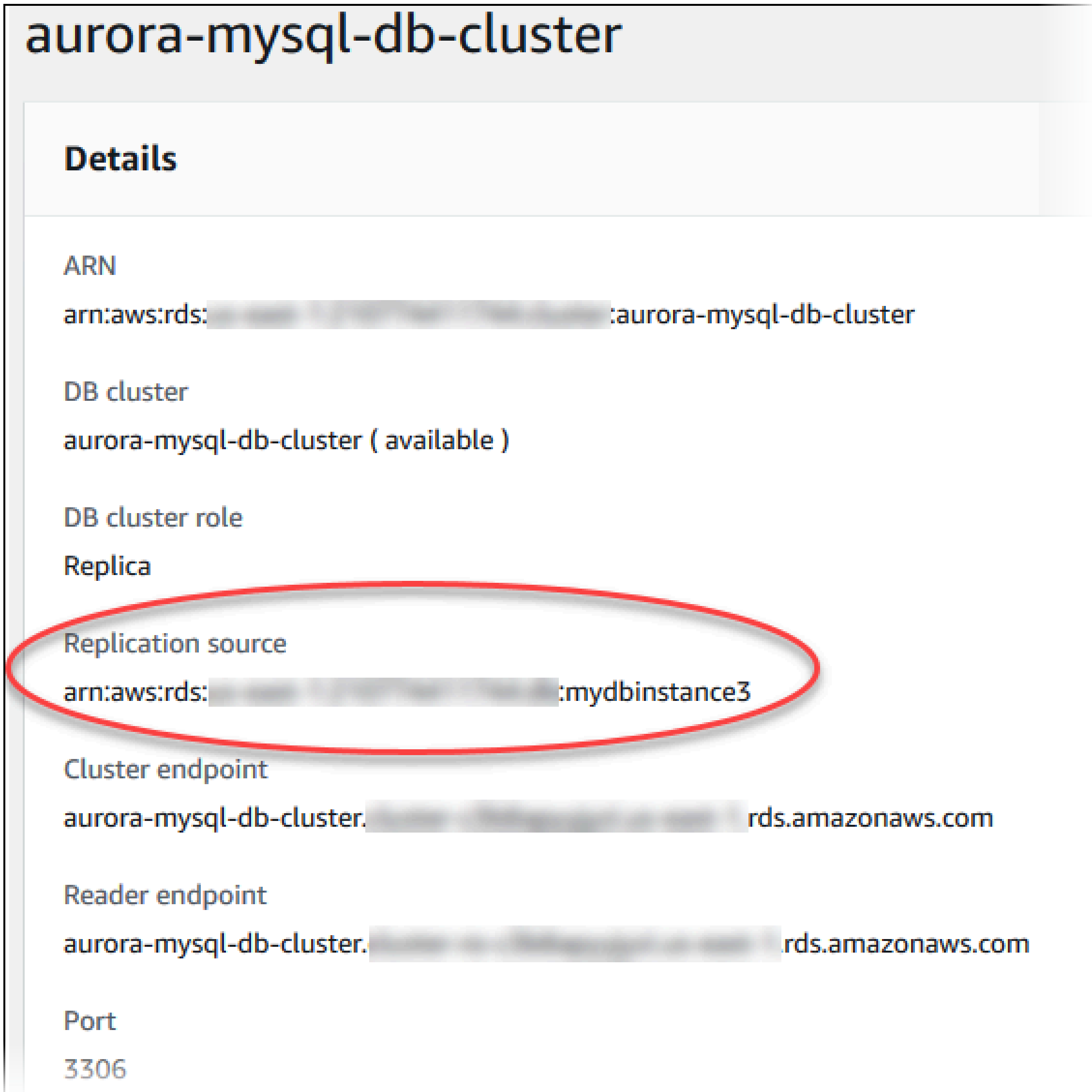
Aurora 읽기 전용 복제본 보기

AWS Management Console 또는 AWS CLI를 사용하여 Aurora MySQL DB 클러스터의 MySQL에서 Aurora MySQL로의 복제 관계를 볼 수 있습니다.

콘솔

Aurora 읽기 전용 복제본의 주 MySQL DB 인스턴스를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Aurora 읽기 전용 복제본의 DB 클러스터를 클릭하여 세부 정보를 표시하십시오. 주 MySQL DB 인스턴스 정보가 Replication source 필드에 나타납니다.



AWS CLI

AWS CLI를 사용하여 Aurora MySQL DB 클러스터의 MySQL에서 Aurora MySQL로의 복제 관계를 보려면 `describe-db-clusters` 및 `describe-db-instances` 명령을 사용합니다.

어떤 MySQL DB 인스턴스가 마스터인지 결정하려면 `describe-db-clusters`를 사용하고 `--db-cluster-identifier` 옵션에서 Aurora 읽기 전용 복제본의 클러스터 식별자를 지정하십시오. 복제 주요 항목인 DB 인스턴스의 ARN 출력에서 `ReplicationSourceIdentifier` 요소를 참조하십시오.

어느 DB 클러스터가 Aurora 읽기 전용 복제본인지 확인하려면 `describe-db-instances`를 사용하여 `--db-instance-identifier` 옵션용 MySQL DB 인스턴스의 인스턴스

식별자를 지정하십시오. Aurora 읽기 전용 복제본의 DB 클러스터 식별자의 출력에서 `ReadReplicaDBClusterIdentifiers` 요소를 참조하십시오.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

Windows의 경우:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

Aurora 읽기 전용 복제본 승격

마이그레이션이 완료된 후 AWS Management Console 또는 AWS CLI를 사용하여 Aurora 읽기 전용 복제본을 독립 실행형 DB 클러스터로 승격할 수 있습니다.

그런 다음 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본의 엔드포인트로 보낼 수 있습니다. Aurora 엔드포인트에 대한 자세한 내용은 [Amazon Aurora 엔드포인트 연결](#) 단원을 참조하세요. 승격은 상당히 빨리 완료되며, 승격 중에 Aurora 읽기 전용 복제본에 대한 읽기 및 쓰기가 가능합니다. 다만 승격 도중에 주 MySQL DB 인스턴스를 삭제하거나 DB 인스턴스와 Aurora 읽기 전용 복제본의 링크를 해제할 수는 없습니다.

Aurora 읽기 전용 복제본을 승격시키기 전에 원본 MySQL DB 인스턴스에 대한 트랜잭션 쓰기를 중단한 다음 Aurora 읽기 전용 복제본에서의 복제 지연이 0이 될 때까지 기다리십시오. Aurora 읽기 전용 복제본에서 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 명령을 호출하여 Aurora 읽기 전용 복제본의 복제 지연을 확인할 수 있습니다. 마스터보다 수 초 지연 값을 확인합니다.

주요 내용에 대한 쓰기 트랜잭션이 멈추고 복제 지연이 0이 된 후 Aurora 읽기 전용 복제본에 대한 쓰기를 시작할 수 있습니다. 그 전에 Aurora 읽기 전용 복제본에 쓰기를 수행하고 테이블을 수정하면

(MySQL에서도 수정됨) Aurora에 대한 복제가 실패할 수 있습니다. 이렇게 될 경우, Aurora 읽기 전용 복제본을 삭제하고 다시 만들어야 합니다.

콘솔

Aurora 읽기 전용 복제본을 Aurora DB 클러스터로 승격시키려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Aurora 읽기 전용 복제본에 사용할 DB 클러스터를 선택합니다.
4. 작업에서 Promote(승격)를 선택합니다.
5. Promote Read Replica를 선택합니다.

프로모션을 진행한 후 다음 절차에 따라 승격이 완료되었는지 확인합니다.

Aurora 읽기 전용 복제본이 승격되었는지 확인하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Events]를 선택합니다.
3. 이벤트 페이지에서 승격한 클러스터에 대한 `Promoted Read Replica cluster to a stand-alone database cluster` 이벤트가 있는지 확인합니다.

승격이 완료된 후에는 주 MySQL DB 인스턴스와 Aurora 읽기 전용 복제본의 링크가 해제되고, 원한다면 DB 인스턴스를 안전하게 삭제할 수 있습니다.

AWS CLI

Aurora 읽기 전용 복제본을 독립형 DB 클러스터로 승격하려면 [`promote-read-replica-db-cluster`]() AWS CLI 명령을 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Windows의 경우:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

Amazon Aurora MySQL 관리

다음 섹션에서는 Amazon Aurora MySQL DB 클러스터 관리에 대해서 살펴보겠습니다.

주제

Amazon Aurora MySQL에 대한 성능 및 조정 관리

Aurora MySQL DB 인스턴스 조정

Aurora MySQL DB 인스턴스는 인스턴스 조정과 읽기 조정, 이렇게 두 가지 방식으로 조정할 수 있습니다. 읽기 조정에 대한 자세한 내용은 읽기 확장 단원을 참조하십시오.

DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora MySQL DB 클러스터의 크기를 조정할 수 있습니다. Aurora MySQL은 Aurora에 최적화된 여러 DB 인스턴스 클래스를 지원합니다. 크기가 40TB보다 큰 Aurora 클러스터에는 db.t2 또는 db.t3 인스턴스 클래스를 사용하지 마십시오. Aurora MySQL에서 지원하는 DB 인스턴스 클래스의 사양 정보는 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 개발 및 테스트에 T 인스턴스 클래스 사용 섹션을 참조하세요.

Aurora MySQL DB 인스턴스에 대한 최대 연결

Aurora MySQL DB 인스턴스에 대해 허용되는 최대 연결 수는 DB 인스턴스의 인스턴스 수준 파라미터 그룹의 `max_connections` 파라미터로 결정됩니다.

다음 표에는 Aurora MySQL에서 사용 가능한 각 DB 인스턴스 클래스에 대한 `max_connections`의 결과 기본값이 나와 있습니다. 인스턴스를 메모리가 더 많은 DB 인스턴스까지 확장하거나, 인스턴스의

DB 파라미터 그룹에 있는 `max_connections` 파라미터의 값을 최대 16,000까지 큰 값으로 설정하여 Aurora MySQL DB 인스턴스의 최대 연결 수를 늘릴 수 있습니다.

> **Tip**
>
> 애플리케이션이 연결을 자주 열고 닫거나, 수명이 긴 연결을 많이 열어 두는 경우, Amazon RDS 프록시를 사용하는 것이 좋습니다. RDS 프록시는 데이터베이스 연결을 효율적으로 안전하게 공유하기 위해 연결 풀링을 사용하는 완전관리형 고가용성 데이터베이스 프록시입니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

Aurora Serverless v2 인스턴스가 이 파라미터를 처리하는 방법에 대한 자세한 내용은 Aurora Serverless v2의 최대 연결 수 섹션을 참조하세요.

인스턴스 클래스	max_connections 기본값		
db.t2.small	45		
db.t2.medium	90		
db.t3.small	45		
db.t3.medium	90		
db.t3.large	135		
db.t4g.medium	90		
db.t4g.large	135		
db.r3.large	1000		
db.r3.xlarge	2000		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		

인스턴스 클래스	max_connections 기본값		
db.r3.8xlarge	5000		
db.r4.large	1000		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		

인스턴스 클래스	max_connections 기본값		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		
db.r6g.16xlarge	6000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6000		
db.r6i.16xlarge	6000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.r7g.large	1000		
db.r7g.xlarge	2000		
db.r7g.2xlarge	3000		
db.r7g.4xlarge	4000		
db.r7g.8xlarge	5000		
db.r7g.12xlarge	6000		
db.r7g.16xlarge	6000		

인스턴스 클래스	max_connections 기본값
db.r7i.large	1000
db.r7i.xlarge	2000
db.r7i.2xlarge	3000
db.r7i.4xlarge	4000
db.r7i.8xlarge	5000
db.r7i.12xlarge	6000
db.r7i.16xlarge	6000
db.r7i.24xlarge	7000
db.r7i.48xlarge	8000
db.r8g.large	1000
db.r8g.xlarge	2000
db.r8g.2xlarge	3000
db.r8g.4xlarge	4000
db.r8g.8xlarge	5000
db.r8g.12xlarge	6000
db.r8g.16xlarge	6000
db.r8g.24xlarge	7000
db.r8g.48xlarge	8000
db.x2g.large	2000

인스턴스 클래스	max_connections 기본값		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

> **Tip**
>
> `max_connections` 파라미터 계산에서는 밑이 2인 로그(자연 로그와 다름)와 선택한 Aurora MySQL 인스턴스 클래스의 바이트 단위 `DBInstanceClassMemory` 값을 사용합니다. 파라미터는 정수 값만 허용하며 계산에서 소수 부분이 잘립니다. 공식은 다음과 같이 연결 한도를 구현합니다.
>
> - 더 큰 R3, R4 및 R5 인스턴스의 경우 1,000 연결 증분
> - T2 및 T3 인스턴스 메모리 변형의 경우 45 연결 증분
>
> 예: db.r6g.large의 경우 공식이 1069.2를 계산하는 동안 시스템은 일관된 증분 패턴을 유지하기 위해 1,000을 구현합니다.

새 파라미터 그룹을 생성하여 연결 제한에 대한 자체 기본값을 사용자 지정하는 경우 이 기본값 연결 제한은 `DBInstanceClassMemory` 값을 기반으로 한 공식을 사용하여 파생된다는 것을 알 수 있습니다. 앞의 표에서 볼 수 있듯이 이 공식은 점점 커지는 R3, R4 및 R5 인스턴스 간에 메모리가 두 배로 늘어남에 따라 1000씩 증가하는 연결 제한을 생성하고 T2 및 T3 인스턴스의 서로 다른 메모리 크기에 대해 45씩 증가하는 연결 제한을 생성합니다.

`DBInstanceClassMemory` 계산 방법에 대한 자세한 내용은 [DB 파라미터 지정](#) 단원을 참조하세요.

Aurora MySQL과 RDS for MySQL DB 인스턴스의 메모리 오버헤드는 서로 다릅니다. 따라서 동일한 인스턴스 클래스를 사용하는 Aurora MySQL과 RDS for MySQL DB 인스턴스의 `max_connections` 값은 다를 수 있습니다. 테이블의 값은 Aurora MySQL DB 인스턴스에만 적용됩니다.

> **Note**
>
> Aurora를 사용할 경우 T2 및 T3 인스턴스 클래스는 프로덕션 워크로드가 아니라 개발 및 테스트 시나리오에만 사용되기 때문에 해당 인스턴스의 연결 한계는 훨씬 낮습니다.

기본값 연결 한계는 버퍼 풀 및 쿼리 캐시와 같은 다른 주요 메모리 소비자에 대한 기본값을 사용하는 시스템에 맞게 조정됩니다. 클러스터의 다른 설정을 변경하는 경우 DB 인스턴스의 사용 가능한 메모리 증가 또는 감소를 고려하여 연결 한계를 조정할 것을 검토하십시오.

Aurora MySQL에 대한 임시 스토리지 한도

Aurora MySQL은 Aurora 스토리지 하위 시스템에 테이블 및 인덱스를 저장합니다. Aurora MySQL은 비영구 임시 파일 및 InnoDB가 아닌 임시 테이블에 대해 별도의 임시 스토리지 또는 로컬 스토리지를 사용합니다. 로컬 스토리지는 쿼리 처리 중 대용량 데이터 세트를 정렬하거나 인덱스 빌드 작업을 위해 사용하는 파일도 포함합니다. InnoDB 임시 테이블은 포함되지 않습니다.

Aurora MySQL 버전 3의 임시 테이블에 대한 자세한 내용은 [Aurora MySQL 버전 3의 새로운 임시 테이블 동작](#) 단원을 참조하세요. 버전 2의 임시 테이블에 대한 자세한 내용은 [Aurora MySQL 버전 2의 임시 테이블스페이스 동작](#) 단원을 참조하세요.

DB 인스턴스를 시작 및 중지할 때와 호스트를 교체하는 동안 이러한 볼륨의 데이터 및 임시 파일이 손실됩니다.

이러한 로컬 스토리지 볼륨은 Amazon Elastic Block Store(EBS)에서 백업하며, 더 큰 DB 인스턴스 클래스를 사용하여 확장할 수 있습니다. 스토리지에 대한 자세한 내용은 [Amazon Aurora 스토리지](#) 단원을 참조하세요.

로컬 스토리지는 `LOAD DATA FROM S3` 또는 `LOAD XML FROM S3`를 사용하여 Amazon S3에서 데이터를 가져오고, SELECT INTO OUTFILE S3를 사용하여 S3로 데이터를 내보내는 데에도 사용됩니다. S3에서 가져오고 S3로 내보내는 방법에 대한 자세한 내용은 다음 내용을 참조하세요.

- [Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드](#)
- [Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장](#)

Aurora MySQL은 대부분의 Aurora MySQL DB 인스턴스 클래스(db.t2, db.t3, db.t4g와 같은 성능 버스트 가능 인스턴스 클래스 유형 제외)에 대한 오류 로그, 일반 로그, 느린 쿼리 로그 및 감사 로그에 별도의 영구 스토리지를 사용합니다. DB 인스턴스를 시작 및 중지할 때와 호스트를 교체하는 동안 이러한 볼륨의 데이터는 유지됩니다.

이 영구 스토리지 볼륨은 Amazon EBS에서도 지원되며 DB 인스턴스 클래스에 따라 크기가 고정되어 있습니다. 더 큰 DB 인스턴스 클래스를 사용하여 확장할 수는 없습니다.

다음 테이블에는 각 Aurora MySQL DB 인스턴스 클래스에 사용할 수 있는 임시 스토리지 및 영구 스토리지의 최대 용량이 나와 있습니다. Aurora에 대한 DB 인스턴스 클래스 지원의 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

DB 인스턴스 클래스	사용 가능한 최대 임시/로컬 스토리지(GiB)	로그 파일에 사용할 수 있는 추가적인 최대 스토리지(GiB)
db.x2g.16xlarge	1,280	500
db.x2g.12xlarge	960	500
db.x2g.8xlarge	640	500
db.x2g.4xlarge	320	500
db.x2g.2xlarge	160	60
db.x2g.xlarge	80	60
db.x2g.large	40	60
db.r8g.48xlarge	3840	500
db.r8g.24xlarge	1920	500
db.r8g.16xlarge	1,280	500
db.r8g.12xlarge	960	500
db.r8g.8xlarge	640	500
db.r8g.4xlarge	320	500

DB 인스턴스 클래스	사용 가능한 최대 임시/로컬 스토리지(GiB)	로그 파일에 사용할 수 있는 추가적인 최대 스토리지(GiB)
db.r8g.2xlarge	160	60
db.r8g.xlarge	80	60
db.r8g.large	32	60
db.r7i.48xlarge	3840	500
db.r7i.24xlarge	1920	500
db.r7i.16xlarge	1,280	500
db.r7i.12xlarge	960	500
db.r7i.8xlarge	640	500
db.r7i.4xlarge	320	500
db.r7i.2xlarge	160	60
db.r7i.xlarge	80	60
db.r7i.large	32	60
db.r7g.16xlarge	1,280	500
db.r7g.12xlarge	960	500
db.r7g.8xlarge	640	500
db.r7g.4xlarge	320	500
db.r7g.2xlarge	160	60
db.r7g.xlarge	80	60
db.r7g.large	32	60
db.r6i.32xlarge	2560	500

DB 인스턴스 클래스	사용 가능한 최대 임시/로컬 스토리지(GiB)	로그 파일에 사용할 수 있는 추가적인 최대 스토리지(GiB)
db.r6i.24xlarge	1920	500
db.r6i.16xlarge	1,280	500
db.r6i.12xlarge	960	500
db.r6i.8xlarge	640	500
db.r6i.4xlarge	320	500
db.r6i.2xlarge	160	60
db.r6i.xlarge	80	60
db.r6i.large	32	60
db.r6g.16xlarge	1,280	500
db.r6g.12xlarge	960	500
db.r6g.8xlarge	640	500
db.r6g.4xlarge	320	500
db.r6g.2xlarge	160	60
db.r6g.xlarge	80	60
db.r6g.large	32	60
db.r5.24xlarge	1920	500
db.r5.16xlarge	1,280	500
db.r5.12xlarge	960	500
db.r5.8xlarge	640	500
db.r5.4xlarge	320	500

DB 인스턴스 클래스	사용 가능한 최대 임시/로컬 스토리지(GiB)	로그 파일에 사용할 수 있는 추가적인 최대 스토리지(GiB)
db.r5.2xlarge	160	60
db.r5.xlarge	80	60
db.r5.large	32	60
db.r4.16xlarge	1,280	500
db.r4.8xlarge	640	500
db.r4.4xlarge	320	500
db.r4.2xlarge	160	60
db.r4.xlarge	80	60
db.r4.large	32	60
db.t4g.large	32	–
db.t4g.medium	32	–
db.t3.large	32	–
db.t3.medium	32	–
db.t3.small	32	–
db.t2.medium	32	–
db.t2.small	32	–

> **Important**
>
> 이러한 값은 이론적으로 각 DB 인스턴스에서 사용할 수 있는 스토리지의 최대 용량을 나타냅니다. 사용 가능한 실제 로컬 스토리지는 더 낮을 수 있습니다. Aurora의 관리 프로세스에 일부 로컬 스토리지가 사용되며 데이터 로드 전에도 DB 인스턴스에 일부 로컬 스토리지가 사용

됩니다. `FreeLocalStorage`에 설명된 [Amazon Aurora에 대한 Amazon CloudWatch 지표](#) CloudWatch CloudWatch 지표를 사용하여 특정 DB 인스턴스에 사용할 수 있는 임시 스토리지를 모니터링할 수 있습니다. 현재 시간의 여유 스토리지 용량을 확인할 수 있습니다. 시간대별 여유 스토리지 양을 차트로 표시할 수도 있습니다. 시간대별 여유 스토리지를 모니터링하면 값이 증가하는지 또는 감소하는지 확인하거나 최소값, 최대값 또는 평균값을 찾는 데 도움이 됩니다.
(Aurora Serverless v2에는 적용되지 않습니다.)

Aurora DB 클러스터 역추적

Amazon Aurora MySQL 호환 버전에서는 백업에서 데이터를 복구하지 않고도 특정 시간으로 DB 클러스터를 되감을 수 있습니다.

목차

역추적 개요

역추적은 DB 클러스터를 지정 시간으로 "되감습니다". 역추적은 어느 시점으로 복원하는 DB 클러스터 백업의 대체가 아닙니다. 다만 역추적은 기존의 백업 및 복원에 비해 다음과 같은 장점이 있습니다.

- 실수를 쉽게 실행 취소할 수 있습니다. WHERE 절 없이 DELETE 작업과 같이 안전하지 않은 작업을 실수로 수행한 경우, 서비스 중단을 최소화한 채로 안전하지 않은 작업 이전 시간으로 DB 클러스터를 역추적할 수 있습니다.
- DB 클러스터를 빠르게 역추적할 수 있습니다. DB 클러스터를 어느 시점으로 복원하면 새 DB 클러스터를 시작하면서 백업 데이터 혹은 DB 클러스터 스냅샷에서 복원하는데 이 과정이 몇 시간 걸릴 수 있습니다. DB 클러스터 역추적에 새 DB 클러스터는 필요하지 않으며 몇 분 만에 DB 클러스터를 되감습니다.
- 이전 데이터 변경 사항을 확인할 수 있습니다. 특정 데이터 변화가 언제 발생했는지 판단을 돕기 위해 DB 클러스터를 앞뒤 시간으로 반복하여 역추적할 수 있습니다. 예를 들면 DB 클러스터를 세 시간 역추적한 후 한 시간 앞으로 역추적할 수 있습니다. 이 경우 역추적 시간은 원래 시간 2시간 전입니다.

> **Note**
>
> DB 클러스터를 특정 시점으로 복원하기에 대한 자세한 정보는 [Aurora DB 클러스터 백업 및 복원에 대한 개요](#) 단원을 참조하십시오.

역추적 기간

역추적에는 대상 역추적 기간과 실제 역추적 기간이 있습니다.

- 대상 역추적 기간은 DB 클러스터를 역추적할 수 있는 기간입니다. 역추적을 활성화할 때 대상 역추적 기간을 지정하십시오. 예를 들어 하루 동안 DB 클러스터를 역추적할 수 있도록 하려면 대상 역추적 기간을 24시간으로 지정하십시오.
- 실제 역추적 기간은 DB 클러스터를 역추적할 수 있는 실제 기간으로, 대상 역추적 기간보다 작을 수 있습니다. 실제 역추적 기간은 변경 레코드라고 부르는 데이터베이스 변경 정보의 저장에 사용할 수 있는 워크로드 및 스토리지가 기반입니다.

역추적이 활성화된 Aurora DB 클러스터에 업데이트를 적용하면 변경 레코드가 생성됩니다. Aurora는 대상 역추적 기간 동안 변경 기록을 보관하고, 사용자는 저장에 대한 시간당 요금을 지불합니다. 저장하는 변경 레코드의 수는 대상 역추적 기간과 DB 클러스터의 워크로드 두 가지에 의해 결정됩니다. 워크로드는 정해진 시간에 DB 클러스터를 변경한 횟수입니다. 워크로드가 많으면 작을 때에 비해서 역추적 기간에 저장하는 변경 레코드가 많습니다.

대상 역추적 기간을 DB 클러스터의 역추적이 가능한 최대 시간의 목표로 봐도 좋습니다. 대부분의 경우 지정해 둔 최대 시간을 역추적할 수 있습니다. 그러나 일부의 경우 DB 클러스터가 최대 시간 동안

역추적할 변경 레코드를 충분히 저장하지 못하면 실제 역추적 기간이 대상 역추적 기간보다 작습니다. 일반적으로 DB 클러스터에 워크로드가 너무 많으면 실제 역추적 기간이 대상 역추적 기간보다 더 작습니다. 실제 역추적 기간이 대상 기간보다 작으면 사용자에게 알림을 보냅니다.

DB 클러스터에 역추적이 활성화된 상태에서 DB 클러스터에 저장한 테이블을 삭제하면 Aurora이 역추적 변경 레코드에 그 테이블을 보관합니다. 이는 테이블 삭제 이전 시점으로 되돌릴 수 있도록 하기 위함입니다. 역추적 기간에 테이블을 저장할 공간이 충분하지 않으면, 결국 역추적 변경 레코드에서 이 테이블이 제거될 수 있습니다.

역추적 시간

Aurora는 항상 DB 클러스터와 일관된 시간으로 역추적합니다. 그렇게 하면 역추적을 종료할 때 커밋되지 않은 트랜잭션의 가능성이 없어집니다. 역추적 시간을 지정할 때 Aurora는 자동으로 가장 가깝고 일관성 있는 시간을 선택합니다. 이러한 방식은 종료한 역추적이 지정 시간과 정확히 일치하지 않을 수도 있다는 의미입니다. 그러나 describe-db-cluster-backtracks AWS CLI 명령을 사용하면 정확한 역추적 시간을 정할 수 있습니다. 자세한 내용은 기존 역추적 검색 섹션을 참조하세요.

역추적 제한

역추적에는 다음과 같은 제한이 적용됩니다.

- 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에서만 역추적이 가능합니다. 역추적 기능을 활성화하도록 DB 클러스터를 수정할 수는 없습니다. 새 DB 클러스터를 만들거나 DB 클러스터의 스냅샷을 복원할 때 역추적 기능을 활성화할 수 있습니다.
- 역추적 기간 제한은 72시간입니다.
- 역추적은 전체 DB 클러스터에 영향을 미칩니다. 예를 들어 단일 테이블 또는 단일 데이터 업데이트를 선택적으로 역추적할 수 없습니다.
- 백트랙이 활성화된 클러스터에서는 크로스 리전 읽기 전용 복제본을 생성할 수 없지만 클러스터에서 이진 로그(binlog) 복제를 여전히 활성화할 수는 있습니다. 또한 이진 로그가 활성화된 DB 클러스터의 역추적을 시도할 경우, 역추적 강제 시행을 선택하지 않으면 일반적으로 오류가 발생합니다. 강제로 역추적을 시도하면 다운스트림 읽기 전용 복제본이 손상되고 블루/그린 배포 등의 다른 작업에 방해가 됩니다.
- 데이터베이스 복제본을 그 데이터베이스 복제본을 생성한 이전 시간으로 역추적할 수는 없습니다. 그러나 원본 데이터베이스를 이용하여 복제본 생성 이전 시간으로 역추적할 수 있습니다. 데이터베이스 복제에 대한 자세한 정보는 Aurora DB 클러스터에 대한 볼륨 복제 단원을 참조하십시오.
- 역추적하면 DB 인스턴스가 잠시 중단됩니다. 역추적 작업을 시작하기 전에는 새 읽기 또는 쓰기 요청이 없도록 애플리케이션을 중단하거나 일시 중지해야 합니다. 역추적 작업 중에 Aurora는 데이터

베이스를 일시 중지하고 열려 있는 연결을 닫으며 커밋되지 않은 읽기 및 쓰기를 모두 중단합니다. 그리고 역추적 작업이 완료되기를 기다립니다.

- 역추적을 지원하지 않는 AWS 리전에서는 역추적 사용 클러스터의 리전 간 스냅샷을 복원할 수 없습니다.
- Aurora MySQL 버전 2에서 버전 3으로 역추적을 활성화한 클러스터에 현재 위치 업그레이드를 수행하면 업그레이드가 발생하기 전의 시점으로 되돌릴 수 없습니다.

리전 및 버전 사용 가능 여부

Aurora PostgreSQL에서 역추적을 사용할 수 없습니다.

Aurora MySQL을 사용하는 역추적에 지원되는 엔진 및 리전 가용성은 다음과 같습니다.

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
미국 동부(버지니아 북부)	모든 버전	모든 버전
미국 동부(오하이오)	모든 버전	모든 버전
미국 서부(캘리포니아 북부)	모든 버전	모든 버전
미국 서부(오레곤)	모든 버전	모든 버전
아프리카(케이프타운)	–	–
아시아 태평양(홍콩)	–	–
아시아 태평양(자카르타)	–	–
아시아 태평양(말레이시아)	–	–

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
아시아 태평양(멜버른)	–	–
아시아 태평양(뭄바이)	모든 버전	모든 버전
아시아 태평양(오사카)	모든 버전	버전 2.07.3 이상
아시아 태평양(서울)	모든 버전	모든 버전
아시아 태평양(싱가포르)	모든 버전	모든 버전
아시아 태평양(시드니)	모든 버전	모든 버전
아시아 태평양(도쿄)	모든 버전	모든 버전
캐나다(중부)	모든 버전	모든 버전
캐나다 서부(캘거리)	–	–
중국(베이징)	–	–
중국(닝샤)	–	–
유럽(프랑크푸르트)	모든 버전	모든 버전
유럽(아일랜드)	모든 버전	모든 버전
유럽(런던)	모든 버전	모든 버전
유럽(밀라노)	–	–

리전	Aurora MySQL 버전 3	Aurora MySQL 버전 2
유럽(파리)	모든 버전	모든 버전
유럽(스페인)	–	–
유럽(스톡홀름)	–	–
유럽(취리히)	–	–
이스라엘(텔아비브)	–	–
중동(바레인)	–	–
중동(UAE)	–	–
남아메리카(상파울루)	–	–
AWS GovCloud(미국 동부)	–	–
AWS GovCloud(미국 서부)	–	–

역추적 지원 클러스터에 대한 업그레이드 고려 사항

Aurora MySQL 버전 3의 모든 마이너 버전에서 역추적이 지원되므로 백추적이 활성화된 DB 클러스터를 Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 수 있습니다.

Aurora MySQL DB 클러스터 역추적 구성

역추적 기능을 사용하려면 역추적을 활성화하고 대상 역추적 기간을 지정해야 합니다. 그렇지 않으면 역추적을 사용할 수 없습니다.

대상 역추적 기간의 경우, 역추적을 이용해서 데이터베이스를 되감으려는 시간을 지정하세요. Aurora는 해당 기간을 지원하기에 충분한 변경 레코드를 보관하려고 합니다.

콘솔

새 DB 클러스터를 생성할 때 콘솔을 사용하여 역추적을 구성할 수 있습니다. DB 클러스터를 수정하여 역추적 지원 클러스터의 역추적 기간을 변경할 수도 있습니다. 역추적 기간을 0으로 설정하여 클러스터에 대해 역추적을 완전히 해제하면 해당 클러스터에 대해 역추적을 다시 활성화할 수 없습니다.

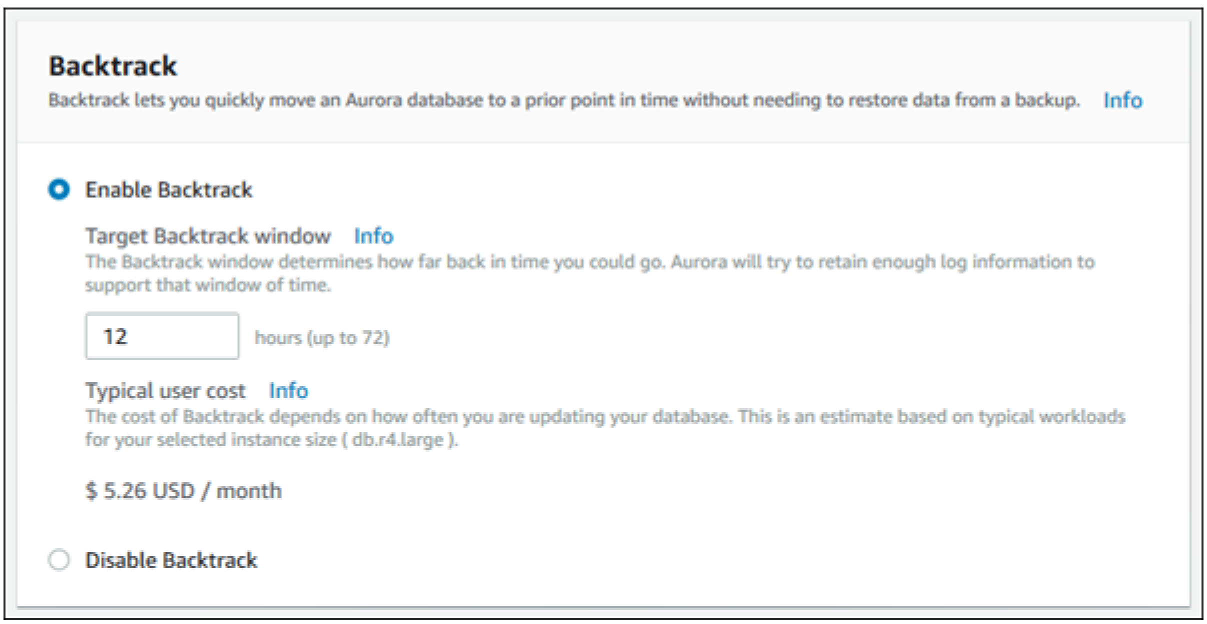
주제

- DB 클러스터 생성 시 콘솔로 역추적 구성
- DB 클러스터 수정 시 콘솔로 역추적 구성

DB 클러스터 생성 시 콘솔로 역추적 구성

새 Aurora MySQL DB 클러스터를 만들 때 Enable Backtrack(역추적 활성화)를 선택하고 역추적 섹션에서 Target Backtrack window(대상 역추적 기간) 값을 0보다 큰 값으로 지정하면 역추적이 구성됩니다.

DB 클러스터를 만들려면 Amazon Aurora DB 클러스터 생성 단원의 지침을 따르십시오. 다음 이미지는 역추적 섹션을 보여줍니다.



새 DB 클러스터를 생성할 때 Aurora에는 DB 클러스터의 워크로드에 대한 데이터가 없습니다. 그러므로 새 DB 클러스터를 대상으로 비용을 추정할 수 없습니다. 대신 콘솔이 일반 사용자에게 일반 워크로드를 바탕으로 지정 대상 역추적 기간에 대한 비용을 제시합니다. 일반 비용이란 역추적 기능 비용으로 주어지는 일반적인 참조 사항을 의미합니다.

> **Important**
>
> 실제 비용은 사용자의 DB 클러스터 워크로드를 바탕으로 하므로 실제 비용이 일반 비용과 일치하지 않을 수 있습니다.

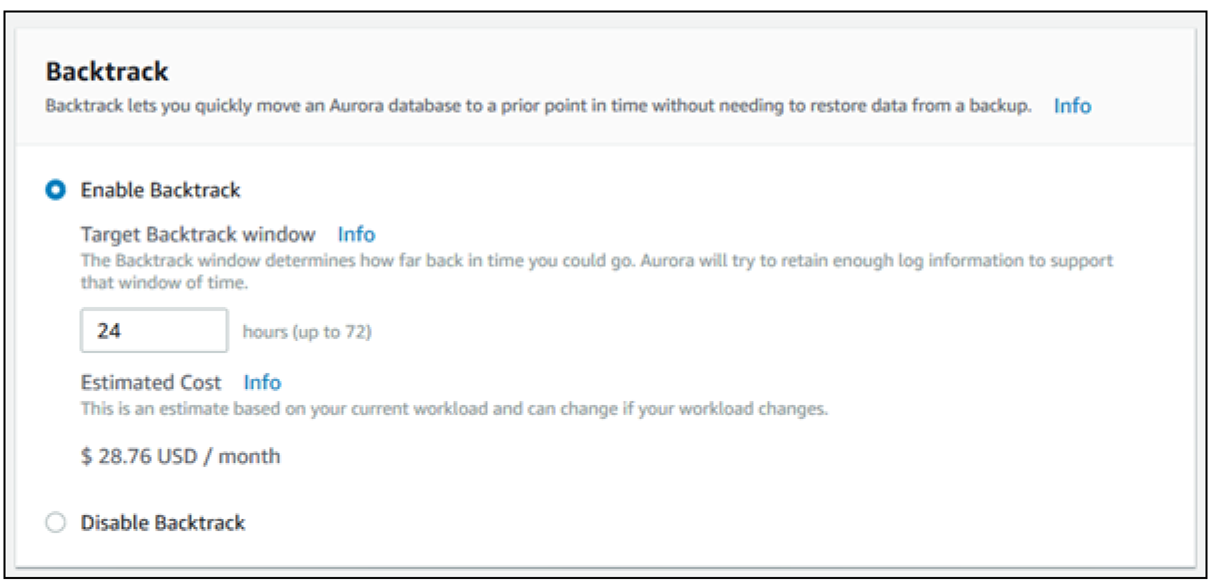
DB 클러스터 수정 시 콘솔로 역추적 구성

콘솔을 사용하여 DB 클러스터의 역추적을 수정할 수 있습니다.

> **Note**
>
> 현재, 역추적 기능이 활성화된 DB 클러스터에 대해서만 역추적을 수정할 수 있습니다. 역추적 기능이 DB 클러스터에서 비활성화된 경우 또는 역추적 기능이 비활성화된 상태에서 생성한 DB 클러스터에는 역추적 섹션이 보이지 않습니다.

콘솔을 사용하여 DB 클러스터의 역추적을 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 수정할 클러스터를 선택하고 수정을 선택합니다.
4. 대상 역추적 기간에서 역추적할 수 있는 기간을 수정하십시오. 한도는 72시간입니다.



DB 클러스터의 과거 워크로드를 바탕으로 사용자가 지정한 시간에 대한 추정 비용이 콘솔에 보입니다.

- DB 클러스터에 역추적을 비활성화한 경우, Amazon CloudWatch의 DB 클러스터에 대한 `VolumeWriteIOPS` 지표를 바탕으로 비용을 추정합니다.
- 이전에 DB 클러스터에 역추적을 활성화한 경우, Amazon CloudWatch의 DB 클러스터에 대한 `BacktrackChangeRecordsCreationRate` 지표를 바탕으로 비용을 추정합니다.

5. [Continue]를 선택합니다.
6. Scheduling of Modifications(수정 사항 예약)에서 다음 중 하나를 선택합니다.
   - Apply during the next scheduled maintenance window(다음 유지 관리 기간에 적용) – 다음 유지 관리 기간까지 기다린 후 Target Backtrack window(대상 역추적 기간) 수정을 적용합니다.
   - 즉시 적용 – Target Backtrack window(대상 역추적 기간) 수정을 가급적 빨리 적용합니다.
7. Modify cluster(클러스터 수정)를 선택합니다.

AWS CLI

create-db-cluster AWS CLI 명령으로 새 Aurora MySQL DB 클러스터를 생성할 때 `--backtrack-window` 값을 0보다 큰 값으로 지정하면 역추적이 구성됩니다. `--backtrack-window` 값이 대상 역추적 기간을 지정합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.

다음 `--backtrack-window` CLI 명령을 사용해 AWS 값을 지정할 수도 있습니다.

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

다음 절차에서는 AWS CLI를 사용하여 DB 클러스터의 대상 역추적 기간을 수정하는 방법을 설명합니다.

AWS CLI를 사용하여 DB 클러스터의 대상 역추적 기간을 수정하려면

- modify-db-cluster AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` – DB 클러스터의 이름입니다.

- `--backtrack-window` – DB 클러스터를 역추적할 수 있는 최대 시간(초)입니다.

다음 예는 `sample-cluster`의 대상 역추적 기간을 하루(86,400초)로 설정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
> 현재는 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에만 역추적을 활성화할 수 있습니다.

RDS API

CreateDBCluster Amazon RDS API 작업을 사용하여 새 Aurora MySQL DB 클러스터를 생성할 때 `BacktrackWindow` 값을 0보다 큰 값으로 지정하면 역추적이 구성됩니다. `BacktrackWindow` 값이 `DBClusterIdentifier` 값에 지정된 DB 클러스터의 대상 역추적 기간을 지정합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.

다음 API 작업으로 `BacktrackWindow` 값을 지정할 수도 있습니다.

- ModifyDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

> **Note**
> 현재는 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에만 역추적을 활성화할 수 있습니다.

Aurora MySQL DB 클러스터의 역추적 수행

DB 클러스터를 지정한 역추적 타임스탬프로 역추적할 수 있습니다. 역추적 타임스탬프가 가장 이른 역추적 시간의 이전이 아니고 미래인 경우, DB 클러스터는 그 타임스탬프로 역추적됩니다.

그렇지 않으면 오류가 발생하는 것이 보통입니다. 또한 이진 로그 기록이 활성화된 DB 클러스터의 역추적을 시도할 경우, 역추적 강제 시행을 선택하지 않으면 일반적으로 오류가 발생합니다. 역추적을 강제로 시행하면 이진 로그 기록을 사용하는 다른 작업이 중단될 수 있습니다.

> **Important**
> 역추적은 역추적으로 인한 변경 사항에 binlog 항목을 생성하지 않습니다. DB 클러스터에 이진 로그 기록이 활성화된 경우에 역추적을 하면 binlog 실행과 호환되지 않을 수도 있습니다.

> **Note**
> 데이터베이스 복제본의 경우 복제본이 생성된 날짜와 시간 이전으로 DB 클러스터를 역추적할 수 없습니다. 데이터베이스 복제에 대한 자세한 내용은 [Aurora DB 클러스터에 대한 볼륨 복제](#) 단원을 참조하세요.

콘솔

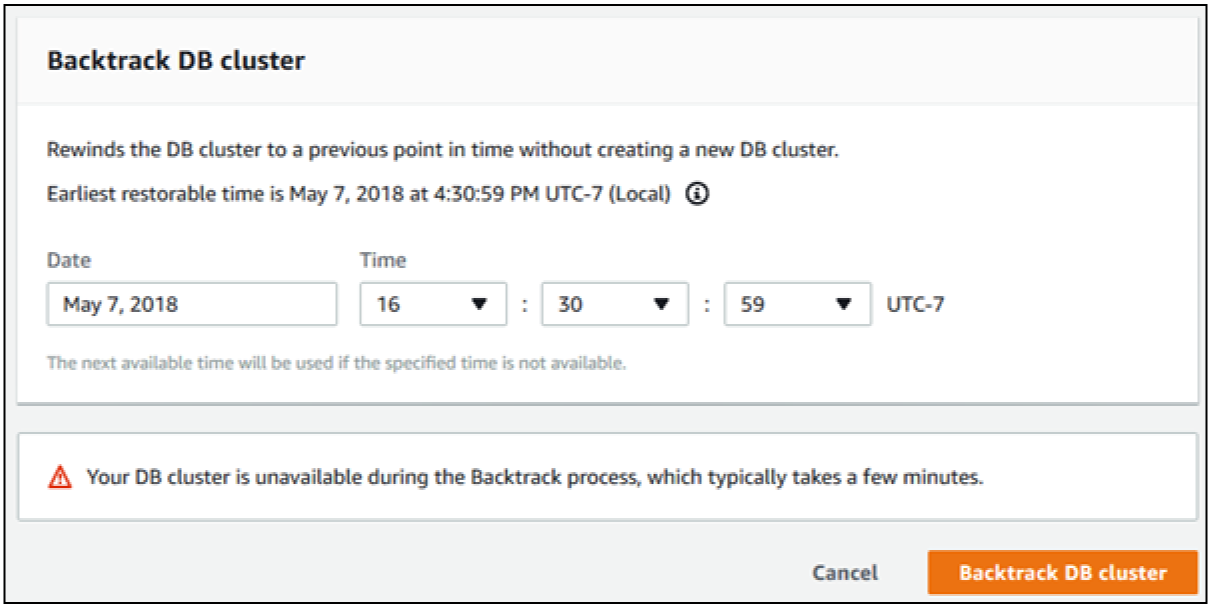
다음 절차에서는 콘솔을 사용하여 DB 클러스터의 역추적 작업을 수행하는 방법을 설명합니다.

콘솔로 역추적 작업을 수행하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Instances(인스턴스)를 선택합니다.
3. 역추적하려는 DB 클러스터의 기본 인스턴스를 선택합니다.

4. 작업에서 Backtrack DB cluster(DB 클러스터 역추적)를 선택합니다.

5. Backtrack DB cluster(DB 클러스터 역추적) 페이지에서 DB 클러스터를 역추적할 역추적 타임스탬프를 입력합니다.



6. DB 클러스터 역추적을 선택합니다.

AWS CLI

다음 절차에서는 AWS CLI를 사용하여 DB 클러스터를 역추적하는 방법에 대해 설명합니다.

AWS CLI를 사용하여 DB 클러스터를 역추적하려면

- backtrack-db-cluster AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` – DB 클러스터의 이름입니다.
  - `--backtrack-to` – DB 클러스터를 역추적할 역추적 타임스탬프로, ISO 8601 형식으로 지정됩니다.

  다음 예제에서는 DB 클러스터를 `sample-cluster` 2018년 3월 19일 오전 10시로 역추적합니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
    --backtrack-to 2018-03-19T10:00:00+00:00
```

Windows의 경우:

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터를 역추적하려면 [BacktrackDBCluster](#) 작업을 사용하십시오. 이 작업은 `DBClusterIdentifier` 값에 지정된 DB 클러스터를 지정 시간으로 역추적합니다.

Aurora MySQL DB 클러스터의 역추적 모니터링

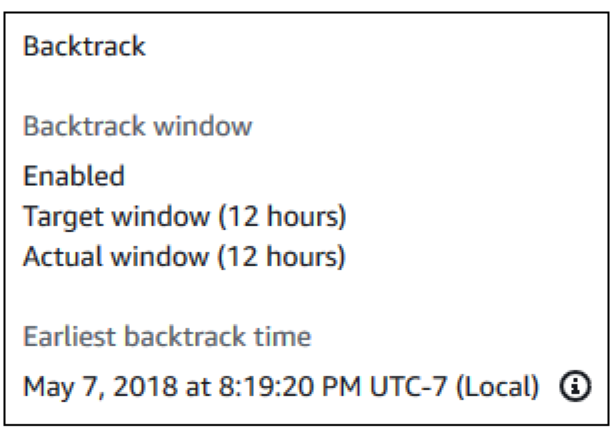
DB 클러스터에 대한 역추적 정보를 보고 역추적 지표를 모니터링할 수 있습니다.

콘솔

콘솔을 이용하여 역추적 정보를 보고 역추적 지표를 모니터링하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 정보를 열 DB 클러스터 이름을 선택합니다.

   역추적 정보는 역추적 섹션에 있습니다.

   역추적이 활성화되면 다음 정보가 제공됩니다.

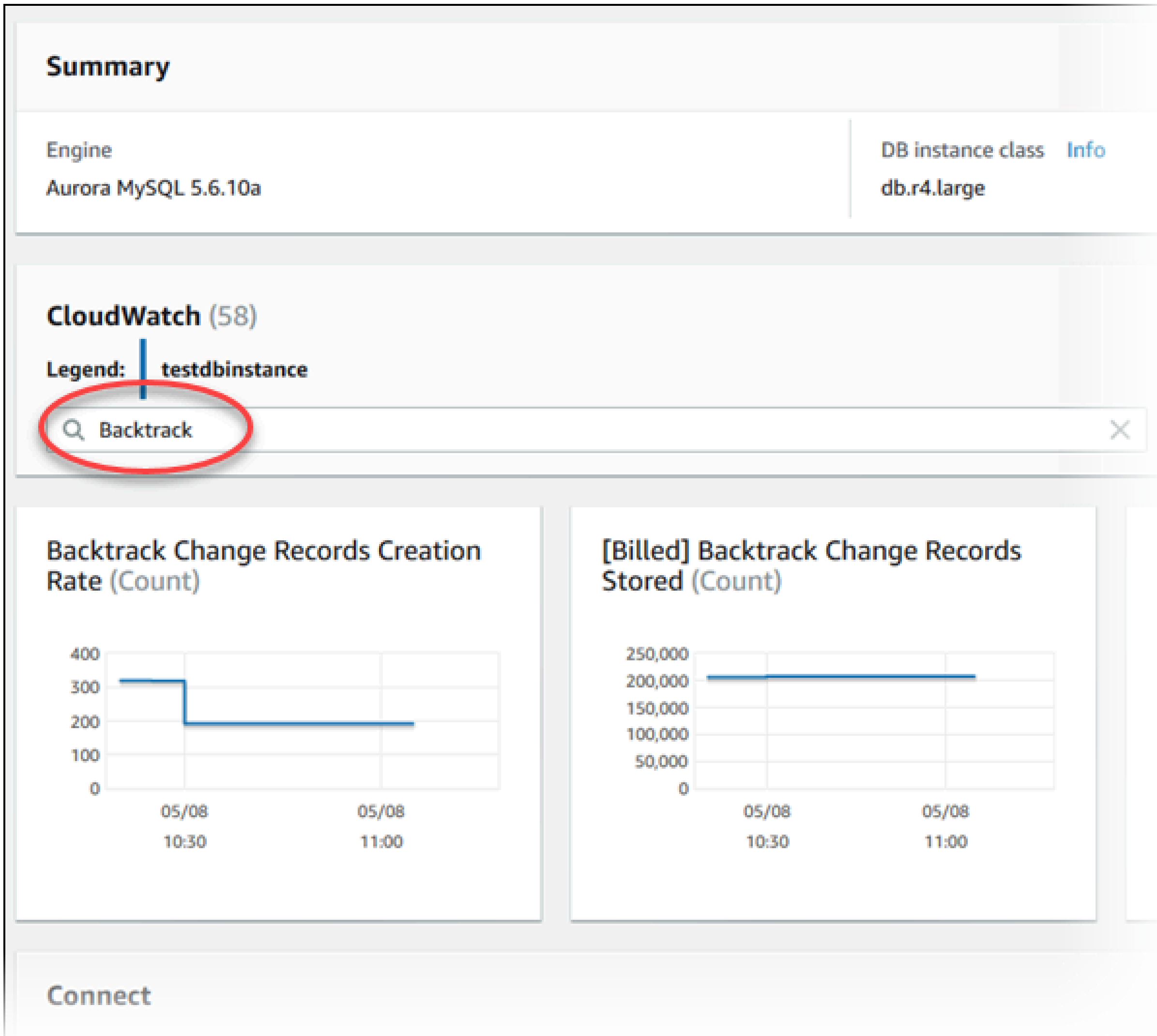
- Target window(대상 기간) – 대상 역추적 기간에 지정된 현재 시간입니다. 대상은 스토리지가 충분한 경우 역추적이 가능한 최대 시간입니다.
- Actual window(실제 기간) – 역추적할 수 있는 실제 기간으로, 대상 역추적 기간보다 작을 수 있습니다. 실제 역추적 기간은 역추적 변경 레코드 유지에 사용할 수 있는 워크로드 및 스토리지가 기반입니다.
- Earliest backtrack time(가장 빠른 역추적 시간) – DB 클러스터에 대하여 최대한 빠른 역추적 시간입니다. 표시된 시간 이전으로는 DB 클러스터를 역추적할 수 없습니다.

4. DB 클러스터의 역추적 지표를 보려면 다음과 같이 하십시오.

   a. 탐색 창에서 Instances(인스턴스)를 선택합니다.

   b. 정보를 표시할 DB 클러스터의 기본 인스턴스 이름을 선택합니다.

   c. CloudWatch 섹션에서 CloudWatch 상자에 **Backtrack**을 입력하여 역추적 지표만 표시하십시오.



다음 지표가 표시됩니다.

- Backtrack Change Records Creation Rate (Count)(역추적 변경 레코드 생성률(개수)) – 5분 동안 DB 클러스터에 생성한 역추적 변경 레코드의 수를 보여주는 지표입니다. 이 지표를 이용하여 대상 역추적 기간 동안 역추적 비용을 추정할 수 있습니다.
- [Billed] Backtrack Change Records Stored (Count)([요금 부과됨] 역추적 변경 레코드 저장됨(개수)) – DB 클러스터가 사용한 역추적 변경 레코드의 실제 수를 보여주는 지표입니다.
- Backtrack Window Actual (Minutes)(역추적 기간 실제(분)) – 대상 역추적 기간과 실제 역추적 기간에 차이가 나는지 보여주는 지표입니다. 예를 들어 대상 역추적 기간이 2시간(120분)인데 이 지표상으로 실제 역추적 기간이 100분인 경우 실제 역추적 기간이 대상 역추적 기간보다 작은 것입니다.
- Backtrack Window Alert (Count)(역추적 기간 알림(개수)) – 정해진 시간 동안 실제 역추적 기간이 대상 역추적 기간보다 작은 빈도를 보여주는 지표입니다.

> **Note**
>
> 다음 지표는 현재 시간보다 지연될 수 있습니다.
>
> - 역추적 변경 레코드 생성 속도(개수)
> - [요금 부과됨] 역추적 변경 레코드 저장됨(개수)

AWS CLI

다음 절차에서는 AWS CLI를 사용하여 DB 클러스터의 역추적 정보를 보는 방법을 설명합니다.

AWS CLI를 사용하여 DB 클러스터의 역추적 정보를 보려면

- describe-db-clusters AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` – DB 클러스터의 이름입니다.

  다음은 `sample-cluster`의 역추적 정보 목록의 예입니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  Windows의 경우:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터의 역추적 정보를 보려면 DescribeDBClusters 작업을 사용하십시오. 이 작업은 `DBClusterIdentifier` 값에 지정된 DB 클러스터의 역추적 정보를 반환합니다.

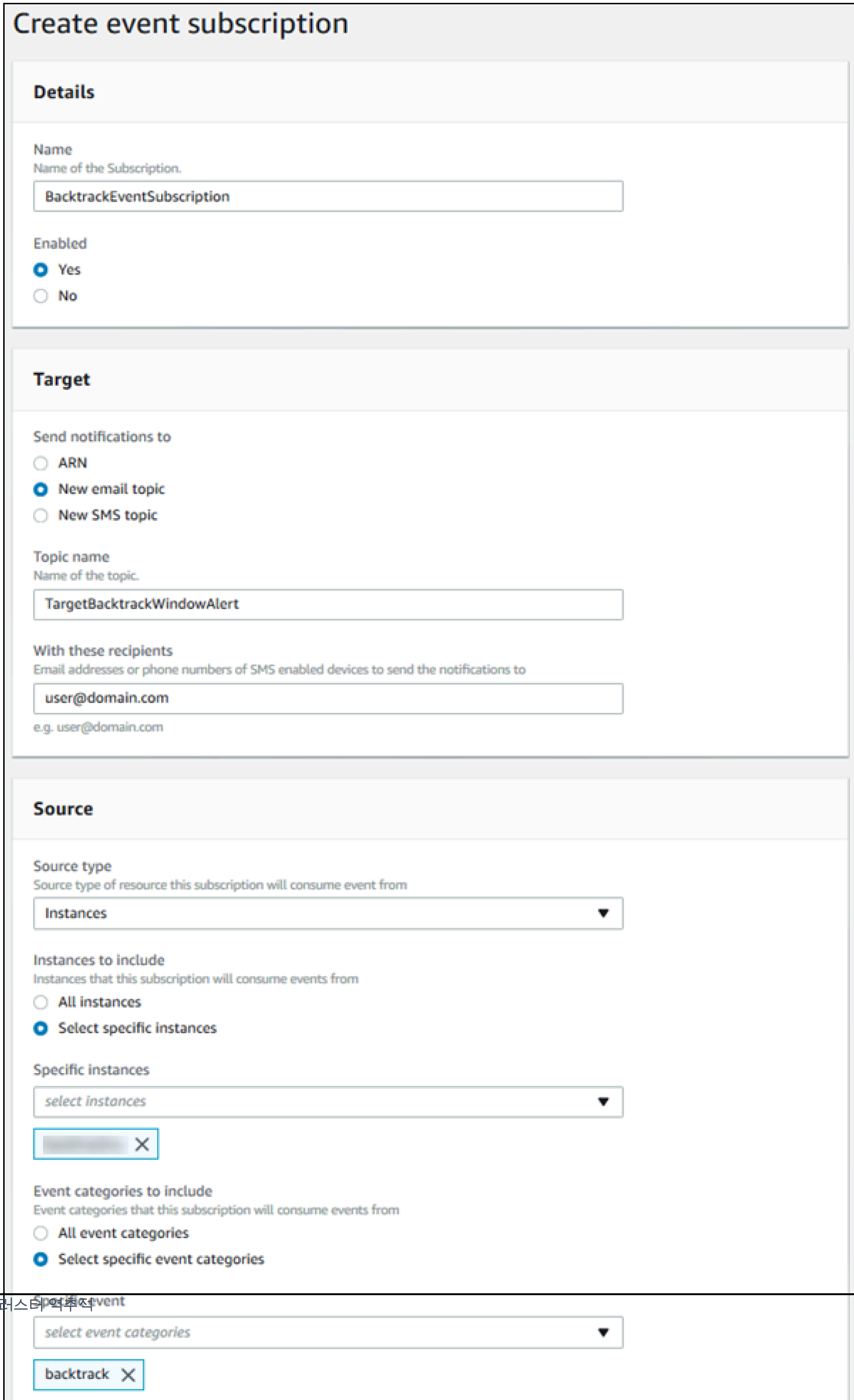
콘솔로 역추적 이벤트 구독

다음 절차에서는 콘솔을 사용하여 역추적 이벤트를 구독하는 방법을 설명합니다. 실제 역추적 기간이 대상 역추적 기간보다 작으면 이벤트가 사용자에게 이메일이나 문자 알림을 보냅니다.

콘솔을 사용하여 역추적 정보를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 이벤트 구독을 선택합니다.
3. 이벤트 구독 생성을 선택합니다.
4. 이름 상자 안에 이벤트 구독에 대한 이름을 입력하고 활성화에서 꼭 예를 선택합니다.
5. 대상 섹션에서 새로운 이메일 주제를 선택합니다.
6. 주제 이름에 주제의 이름을 입력하고, 수신자에는 알림을 받을 이메일 주소 혹은 전화번호를 입력합니다.
7. 소스 섹션에서 소스 유형에 인스턴스를 선택합니다.
8. 포함할 인스턴스에 Select specific instances(특정 인스턴스 선택)을 선택하고 DB 인스턴스를 선택합니다.
9. 포함할 이벤트 범주에 특정 이벤트 범주 선택을 선택하고 역추적을 선택합니다.

   이제 페이지가 다음 페이지와 같아야 합니다.


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
Specific event
select event categories
backtrack

10. Create(생성)를 선택합니다.

기존 역추적 검색

DB 클러스터의 기존 역추적 정보를 검색할 수 있습니다. 이 정보에는 역추적의 고유 식별자, 역추적한 양방향의 날짜와 시간, 역추적을 요청한 날짜와 시간, 역추적의 현재 상태 등이 포함됩니다.

> **Note**
> 현재는 콘솔을 사용하여 기존 역추적을 검색할 수 없습니다.

AWS CLI

다음 절차에서는 AWS CLI를 사용하여 DB 클러스터의 기존 역추적을 검색하는 방법을 설명합니다.

AWS CLI를 사용하여 기존 역추적을 검색하려면

- describe-db-cluster-backtracks AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` – DB 클러스터의 이름입니다.

  다음은 `sample-cluster`의 기존 역추적을 검색하는 예입니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  Windows의 경우:

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터의 역추적 정보를 검색하려면 DescribeDBClusterBacktracks 작업을 사용하십시오. 이 작업은 `DBClusterIdentifier` 값에 지정된 DB 클러스터의 역추적 정보를 반환합니다.

Aurora MySQL DB 클러스터의 역추적 비활성화

DB 클러스터의 역추적 기능을 비활성화할 수 있습니다.

콘솔

콘솔을 사용하여 DB 클러스터의 역추적을 비활성화할 수 있습니다. 클러스터에 대해 역추적을 완전히 해제한 후에는 해당 클러스터에 대해 다시 활성화할 수 없습니다.

콘솔을 사용하여 DB 클러스터의 역추적 기능을 비활성화하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 수정할 클러스터를 선택하고 수정을 선택합니다.
4. 역추적 섹션에서 Disable Backtrack(역추적 비활성화)을 선택합니다.
5. [Continue]를 선택합니다.
6. Scheduling of Modifications(수정 사항 예약)에서 다음 중 하나를 선택합니다.
   - Apply during the next scheduled maintenance window(예약된 다음 유지 관리 기간에 적용) –수정 사항의 적용을 다음 유지 관리 기간까지 기다립니다.
   - 즉시 적용 – 수정 사항을 가능한 한 빨리 적용합니다.
7. [Modify Cluster]를 선택합니다.

AWS CLI

대상 역추적 기간을 AWS CLI(제로)으로 설정하면 `0`를 사용하여 DB 클러스터의 역추적 기능을 비활성화할 수 있습니다. 클러스터에 대해 역추적을 완전히 해제한 후에는 해당 클러스터에 대해 다시 활성화할 수 없습니다.

AWS CLI를 사용하여 DB 클러스터의 대상 역추적 기간을 수정하려면

- modify-db-cluster AWS CLI 명령을 호출하고 다음 값을 입력합니다.

- `--db-cluster-identifier` – DB 클러스터의 이름입니다.
- `--backtrack-window` –은(는) 역추적을 해제하도록 `0`을(를) 지정합니다.

다음은 `sample-cluster`를 `--backtrack-window`으로 설정함으로써 `0`의 역추적 기능을 비활성화하는 예제입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터의 역추적 기능을 비활성화하려면 ModifyDBCluster 작업을 사용하십시오. `BacktrackWindow` 값을 `0`(제로)으로 설정하고, `DBClusterIdentifier` 값에서 DB 클러스터를 지정합니다. 클러스터에 대해 역추적을 완전히 해제한 후에는 해당 클러스터에 대해 다시 활성화할 수 없습니다.

오류 삽입 쿼리를 사용하여 Amazon Aurora MySQL 테스트

오류 삽입 쿼리를 사용하여 Aurora MySQL DB 클러스터의 내결함성을 테스트할 수 있습니다. 오류 삽입 쿼리는 Amazon Aurora 인스턴스에 SQL 명령으로 발행됩니다. 다음 이벤트 중 하나의 발생 시뮬레이션을 예약할 수 있습니다.

- 라이터 또는 리더 DB 인스턴스의 충돌
- Aurora 복제본의 실패
- 디스크 실패
- 디스크 정체

오류 삽입 쿼리가 충돌을 지정하면 Aurora MySQL DB 인스턴스가 강제로 충돌합니다. 다른 오류 삽입 쿼리로 인해서도 오류 이벤트 시뮬레이션이 발생하지만 이 경우 이벤트는 발생하지 않습니다. 오류 삽입 쿼리를 제출할 때 실패 이벤트 시뮬레이션이 발생하는 시간 길이를 지정할 수도 있습니다.

Aurora 복제본의 엔드포인트에 연결하여 오류 삽입 쿼리를 Aurora 복제본 중 하나로 제출할 수 있습니다. 자세한 내용은 Amazon Aurora 엔드포인트 연결 단원을 참조하십시오.

오류 삽입 쿼리를 실행하려면 마스터 사용자의 모든 권한이 필요합니다. 자세한 내용은 마스터 사용자 계정 권한 단원을 참조하십시오.

인스턴스 충돌 테스트

`ALTER SYSTEM CRASH` 오류 삽입 쿼리를 사용하여 Amazon Aurora 인스턴스의 충돌을 강제로 일으킬 수 있습니다.

이 오류 삽입 쿼리의 경우 장애 조치가 발생하지 않습니다. 장애 조치를 테스트하려면 RDS 콘솔에서 DB 클러스터에 대한 장애 조치(Failover) 인스턴스 작업을 선택하거나 failover-db-cluster AWS CLI 명령 또는 FailoverDBCluster RDS API 작업을 사용합니다.

구문

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

옵션

이 오류 삽입 쿼리에는 다음 충돌 유형 중 하나가 사용됩니다.

- **`INSTANCE`** — Amazon Aurora인스턴스용 MySQL 호환 데이터베이스의 충돌이 시뮬레이션됩니다.
- **`DISPATCHER`** — Aurora DB 클러스터용 라이터 인스턴스에서 디스패처 충돌이 시뮬레이션됩니다. 디스패처가 Amazon Aurora DB 클러스터용 클러스터 볼륨에 업데이트 쓰기 작업을 합니다.
- **`NODE`** —MySQL 호환 데이터베이스 및 Amazon Aurora 인스턴스용 디스패처의 충돌이 모두 시뮬레이션됩니다. 이 오류 삽입 시뮬레이션의 경우 캐시도 삭제됩니다.

기본 충돌 유형은 `INSTANCE`입니다.

Aurora 복제본 실패 테스트

`ALTER SYSTEM SIMULATE READ REPLICA FAILURE` 오류 삽입 쿼리를 사용하여 Aurora 복제본의 실패를 시뮬레이션할 수 있습니다.

Aurora 복제본 실패가 발생하면 라이터 인스턴스에서 Aurora 복제본 또는 DB 클러스터의 모든 Aurora 복제본에 대한 요청 전체를 지정된 시간 간격 동안 차단합니다. 시간 간격이 경과되면 해당 Aurora 복제본이 라이터 인스턴스와 자동으로 동기화됩니다.

구문

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

- **percentage_of_failure** — 실패 이벤트 도중 차단되는 요청 비율(%). 이 값은 0 ~ 100의 실수(Double)입니다. 0을 지정하면 요청이 차단되지 않습니다. 100을 지정하면 모든 요청이 차단됩니다.
- 결함 유형 — 시뮬레이션할 결함의 유형입니다. DB 클러스터의 모든 Aurora 복제본에 대한 실패를 시뮬레이션하려면 `TO ALL` 을 지정합니다. 단일 Aurora 복제본의 실패를 시뮬레이션하려면 `TO`와 Aurora 복제본의 이름을 지정하십시오. 기본 실패 유형은 `TO ALL`입니다.
- **quantity**—Aurora 복제본 결함 시뮬레이션 시간 길이. 이 간격 값 뒤에 시간 단위를 지정해야 합니다. 지정된 단위의 시간 동안 시뮬레이션이 발생합니다. 예를 들어 `20 MINUTE`를 지정하면 20분 동안 시뮬레이션이 실행됩니다.

> **Note**
>
> Aurora 복제본 실패 이벤트에 대한 시간 간격을 지정할 때는 각별히 주의하십시오. 시간 간격을 너무 길게 지정하고 라이터 인스턴스가 실패 이벤트 중에 다량의 데이터를 기록하는 경우, Aurora DB 클러스터에서 Aurora 복제본이 충돌했다고 간주하여 해당 복제본을 교체할 수도 있습니다.

디스크 실패 테스트

`ALTER SYSTEM SIMULATE DISK FAILURE` 오류 삽입 쿼리를 사용하여 Aurora DB 클러스터에 대한 디스크 실패를 시뮬레이션할 수 있습니다.

디스크 실패 시뮬레이션 중에는 Aurora DB 클러스터에서 임의로 디스크 세그먼트를 오류 상태로 표시합니다. 시뮬레이션 기간 동안 이러한 세그먼트에 대한 요청이 차단됩니다.

구문

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

- **percentage_of_failure** — 실패 이벤트 중에 오류 상태로 표시할 디스크의 비율(%). 이 값은 0 ~ 100의 실수(Double)입니다. 0을 지정하면 디스크의 어떤 부분도 오류 상태로 표시되지 않습니다. 100을 지정하면 전체 디스크가 오류 상태로 표시됩니다.
- **DISK index** — 실패 이벤트를 시뮬레이션할 특정 논리 데이터 블록. 가용 논리 데이터 블록의 범위를 초과하는 경우, 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 자세한 내용은 Aurora MySQL DB 클러스터를 위한 볼륨 상태 표시 섹션을 참조하세요.
- **NODE index** — 실패 이벤트를 시뮬레이션할 특정 스토리지 노드. 가용 스토리지 노드의 범위를 초과하는 경우, 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 자세한 내용은 Aurora MySQL DB 클러스터를 위한 볼륨 상태 표시 섹션을 참조하세요.
- **quantity** — 디스크 실패를 시뮬레이션하는 시간 길이. 이 간격 값 뒤에 시간 단위를 지정해야 합니다. 지정된 단위의 시간 동안 시뮬레이션이 발생합니다. 예를 들어 `20 MINUTE`를 지정하면 20분 동안 시뮬레이션이 실행됩니다.

디스크 정체 테스트

`ALTER SYSTEM SIMULATE DISK CONGESTION` 오류 삽입 쿼리를 사용하여 Aurora DB 클러스터에 대한 디스크 실패를 시뮬레이션할 수 있습니다.

디스크 정체 시뮬레이션 중에는 Aurora DB 클러스터에서 임의로 디스크 세그먼트를 정체 상태로 표시합니다. 시뮬레이션 기간 동안 지정된 최소 지연 시간과 최대 지연 시간 사이에서 이러한 세그먼트에 대한 요청이 지연됩니다.

구문

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
```

```
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

- **`percentage_of_failure`** — 실패 이벤트 중에 정체 상태로 표시할 디스크의 비율(%). 이 값은 0 ~ 100의 실수(Double)입니다. 0을 지정하면 디스크의 어떤 부분도 정체 상태로 표시되지 않습니다. 100을 지정하면 전체 디스크가 정체 상태로 표시됩니다.
- **`DISK index`** 또는 **`NODE index`** — 실패 이벤트를 시뮬레이션할 특정 디스크 또는 노드. 디스크 또는 노드의 인덱스 범위를 초과할 경우 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다.
- **`minimum`** 및 **`maximum`** — 최대 및 최소 정체 지연 시간(밀리초). 정체 상태로 표시된 디스크 세그먼트가 시뮬레이션 기간 중에 최소 밀리초 시간부터 최대 밀리초 시간까지의 시간 범위 내에서 임의 시간 동안 지연됩니다.
- **`quantity`** —디스크 정체를 시뮬레이션하는 시간 길이. 이 간격 값 뒤에 시간 단위를 지정해야 합니다. 지정된 단위의 시간 동안 시뮬레이션이 발생합니다. 예를 들어 `20 MINUTE`를 지정하면 20분 동안 시뮬레이션이 실행됩니다.

빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정

Amazon Aurora에는 거의 즉각적으로 현재 위치에서 `ALTER TABLE` 작업을 실행할 최적화 기능이 포함되어 있습니다. 이 작업은 테이블을 복사하거나 다른 DML 명령문에 영향을 거의 주지 않고 완료됩니다. 이 작업은 테이블 복사를 위해 임시 스토리지를 사용하지 않으므로 스몰 인스턴스 클래스의 라지 테이블에 대해서도 DDL 문을 유용하게 만듭니다.

Aurora MySQL 버전 3은 인스턴트 DDL이라는 MySQL 8.0 기능과 호환됩니다. Aurora MySQL 버전 2는 빠른 DDL이라는 다른 구현을 사용합니다.

주제

인스턴트 DDL(Aurora MySQL 버전 3)

일부 DDL 작업의 효율성을 향상시키기 위해 Aurora MySQL 버전 3에서 수행하는 최적화를 인스턴트 DDL이라고 합니다.

Aurora MySQL 버전 3은 커뮤니티 MySQL 8.0의 인스턴트 DDL과 호환됩니다. `ALTER TABLE` 문이 포함된 `ALGORITHM=INSTANT` 절을 사용하여 인스턴트 DDL 작업을 수행합니다. 인스턴트 DDL에 대한 구문 및 사용 세부 정보는 MySQL 설명서의 [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) 및 [Online DDL Operations](https://dev.mysql.com/doc/refman/8.0/en/innodb-online-ddl-operations.html)(온라인 DDL 작업)을 참조하세요.

다음 예에서는 인스턴트 DDL 기능을 설명합니다. `ALTER TABLE` 문은 열을 추가하고, 기본 열 값을 변경합니다. 예에는 일반 열과 가상 열, 그리고 일반 테이블 및 파티션된 테이블이 모두 포함됩니다. 각 단계에서 `SHOW CREATE TABLE` 및 `DESCRIBE` 문을 발행하여 결과를 확인할 수 있습니다.

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
```

```
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

빠른 DDL(Aurora MySQL 버전 2)

Aurora MySQL의 빠른 DDL은 가동 중지 시간과 리소스 사용량을 줄여 열 추가 또는 삭제와 같은 특정 스키마 변경의 성능을 개선하도록 설계된 최적화 기능입니다. 이를 통해 기존 DDL 방법에 비해 이러한 작업을 더 효율적으로 완료할 수 있습니다.

> **Important**
>
> 현재 빠른 DDL을 사용하려면 Aurora 랩 모드를 활성화해야 합니다. 랩 모드 활성화에 대한 자세한 내용은 [Amazon Aurora MySQL 랩 모드](#) 섹션을 참조하세요.

특정 DDL 작업의 효율성을 개선하기 위해 Aurora MySQL 버전 2의 랩 모드에서 빠른 DDL 최적화가 처음으로 도입되었습니다. Aurora MySQL 버전 3에서는 랩 모드가 중단되었으며 빠른 DDL이 MySQL 8.0 인스턴트 DDL 기능으로 대체되었습니다.

MySQL에서 많은 데이터 정의 언어(DDL) 작업이 상당한 성능 임팩트를 가집니다.

예를 들어 `ALTER TABLE` 작업을 이용해 열을 테이블에 추가한다고 가정해 보겠습니다. 지정된 알고리즘에 따라 이 작업은 다음을 포함할 수 있습니다.

- 테이블의 전체 사본 생성
- 동시에 발생하는 데이터 조작 언어(DML) 작업을 처리하는 임시 테이블 생성
- 테이블용의 모든 인덱스 리빌드
- 동시에 발생하는 DML 변경 사항을 적용하면서 테이블 록 적용
- 동시에 발생하는 DML 처리량 표시

이러한 성능 영향은 테이블이 크거나 트랜잭션 볼륨이 많은 환경에서 특히 문제가 될 수 있습니다. 빠른 DDL은 스키마 변경을 최적화하여 리소스를 덜 사용하면서 더 빠르게 작업을 완료할 수 있도록 함으로써 이러한 문제를 완화하는 데 도움이 됩니다.

빠른 DDL 제한

현재 빠른 DDL에는 다음과 같은 제한 사항이 있습니다.

- 빠른 DDL은 기존 테이블 끝까지 기본값 없이 null이 허용된 열 추가만 지원합니다.
- 빠른 DDL은 분할된 테이블에서 작동하지 않습니다.
- 빠른 DDL은 중복 행 형식을 사용하는 InnoDB 테이블을 지원하지 않습니다.
- 전체 텍스트 검색 인덱스가 있는 테이블에는 빠른 DDL이 작동하지 않습니다.
- DDL 작업에 대한 최대 가능한 레코드 크기가 너무 크면 빠른 DDL이 사용되지 않습니다. 레코드 크기가 페이지 크기의 절반보다 큰 경우 해당 레코드가 너무 큽니다. 레코드의 최대 크기는 모든 열의 최대 크기를 더하여 계산합니다. 가변 크기 열의 경우에는 InnoDB 표준에 따라 extern 바이트가 계산에 포함되지 않습니다.

빠른 DDL 구문

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

이 명령문의 옵션은 다음과 같습니다.

- **tbl_name** — 수정할 테이블 이름.
- **col_name** — 추가할 열 이름.
- **col_definition** — 추가할 열 정의.

> **Note**
>
> 기본값 없이 null이 허용된 열 정의를 지정해야 합니다. 그렇지 않으면 빠른 DDL을 사용할 수 없습니다.

빠른 DDL 예제

다음 예시에서는 빠른 DDL 작업의 속도 향상을 보여줍니다. 첫 번째 SQL 예시에서는 빠른 DDL을 사용하지 않고 큰 테이블에서 `ALTER TABLE` 선언을 실행합니다. 이 작업에는 상당한 시간이 걸립니다. CLI 예시는 클러스터용 빠른 DDL을 활성화하는 방법을 보여줍니다. 그리고 다른 SQL 예제는 동일한 테이블에서 동일한 `ALTER TABLE` 명령문을 실행합니다. 빠른 DDL을 활성화하면 작업이 매우 빨라집니다.

이 예제에서는 1억 5천만 개의 행을 포함하는 TPC-H 벤치마크의 `ORDERS` 테이블을 사용합니다. 이 클러스터는 의도적으로 비교적 작은 인스턴스 클래스를 사용하여 빠른 DDL을 사용할 수 없을 때 `ALTER TABLE` 선언이 얼마나 오래 걸릴 수 있는지 보여줍니다. 이 예제에서는 동일한 데이터를 포함하는 원본 테이블의 클론을 생성합니다. 이 `aurora_lab_mode` 설정을 확인하면 랩 모드를 사용할 수 없기 때문에 클러스터가 빠른 DDL을 사용할 수 없음을 확인합니다. 그리고 `ALTER TABLE ADD COLUMN` 선언은 테이블 끝에 새 열을 추가하는 데 상당한 시간이 소요됩니다.

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 0 |
+-------------------+
```

```
mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

이 예제는 이전 예제와 동일한 대형 테이블을 준비합니다. 그러나 대화형 SQL 세션 내에서 랩 모드를 사용하도록 설정할 수는 없습니다. 사용자 지정 파라미터 그룹에서 해당 설정을 사용하도록 설정해야 합니다. 이렇게 하려면 `mysql` 세션에서 전환하고 일부 AWS CLI 명령을 실행하거나 AWS Management Console를 사용합니다.

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

클러스터에 대해 랩 모드를 사용하려면 파라미터 그룹을 사용한 일부 작업이 필요합니다. 이 AWS CLI 예제에서는 클러스터 파라미터 그룹을 사용하여 클러스터의 모든 DB 인스턴스가 랩 모드 설정에 동일한 값을 사용하게 합니다.

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
```

```
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

다음 예제에서는 파라미터 그룹 변경 사항이 적용된 후 나머지 단계를 보여 줍니다. 클러스터가 빠른 DDL을 사용할 수 있는지 확인하기 위해 aurora_lab_mode 설정을 테스트합니다. 그런 다음 ALTER TABLE 진술을 실행하여 다른 큰 테이블의 끝에 열을 추가합니다. 이번에는 진술이 매우 빨리 끝납니다.

```
mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 1 |
+-------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
```

```
Query OK, 0 rows affected (0.40 sec)
```

Aurora MySQL DB 클러스터를 위한 볼륨 상태 표시

Amazon Aurora에서 DB 클러스터 볼륨은 논리 블록의 모음으로 구성됩니다. 이 각각은 할당된 스토리지의 10기가바이트를 나타냅니다. 이러한 블록을 보호 그룹이라고 합니다.

각 보호 그룹의 데이터는 스토리지 노드라고 하는 6개의 물리 스토리지 장치에 두루 복제됩니다. 이러한 스토리지 노드는 DB 클러스터가 상주하는 AWS 리전의 3개 가용 영역(AZ)에 할당됩니다. 또한 각 스토리지 노드에는 DB 클러스터 볼륨에 대해 1개 이상의 논리 데이터 블록이 포함됩니다. 보호 그룹 및 스토리지 노드에 대해 자세히 알아보려면 AWS 데이터베이스 블로그에서 Introducing the Aurora storage engine을 참조하세요.

전체 스토리지 노드 또는 스토리지 노드 내부의 단일 논리 데이터 블록의 장애를 시뮬레이션할 수 있습니다. 이를 위해 `ALTER SYSTEM SIMULATE DISK FAILURE` 오류 주입문을 사용합니다. 이 문의 경우 쿼리에서 특정 논리 데이터 블록 또는 스토리지 노드의 인덱스 값을 지정합니다. 그러나 DB 클러스터 볼륨이 사용하는 논리 데이터 블록 또는 스토리지 노드의 수보다 큰 인덱스 값을 지정하면, 이 문은 오류를 반환합니다. 오류 삽입 쿼리에 대한 자세한 내용은 오류 삽입 쿼리를 사용하여 Amazon Aurora MySQL 테스트 단원을 참조하십시오.

`SHOW VOLUME STATUS` 문을 사용하여 오류를 피할 수 있습니다. 이 문은 두 서버 상태 변수, `Disks` 및 `Nodes`를 반환합니다. 이러한 변수는 DB 클러스터 블록에 대해 각각 논리 데이터 블록과 스토리지 노드의 총 수를 표시합니다.

구문

```
SHOW VOLUME STATUS
```

예

다음 예는 전형적인 SHOW VOLUME STATUS 결과를 보여줍니다.

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
```

```
+----------------+--------+
```

Aurora MySQL 튜닝

대기 이벤트 및 스레드 상태는 Aurora MySQL의 중요한 튜닝 도구입니다. 세션이 리소스를 기다리는 이유와 어떤 작업을 수행하는지 알 수 있다면 병목 현상을 더 잘 줄일 수 있습니다. 이 섹션의 정보를 통해 가능한 원인과 해결 조치를 찾을 수 있습니다.

Amazon DevOps Guru for RDS는 Aurora MySQL 데이터베이스가 나중에 더 큰 문제를 일으킬 수 있는 문제를 겪고 있는지를 사전에 판단할 수 있습니다. Amazon DevOps Guru for RDS는 사전 예방 인사이트에 수정 조치에 대한 설명과 권장 사항을 게시합니다. 이 섹션에는 일반적인 문제에 대한 인사이트가 포함되어 있습니다.

> Important
>
> 이 섹션의 대기 이벤트 및 스레드 상태는 Aurora MySQL에만 해당됩니다. 이 섹션의 정보는 Amazon RDS for MySQL이 아닌 Amazon Aurora만 튜닝하는 데 사용하세요.
> 이 섹션에서 다루는 일부 대기 이벤트는 이러한 데이터베이스 엔진의 오픈 소스 버전에서는 유사한 것이 없습니다. 다른 대기 이벤트는 오픈 소스 엔진의 이벤트와 이름이 같지만 다르게 동작합니다. 예를 들어, Amazon Aurora 스토리지는 오픈 소스 스토리지와 다르게 작동하므로 스토리지 관련 대기 이벤트는 리소스 상태가 다르다는 것을 나타냅니다.

주제

- Aurora PostgreSQL 튜닝을 위한 필수 개념
- 대기 이벤트로 Aurora MySQL 튜닝
- 스레드 상태로 Aurora MySQL 튜닝
- Amazon DevOps Guru의 사전 예방 인사이트를 활용하여 Aurora MySQL 조정

Aurora PostgreSQL 튜닝을 위한 필수 개념

Aurora MySQL 데이터베이스를 튜닝하기 전에 대기 이벤트 및 스레드 상태가 어떠한지, 왜 이런 일이 발생하는지 확인하세요. 또한 InnoDB 스토리지 엔진을 사용할 때 Aurora MySQL의 기본 메모리 및 디스크 아키텍처도 검토하세요. 유용한 아키텍처 다이어그램에 대해서는 MySQL 참조 설명서를 참조하세요.

주제

- Aurora MySQL 대기 이벤트

- [Aurora MySQL 스레드 상태]
- [Aurora MySQL 메모리]
- [Aurora MySQL 프로세스]

Aurora MySQL 대기 이벤트

대기 이벤트는 세션이 대기 중인 리소스를 나타냅니다. 예를 들어 대기 이벤트 `io/socket/sql/client_connection`은 스레드가 새 연결을 처리하고 있음을 나타냅니다. 세션이 기다리는 일반적인 리소스는 다음과 같습니다.

- 버퍼에 대한 단일 스레드 액세스(예: 세션이 버퍼 수정을 시도하는 경우)
- 현재 다른 세션에 의해 잠겨 있는 행
- 데이터 파일 읽기
- 로그 파일 쓰기

예를 들어 쿼리를 충족하기 위해 세션에서 전체 테이블 스캔을 수행할 수 있습니다. 데이터가 아직 메모리에 없는 경우 세션은 디스크 I/O가 완료될 때까지 기다립니다. 버퍼를 메모리로 읽으면 다른 세션이 동일한 버퍼에 액세스하기 때문에 세션을 기다려야 할 수 있습니다. 데이터베이스는 미리 정의된 대기 이벤트를 사용하여 대기를 기록합니다. 이러한 이벤트는 카테고리로 그룹화됩니다.

대기 이벤트 자체가 성능 문제를 나타내지는 않습니다. 예를 들어 요청된 데이터가 메모리에 없으면 디스크에서 데이터를 읽어야 합니다. 한 세션이 업데이트를 위해 행을 잠그면 다른 세션은 해당 행의 잠금이 해제될 때까지 기다려 업데이트할 수 있습니다. 커밋을 수행하려면 로그 파일에 대한 쓰기가 완료될 때까지 대기해야 합니다. 대기는 데이터베이스의 정상적인 기능에 필수적입니다.

많은 수의 대기 이벤트는 일반적으로 성능 문제를 나타냅니다. 이러한 경우 대기 이벤트 데이터를 사용하여 세션이 시간을 소비하는 위치를 결정할 수 있습니다. 예를 들어 일반적으로 몇 분 안에 실행되는 보고서가 몇 시간 동안 실행되는 경우 총 대기 시간에 가장 많이 기여하는 대기 이벤트를 식별할 수 있습니다. 최상위 대기 이벤트의 원인을 확인할 수 있는 경우 성능을 향상시키는 변경 작업을 수행할 수 있습니다. 예를 들어 세션이 다른 세션에 의해 잠긴 행에서 대기 중인 경우 잠금 세션을 종료할 수 있습니다.

Aurora MySQL 스레드 상태

일반 스레드 상태는 일반 쿼리 처리와 연결된 `State` 값입니다. 예를 들어 `sending data` 스레드 상태는 스레드가 올바른 결과 집합을 결정하기 위해 쿼리의 행을 읽고 필터링하고 있음을 나타냅니다.

스레드 상태를 사용하여 대기 이벤트를 사용하는 방법과 유사한 방식으로 Aurora MySQL 튜닝할 수 있습니다. 예를 들어 `sending data`의 빈번한 발생은 일반적으로 쿼리가 인덱스를 사용하고 있지 않음을 나타냅니다. 스레드 상태에 대한 자세한 내용은 MySQL 참조 설명서에서 [일반 스레드 상태]를 참조하세요.

성능 개선 도우미를 사용하는 경우 다음 조건 중 하나에 해당됩니다.

- 성능 스키마가 켜져 있음 - Aurora MySQL이 스레드 상태가 아닌 대기 이벤트를 표시합니다.
- 성능 스키마가 꺼져 있음 - Aurora MySQL이 스레드 상태를 표시합니다.

자동 관리를 위해 성능 스키마를 구성하는 것이 좋습니다. 성능 스키마는 잠재적인 성능 문제를 조사하기 위한 추가적인 인사이트 및 더 나은 도구를 제공합니다. 자세한 내용은 [Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요] 단원을 참조하십시오.

Aurora MySQL 메모리

Aurora MySQL에서 가장 중요한 메모리 영역은 버퍼 풀과 로그 버퍼입니다.

주제

- [버퍼 풀]

버퍼 풀

버퍼 풀은 Aurora MySQL이 테이블 및 인덱스 데이터를 캐시하는 공유 메모리 영역입니다. 쿼리는 디스크에서 읽지 않고도 메모리에서 자주 사용하는 데이터에 직접 액세스할 수 있습니다.

버퍼 풀은 연결된 페이지 목록으로 구성됩니다. 페이지에는 여러 개의 행이 있을 수 있습니다. Aurora MySQL은 최소최근사용(LRU) 알고리즘을 사용하여 풀에서 페이지를 에이징합니다.

자세한 내용은 MySQL 참조 설명서의 [Buffer Pool](버퍼 풀)을 참조하세요.

Aurora MySQL 프로세스

Aurora MySQL은 Aurora PostgreSQL과는 매우 다른 프로세스 모델을 사용합니다.

주제

- [MySQL 서버(mysqld)]
- [스레드]
- [스레드 풀]

MySQL 서버(mysqld)

MySQL 서버는 mysqld라는 단일 운영 체제 프로세스입니다. MySQL 서버는 추가 프로세스를 생성하지 않습니다. 따라서 Aurora MySQL 데이터베이스는 mysqld를 사용하여 대부분의 작업을 수행합니다.

MySQL 서버가 시작되면 MySQL 클라이언트에서 네트워크 연결을 수신합니다. 클라이언트가 데이터베이스에 연결되면 mysqld가 스레드를 엽니다.

스레드

연결 관리자 스레드는 각 클라이언트 연결을 전용 스레드와 연결합니다. 이 스레드는 인증을 관리하고 명령문을 실행하며 결과를 클라이언트로 반환합니다. 필요한 경우 연결 관리자가 새 스레드를 만듭니다.

스레드 캐시는 사용 가능한 스레드의 집합입니다. 연결이 해제되는 경우 캐시가 꽉 차지 않으면 MySQL은 스레드를 스레드 캐시로 반환합니다. `thread_cache_size` 시스템 변수를 사용하여 스레드 캐시 크기를 결정합니다.

스레드 풀

스레드 풀은 여러 스레드 그룹으로 구성됩니다. 각 그룹은 클라이언트 연결 집합을 관리합니다. 클라이언트가 데이터베이스에 연결되면 스레드 풀은 라운드 로빈 방식으로 스레드 그룹에 연결을 할당합니다. 스레드 풀은 연결과 스레드를 분리합니다. 연결 및 해당 연결에서 받은 명령문을 실행하는 스레드 간에는 관계가 고정되지 않습니다.

대기 이벤트로 Aurora MySQL 튜닝

다음 표에는 일반적으로 성능 문제를 나타내는 Aurora MySQL 대기 이벤트가 요약되어 있습니다. 다음 대기 이벤트는 Aurora MySQL 대기 이벤트 목록의 하위 집합입니다.

대기 이벤트	설명
cpu	이 이벤트는 스레드가 CPU에서 활성 상태이거나 CPU에 대해 대기 중일 때 발생합니다.
io/aurora_redo_log_flush	이 이벤트는 세션이 Aurora 스토리지에 영구 데이터를 쓸 때 발생합니다.
io/aurora_respond_to_client	이 이벤트는 스레드가 결과 집합을 클라이언트에 반환하기 위해 대기 중일 때 발생합니다.

대기 이벤트	설명
io/redo_log_flush	이 이벤트는 세션이 Aurora 스토리지에 영구 데이터를 쓸 때 발생합니다.
io/socket/sql/client_connection	이 이벤트는 스레드가 새 연결을 처리하는 과정에 있을 때 발생합니다.
io/table/sql/handler	이 이벤트는 작업이 스토리지 엔진에 위임된 경우에 발생합니다.
synch/cond/innodb/row_lock_wait	이 이벤트는 한 세션이 업데이트에 대해 행을 잠그고 다른 세션에서 동일한 행을 업데이트하려고 하는 경우 발생합니다.
synch/cond/innodb/row_lock_wait_cond	이 이벤트는 한 세션이 업데이트에 대해 행을 잠그고 다른 세션에서 동일한 행을 업데이트하려고 하는 경우 발생합니다.
synch/cond/sql/MDL_context::COND_wait_status	이 이벤트는 테이블 메타데이터 잠금에 대기 중인 스레드가 있는 경우 발생합니다.
synch/mutex/innodb/aurora_lock_thread_slot_futex	이 이벤트는 한 세션이 업데이트에 대해 행을 잠그고 다른 세션에서 동일한 행을 업데이트하려고 하는 경우 발생합니다.
synch/mutex/innodb/buf_pool_mutex	이 이벤트는 스레드가 메모리의 페이지에 액세스하기 위해 InnoDB 버퍼 풀에서 잠긴 경우 발생합니다.
synch/mutex/innodb/fil_system_mutex	이 이벤트는 세션이 테이블스페이스 메모리 캐시에 액세스하기 위해 대기 중일 때 발생합니다.
synch/mutex/innodb/trx_sys_mutex	이 이벤트는 트랜잭션 수가 많은 데이터베이스 작업이 많을 때 발생합니다.
synch/sxlock/innodb/hash_table_locks	이 이벤트는 버퍼 풀에 없는 페이지를 파일에서 읽어야 할 때 발생합니다.

cpu

cpu 대기 이벤트는 스레드가 CPU에서 활성 상태이거나 CPU에 대해 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

모든 vCPU의 경우 연결은 이 CPU에서 작업을 실행할 수 있습니다. 경우에 따라 실행할 준비가 된 활성 연결 수가 vCPU 수보다 많을 수 있습니다. 이러한 불균형으로 인해 CPU 리소스에 대해 대기 중인 연결이 발생합니다. 활성 연결 수가 vCPU 수보다 일관되게 높게 유지되면 해당 인스턴스에 CPU 경합이 발생합니다. 경합으로 인해 cpu 대기 이벤트가 발생합니다.

> **Note**
>
> CPU의 성능 개선 도우미 지표는 `DBLoadCPU`입니다. `DBLoadCPU`의 값은 CloudWatch 지표 `CPUUtilization`의 값과 다를 수 있습니다. 후자의 지표는 데이터베이스 인스턴스에 대한 HyperVisor에서 수집됩니다.

성능 개선 도우미 OS 지표는 CPU 사용률에 대한 자세한 정보를 제공합니다. 예를 들어 다음 지표가 표시될 수 있습니다.

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`

- `os.cpuUtilization.idle.avg`

성능 개선 도우미는 데이터베이스 엔진의 CPU 사용량을 `os.cpuUtilization.nice.avg`와 같이 보고합니다.

대기 증가의 가능한 원인

이 이벤트가 정상 상태보다 많이 발생하여 성능 문제를 나타낼 수 있는 경우 일반적인 원인은 다음과 같습니다.

- 분석 쿼리
- 많은 동시 트랜잭션 수
- 장기 실행 트랜잭션
- 연결 수의 급격한 증가를 로그인 스톰(login storm)이라고 함
- 컨텍스트 전환 증가

작업

cpu 대기 이벤트가 데이터베이스 활동을 지배하는 경우 반드시 성능 문제를 나타내지는 않습니다. 성능이 저하되는 경우만 이 이벤트에 응답하세요.

CPU 사용률 증가의 원인에 따라 다음 전략을 고려하세요.

- 호스트의 CPU 용량을 늘리세요. 이 접근법은 일반적으로 일시적인 구제만 해당합니다.
- 잠재적인 최적화를 위해 상위 쿼리를 식별하세요.
- 해당하는 경우 일부 읽기 전용 워크로드를 리더 노드로 리디렉션하세요.

주제

- 문제를 일으키는 세션 또는 쿼리 식별
- 높은 CPU 워크로드 분석 및 최적화

문제를 일으키는 세션 또는 쿼리 식별

세션과 쿼리를 찾으려면 CPU 로드가 가장 높은 SQL 문에 대한 성능 개선 도우미의 상위 SQL(Top SQL) 테이블을 참조하세요. 자세한 내용은 성능 개선 도우미 대시보드를 사용한 지표 분석 단원을 참조하십시오.

일반적으로 하나 또는 두 개의 SQL 문은 대부분의 CPU 사이클을 사용합니다. 이러한 명령문에 집중하세요. DB 인스턴스에 평균 활성 세션(AAS)이 3.1인 DB 로드가 있는 vCPU 2개가 있고 모두 CPU 상태에 있다고 가정합니다. 이 경우 인스턴스는 CPU 바인딩입니다. 다음과 같은 전략을 생각해 보세요.

- vCPU가 더 많은 대규모 인스턴스 클래스로 업그레이드하세요.
- CPU 부하가 낮도록 쿼리를 튜닝하세요.

이 예에서는 최상위 SQL 쿼리에 1.5 AAS의 DB 로드가 있고 모두 CPU 상태입니다. 다른 SQL 문은 CPU 상태의 로드가 0.1입니다. 이 예에서는 로드가 가장 낮은 SQL 문을 중지한 경우 데이터베이스 로드가 크게 줄어들지 않습니다. 그러나 두 개의 로드가 높은 쿼리를 효율적으로 두 배로 최적화하면 CPU 병목 현상을 없앨 수 있습니다. 1.5 AAS의 CPU 로드를 50% 줄이면 각 명령문에 대한 AAS가 0.75로 감소합니다. CPU에 사용된 총 DB 로드는 이제 1.6 AAS입니다. 이 값은 최대 vCPU 라인인 2.0 미만입니다.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, [성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석](#)을 참조하세요. 또한 AWS 지원 문서, [Amazon RDS for MySQL 인스턴스에서 높은 CPU 사용률 문제를 해결하려면 어떻게 해야 하나요?](#)를 참조하세요.

높은 CPU 워크로드 분석 및 최적화

CPU 사용량을 늘리는 하나 이상의 쿼리를 식별한 후 이를 최적화하거나 연결을 종료할 수 있습니다. 다음 예에서는 연결을 종료하는 방법을 보여줍니다.

```
CALL mysql.rds_kill(processID);
```

자세한 내용은 mysql.rds_kill 단원을 참조하십시오.

세션을 종료하면 해당 작업은 긴 롤백을 트리거할 수 있습니다.

쿼리 최적화 지침 준수

쿼리를 최적화하려면 다음 지침을 고려하세요.

- `EXPLAIN` 문을 실행하세요.

  이 명령은 쿼리 실행과 관련된 개별 단계를 보여줍니다. 자세한 내용은 MySQL 설명서의 EXPLAIN으로 쿼리 최적화를 참조하세요.

- `SHOW PROFILE` 문을 실행하세요.

이 명령문을 사용하여 현재 세션 도중에 실행되는 명령문의 리소스 사용량을 나타내는 프로필 세부 정보를 검토할 수 있습니다. 자세한 내용은 MySQL 설명서의 REPAIR PROFILE 문을 참조하세요.

- ANALYZE TABLE 문을 실행하세요.

  이 명령문을 사용하여 CPU 사용량이 높은 쿼리에서 액세스한 테이블의 인덱스 통계를 새로 고칠 수 있습니다. 명령문을 분석하면 최적화 프로그램이 적절한 실행 계획을 선택할 수 있도록 도울 수 있습니다. 자세한 내용은 MySQL 설명서의 ANALYZE TABLE 문을 참조하세요.

CPU 사용량 향상을 위한 지침 준수

데이터베이스 인스턴스의 CPU 사용을 개선하려면 다음 지침을 따르세요.

- 모든 쿼리가 적절한 인덱스를 사용하고 있는지 확인합니다.
- Aurora 병렬 쿼리를 사용할 수 있는지 확인합니다. 이 기술을 사용하여 기능 처리, 행 필터링 및 WHERE 절의 열 프로젝션을 낮추어 헤드 노드에서 CPU 사용량을 줄일 수 있습니다.
- 초당 SQL 실행 수가 예상 임계값을 충족하는지 확인합니다.
- 인덱스 유지 관리 또는 새 인덱스 생성이 프로덕션 워크로드에 필요한 CPU 사이클을 차지하는지 확인합니다. 피크 활동 시간 이외의 시간에 유지 관리 활동을 예약합니다.
- 파티셔닝을 사용하여 쿼리 데이터 세트를 줄일 수 있는지 확인합니다. 자세한 내용은 블로그 게시물, 통합 워크로드를 위해 MySQL과 호환되는 Amazon Aurora를 계획하고 최적화하는 방법을 참조하세요.

연결 스톰 확인

DBLoadCPU 지표는 그다지 높지 않지만 CPUUtilization 지표가 높은 경우 CPU 사용률이 높은 원인은 데이터베이스 엔진 외부에 있습니다. 전형적인 예는 연결 스톰입니다.

다음 조건이 true인지 확인합니다.

- 두 성능 개선 도우미, CPUUtilization 지표 및 Amazon CloudWatch DatabaseConnections 지표 모두에서 증가했습니다.
- CPU의 스레드 수가 vCPU 수보다 큽니다.

위의 조건이 true이면 데이터베이스 연결 수를 줄이는 것이 좋습니다. 예를 들어 RDS Proxy와 같은 연결 풀을 사용할 수 있습니다. 효과적인 연결 관리 및 크기 조정에 대한 모범 사례를 알아보려면 연결 관리를 위한 Amazon Aurora MySQL DBA 핸드북 백서를 참조하세요.

io/aurora_redo_log_flush

`io/aurora_redo_log_flush` 이벤트는 세션이 Amazon Aurora 스토리지에 영구 데이터를 쓸 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2

컨텍스트

`io/aurora_redo_log_flush` 이벤트는 Aurora MySQL에서 쓰기 입력/출력(I/O) 작업을 위한 것입니다.

> **Note**
>
> Aurora MySQL 버전 3에서는 이 대기 이벤트의 이름이 io/redo_log_flush입니다.

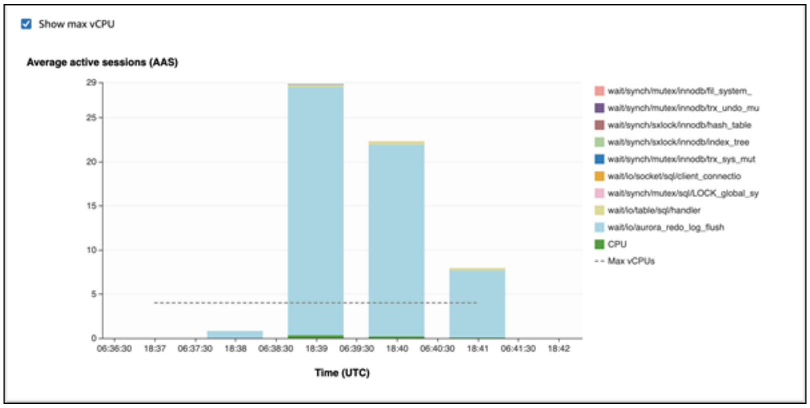
대기 증가의 가능한 원인

데이터 지속성을 위해 커밋에는 안정적인 스토리지에 대한 내구성 있는 쓰기가 필요합니다. 데이터베이스가 너무 많은 커밋을 수행하는 경우 쓰기 I/O 작업에 대기 이벤트인 `io/aurora_redo_log_flush` 대기 이벤트가 있습니다.

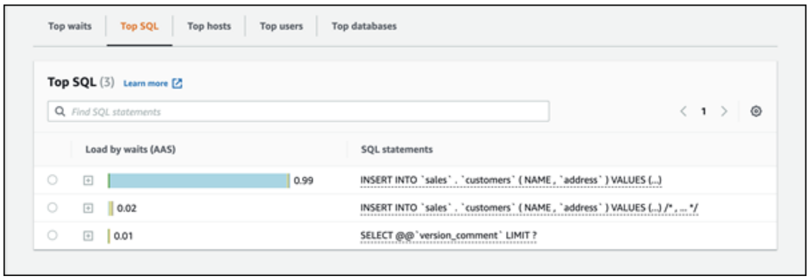
다음 예에서는 db.r5.xlarge DB 인스턴스 클래스를 사용하여 50,000개의 레코드를 Aurora MySQL DB 클러스터에 삽입합니다.

- 첫 번째 예에서 각 세션은 행별로 10,000개의 레코드를 삽입합니다. 기본적으로 데이터 조작 언어(DML) 명령이 트랜잭션 내에 있지 않으면 Aurora MySQL은 암시적 커밋을 사용합니다. 자동 커밋

이 활성화됩니다. 즉, 각 행 삽입마다 커밋이 있습니다. 성능 개선 도우미는 연결이 대부분의 시간을 `io/aurora_redo_log_flush` 대기 이벤트에서 기다리면서 소비한다는 것을 보여줍니다.

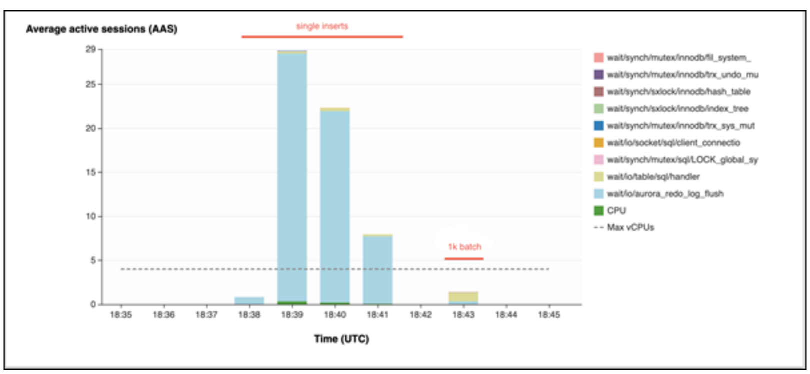


이 이벤트는 사용되는 간단한 삽입 문 때문에 발생합니다.



50,000개의 레코드를 삽입하는 데 3.5분이 걸립니다.

- 두 번째 예에서는 삽입이 1,000개의 배치로 만들어집니다. 즉, 각 연결은 10,000개가 아닌 10개의 커밋을 수행합니다. 성능 개선 도우미에 따르면 연결은 대부분의 시간을 `io/aurora_redo_log_flush` 대기 이벤트에서 소비하지 않습니다.



50,000개의 레코드를 삽입하는 데 4초가 걸립니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

문제가 있는 세션 및 쿼리 식별

DB 인스턴스에 병목 현상이 발생하는 경우 첫 번째 작업은 이를 유발하는 세션과 쿼리를 찾는 것입니다. 유용한 AWS 데이터베이스 블로그 게시물은 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

병목 현상을 일으키는 세션 및 쿼리를 식별하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 데이터베이스 로드(Database load)에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   목록 맨 위에 있는 쿼리로 인해 데이터베이스에 대한 로드가 가장 높습니다.

쓰기 작업 그룹화

다음 예에서는 `io/aurora_redo_log_flush` 대기 이벤트를 트리거합니다. (자동 커밋이 활성화됩니다.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
```

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

`io/aurora_redo_log_flush` 대기 이벤트에서 대기하는 시간을 줄이려면 쓰기 작업을 논리적으로 단일 커밋으로 그룹화하여 스토리지에 대한 지속적인 호출을 줄입니다.

자동 커밋 해제

다음 예와 같이 트랜잭션 내에 없는 대규모 변경을 수행하기 전에 자동 커밋을 해제합니다.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

트랜잭션 사용

다음 예와 같이 트랜잭션을 사용할 수 있습니다.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
```

```
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

배치 사용

다음 예와 같이 배치를 변경할 수 있습니다. 그러나 너무 큰 배치를 사용하면 특히 읽기 전용 복제본이나 특정 시점으로 복구(PITR)를 수행할 때 성능 문제가 발생할 수 있습니다.

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

`io/aurora_respond_to_client` 이벤트는 스레드가 결과 세트를 클라이언트에 반환하기 위해 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2

컨텍스트

`io/aurora_respond_to_client` 이벤트는 스레드가 결과 세트를 클라이언트에 반환하기 위해 대기 중임을 나타냅니다.

쿼리 처리가 완료되고 결과가 애플리케이션 클라이언트로 다시 반환되는 중입니다. 그러나 DB 클러스터에 네트워크 대역폭이 부족하기 때문에 스레드가 결과 집합을 반환하기 위해 대기 중입니다.

대기 증가의 가능한 원인

`io/aurora_respond_to_client` 이벤트가 정상보다 많이 나타나 성능 문제를 나타내는 경우 일반적인 원인은 다음과 같습니다.

워크로드에 대해 부족한 DB 인스턴스 클래스

DB 클러스터에서 사용하는 DB 인스턴스 클래스에는 워크로드를 효율적으로 처리하는 데 필요한 네트워크 대역폭이 없습니다.

대규모 결과 세트

쿼리가 더 많은 수의 행을 반환하기 때문에 반환되는 결과 집합의 크기가 증가했습니다. 결과 집합이 클수록 네트워크 대역폭을 더 많이 소모합니다.

클라이언트에 대한 로드 증가

클라이언트에 CPU 압력, 메모리 압력 또는 네트워크 포화가 있을 수 있습니다. 클라이언트에 대한 로드가 증가하면 Aurora MySQL DB 클러스터에서 데이터 수신이 지연됩니다.

네트워크 대기 시간 증가

Aurora MySQL DB 클러스터와 클라이언트 간에 네트워크 대기 시간이 늘어날 수 있습니다. 네트워크 대기 시간이 높을수록 클라이언트가 데이터를 수신하는 데 필요한 시간이 늘어납니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 이벤트를 일으키는 세션 및 쿼리 식별
- DB 인스턴스 클래스 크기 조정
- 예기치 않은 결과가 있는지 워크로드 확인

- 리더 인스턴스를 통해 워크로드 배포
- SQL_BUFFER_RESULT 한정자 사용

이벤트를 일으키는 세션 및 쿼리 식별

성능 개선 도우미를 사용하여 `io/aurora_respond_to_client` 대기 이벤트에 의해 차단된 쿼리를 표시할 수 있습니다. 일반적으로 보통에서 중요한 로드가 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을 소비하는 위치를 확인합니다. 가장 높은 로드를 일으키는 대기 이벤트를 살펴보고 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

높은 로드에 책임이 있는 SQL 쿼리를 찾으려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다. DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결에 대한 유용한 개요는 AWS 데이터베이스 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

DB 인스턴스 클래스 크기 조정

`NetworkReceiveThroughput` 및 `NetworkTransmitThroughput`과 같은 네트워크 처리량과 관련된 Amazon CloudWatch 지표 값의 증가가 있는지 확인합니다. DB 인스턴스 클래스 네트워크 대역폭에 도달하면 DB 클러스터를 수정하여 DB 클러스터가 사용하는 DB 인스턴스 클래스를 크기 조정할 수 있습니다. 네트워크 대역폭이 큰 DB 인스턴스 클래스는 클라이언트에 데이터를 보다 효율적으로 반환합니다.

Amazon CloudWatch 지표 모니터링에 대한 자세한 내용은 Amazon RDS 콘솔에서 지표 보기 섹션을 참조하세요. DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요. DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

예기치 않은 결과가 있는지 워크로드 확인

DB 클러스터의 워크로드를 확인하고 예상치 못한 결과가 나오지 않는지 확인합니다. 예를 들어 예상보다 많은 수의 행을 반환하는 쿼리가 있을 수 있습니다. 이 경우 Innodb_rows_read와 같은 성능 개선 도우미 카운터 지표를 사용할 수 있습니다. 자세한 내용은 성능 개선 도우미 카운터 단원을 참조하십시오.

리더 인스턴스를 통해 워크로드 배포

Aurora 복제본을 통해 읽기 전용 워크로드를 배포할 수 있습니다. Aurora 복제본을 더 많이 추가하여 수평으로 확장할 수 있습니다. 이렇게 하면 네트워크 대역폭에 대한 제한이 증가할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 단원을 참조하십시오.

SQL_BUFFER_RESULT 한정자 사용

명령문이 클라이언트에 반환되기 전에 결과를 임시 테이블로 강제하는 SELECT 문에 SQL_BUFFER_RESULT 한정자를 추가할 수 있습니다. 이 한정자는 쿼리가 io/aurora_respond_to_client 대기 상태에 있어 InnoDB 잠금이 해제되지 않는 경우 성능 문제에 도움이 될 수 있습니다. 자세한 내용은 MySQL 설명서의 SELECT 문을 참조하세요.

io/redo_log_flush

io/redo_log_flush 이벤트는 세션이 Amazon Aurora 스토리지에 영구 데이터를 쓸 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 3

컨텍스트

io/redo_log_flush 이벤트는 Aurora MySQL에서 쓰기 입력/출력(I/O) 작업을 위한 것입니다.

> **Note**
>
> Aurora MySQL 버전 2에서는 이 대기 이벤트의 이름이 io/aurora_redo_log_flush입니다.

대기 증가의 가능한 원인

데이터 지속성을 위해 커밋에는 안정적인 스토리지에 대한 내구성 있는 쓰기가 필요합니다. 데이터베이스가 너무 많은 커밋을 수행하는 경우 쓰기 I/O 작업에 대기 이벤트인 `io/redo_log_flush` 대기 이벤트가 있습니다.

이 대기 이벤트의 동작 예제는 io/aurora_redo_log_flush 섹션을 참조하세요.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 문제가 있는 세션 및 쿼리 식별
- 쓰기 작업 그룹화
- 자동 커밋 해제
- 트랜잭션 사용
- 배치 사용

문제가 있는 세션 및 쿼리 식별

DB 인스턴스에 병목 현상이 발생하는 경우 첫 번째 작업은 이를 유발하는 세션과 쿼리를 찾는 것입니다. 유용한 AWS 데이터베이스 블로그 게시물은 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

병목 현상을 일으키는 세션 및 쿼리를 식별하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 데이터베이스 로드(Database load)에서 대기별 조각(Slice by wait)을 선택합니다.

5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   목록 맨 위에 있는 쿼리로 인해 데이터베이스에 대한 로드가 가장 높습니다.

쓰기 작업 그룹화

다음 예에서는 `io/redo_log_flush` 대기 이벤트를 트리거합니다. (자동 커밋이 활성화됩니다.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

`io/redo_log_flush` 대기 이벤트에서 대기하는 시간을 줄이려면 쓰기 작업을 논리적으로 단일 커밋으로 그룹화하여 스토리지에 대한 지속적인 호출을 줄입니다.

자동 커밋 해제

다음 예와 같이 트랜잭션 내에 없는 대규모 변경을 수행하기 전에 자동 커밋을 해제합니다.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;
```

```
SET SESSION AUTOCOMMIT=ON;
```

트랜잭션 사용

다음 예와 같이 트랜잭션을 사용할 수 있습니다.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

배치 사용

다음 예와 같이 배치를 변경할 수 있습니다. 그러나 너무 큰 배치를 사용하면 특히 읽기 전용 복제본이나 특정 시점으로 복구(PITR)를 수행할 때 성능 문제가 발생할 수 있습니다.

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

`io/socket/sql/client_connection` 이벤트는 스레드가 새 연결을 처리하고 있을 때 발생합니다.

주제

- [지원되는 엔진 버전](#)

- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

`io/socket/sql/client_connection` 이벤트는 mysqld가 들어오는 새 클라이언트 연결을 처리하기 위해 스레드를 만드는 중임을 나타냅니다. 이 시나리오에서는 연결이 스레드가 할당될 때까지 기다리는 동안 새 클라이언트 연결 요청 서비스 처리 속도가 느려집니다. 자세한 내용은 MySQL 서버 (mysqld) 섹션을 참조하세요.

대기 증가의 가능한 원인

이 이벤트가 정상보다 많이 발생해 성능 문제를 일으킬 수 있는 경우 일반적인 원인은 다음과 같습니다.

- 애플리케이션에서 Amazon RDS 인스턴스로 새 사용자 연결이 급증했습니다.
- 네트워크, CPU 또는 메모리가 제한되기 때문에 DB 인스턴스에서 새 연결을 처리할 수 없습니다.

작업

`io/socket/sql/client_connection`이 데이터베이스 활동을 지배한다고 해서 반드시 성능 문제를 나타내는 것은 아닙니다. 유휴 상태가 아닌 데이터베이스에서는 대기 이벤트가 항상 맨 위에 있습니다. 성능이 저하될 때만 작동합니다. 대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 문제가 있는 세션 및 쿼리 식별
- 연결 관리를 위한 모범 사례 준수
- 리소스가 제한되는 경우 인스턴스 확장
- 상위 호스트 및 상위 사용자 확인
- performance _schema 테이블 쿼리

- 쿼리의 스레드 상태 확인
- 요청 및 쿼리 감사
- 데이터베이스 연결 풀링

문제가 있는 세션 및 쿼리 식별

DB 인스턴스에 병목 현상이 발생하는 경우 첫 번째 작업은 이를 유발하는 세션과 쿼리를 찾는 것입니다. 유용한 블로그 게시물은 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

병목 현상을 일으키는 세션 및 쿼리를 식별하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다.
4. 데이터베이스 로드(Database load)에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   목록 맨 위에 있는 쿼리로 인해 데이터베이스에 대한 로드가 가장 높습니다.

연결 관리를 위한 모범 사례 준수

연결을 관리하려면 다음 전략을 고려하세요.

- 연결 풀링 사용

  필요에 따라 연결 수를 점진적으로 늘릴 수 있습니다. 자세한 내용은 Amazon Aurora MySQL 데이터베이스 관리자 핸드북 백서를 참조하세요.
- 리더 노드를 사용하여 읽기 전용 트래픽을 재배포합니다.

  자세한 내용은 Aurora 복제본 및 Amazon Aurora 엔드포인트 연결를 참조하세요.

리소스가 제한되는 경우 인스턴스 확장

다음 리소스에서 제한의 예를 찾아보세요.

- CPU

CPU 사용량이 높은지 Amazon CloudWatch 지표를 확인하세요.

- Network

  CloudWatch 지표, `network receive throughput` 및 `network transmit throughput`의 값이 증가했는지 확인합니다. 인스턴스가 인스턴스 클래스의 네트워크 대역폭 제한에 도달한 경우 RDS 인스턴스를 더 높은 인스턴스 클래스 유형으로 확장하는 것이 좋습니다. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.
- 여유 메모리

  CloudWatch 지표 `FreeableMemory`가 하락했는지 확인합니다. 또한 향상된 모니터링을 켜는 것이 좋습니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 단원을 참조하십시오.

상위 호스트 및 상위 사용자 확인

성능 개선 도우미를 사용하여 상위 호스트와 상위 사용자를 확인합니다. 자세한 내용은 성능 개선 도우미 대시보드를 사용한 지표 분석 단원을 참조하십시오.

performance _schema 테이블 쿼리

현재 및 총 연결의 정확한 개수를 얻으려면 `performance_schema` 테이블을 쿼리합니다. 이 기술을 사용하면 많은 수의 연결을 생성하는 소스 사용자 또는 호스트를 식별할 수 있습니다. 예를 들어 다음과 같이 `performance_schema` 테이블을 쿼리합니다.

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

쿼리의 스레드 상태 확인

성능 문제가 진행 중인 경우 쿼리의 스레드 상태를 확인합니다. `mysql` 클라이언트에서 다음 명령을 실행합니다.

```
show processlist;
```

요청 및 쿼리 감사

사용자 계정의 요청 및 쿼리의 특성을 확인하려면 AuroraAurora MySQL 고급 감사를 사용합니다. 감사를 활성화하는 방법은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 섹션을 참조하세요.

데이터베이스 연결 풀링

연결 관리를 위해 Amazon RDS Proxy를 사용하는 것이 좋습니다. RDS Proxy를 사용하면 애플리케이션이 데이터베이스 연결을 풀링하고 공유하도록 허용하여 확장 기능을 향상할 수 있습니다. RDS 프록시는 애플리케이션 연결을 유지하면서 예비 DB 인스턴스에 자동으로 연결하여 데이터베이스 장애에 대한 애플리케이션의 복원력을 높입니다. 자세한 내용은 [Amazon RDS Proxy for Aurora](#) 단원을 참조하십시오.

io/table/sql/handler

`io/table/sql/handler` 이벤트는 작업을 스토리지 엔진에 작업이 위임했을 때 발생합니다.

주제

- [지원되는 엔진 버전](#)
- [컨텍스트](#)
- [대기 증가의 가능한 원인](#)
- [작업](#)

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

`io/table` 이벤트는 테이블에 대한 액세스를 기다리고 있음을 나타냅니다. 이 이벤트는 데이터가 버퍼 풀에 캐시되는지 디스크에서 액세스되는지 관계없이 발생합니다. `io/table/sql/handler` 이벤트는 워크로드 활동의 증가를 나타냅니다.

처리기(handler)는 특정 유형의 데이터를 전문화하거나 특정 특수 작업에 중점을 둔 루틴입니다. 예를 들어 이벤트 처리기는 운영 체제 또는 사용자 인터페이스에서 이벤트와 신호를 수신하고 다이제스트합니다. 메모리 처리기는 메모리와 관련된 작업을 수행합니다. 파일 입력 처리기는 컨텍스트에 따라 파일 입력을 수신하고 데이터에 대한 특수 작업을 수행하는 기능입니다.

`performance_schema.events_waits_current`와 같은 보기는 실제 대기가 잠금과 같은 중첩된 대기 이벤트인 경우 `io/table/sql/handler`를 보여주는 경우가 많습니다. 실제 대기 시간이 `io/table/sql/handler`가 아닌 경우 성능 개선 도우미는 중첩된 대기 이벤트를 보고합니다. 성능 개선 도우미가 `io/table/sql/handler`를 보고하는 경우는 숨겨진 중첩 대기 이벤트가 아니라 I/O 요청

의 InnoDB 처리를 나타냅니다. 자세한 내용은 MySQL 참조 설명서에서 성능 스키마 원자 및 분자 이벤트를 참조하세요.

`io/table/sql/handler` 이벤트는 `io/aurora_redo_log_flush`와 같은 I/O 대기와 함께 상위 대기 이벤트에 나타나는 경우가 많습니다.

대기 증가의 가능한 원인

성능 개선 도우미에서 `io/table/sql/handler` 이벤트의 급격한 스파이크는 워크로드 활동의 증가를 나타냅니다. 활동 증가는 I/O 증가를 의미합니다.

성능 개선 도우미는 중첩 이벤트 ID를 필터링하고 기본 중첩 이벤트가 잠금 대기인 경우 `io/table/sql/handler` 대기를 보고하지 않습니다. 예를 들어 근본 원인 이벤트가 synch/mutex/innodb/aurora_lock_thread_slot_futex인 경우 성능 개선 도우미는 이 대기를 `io/table/sql/handler`가 아닌 상위 대기 이벤트에 표시합니다.

`performance_schema.events_waits_current`와 같은 보기에서 `io/table/sql/handler`의 대기는 실제 대기가 잠금과 같은 중첩된 대기 이벤트인 경우 나타나는 경우가 많습니다. 실제 대기가 `io/table/sql/handler`와 다른 경우 성능 개선 도우미는 중첩된 대기를 조회하고 `io/table/sql/handler`가 아닌 실제 대기를 보고합니다. 성능 개선 도우미가 `io/table/sql/handler`를 보고하는 경우 실제 대기는 숨겨진 중첩 대기 이벤트가 아닌 `io/table/sql/handler`입니다. 자세한 내용은 MySQL 5.7 참조 설명서에서 성능 스키마 원자 및 분자 이벤트를 참조하세요.

작업

이 대기 이벤트가 데이터베이스 활동을 지배하는 경우 반드시 성능 문제를 나타내는 것은 아닙니다. 데이터베이스가 활성 상태일 때 대기 이벤트는 항상 맨 위에 있습니다. 성능이 저하될 때만 조치해야 합니다.

표시되는 다른 대기 이벤트에 따라 다른 작업을 권장합니다.

주제

- 이벤트를 일으키는 세션 및 쿼리 식별
- 성능 개선 도우미 카운터 지표를 통해 상관관계 확인
- 다른 상호 연관된 대기 이벤트 확인

이벤트를 일으키는 세션 및 쿼리 식별

일반적으로 보통 로드에서 중요 로드까지 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을

소비하는 위치를 검사합니다. 가장 높은 로드를 일으키는 대기 이벤트를 살펴보고 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

높은 로드에 책임이 있는 SQL 쿼리를 찾으려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다. DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

성능 개선 도우미 카운터 지표를 통해 상관관계 확인

`Innodb_rows_changed`와 같은 성능 개선 도우미 카운터 지표를 확인합니다. 카운터 지표가 `io/table/sql/handler`와 상호 연관되어 있는 경우 다음 단계를 따르세요.

1. 성능 개선 도우미에서 `io/table/sql/handler` 상위 대기 이벤트에 해당하는 SQL 문을 찾습니다. 가능한 경우 더 적은 수의 행을 반환하도록 이 명령문을 최적화합니다.
2. `schema_table_statistics` 및 `x$schema_table_statistics` 보기에서 상위 테이블을 검색합니다. 이러한 보기는 테이블당 소요된 시간을 보여줍니다. 자세한 내용은 MySQL 참조 설명서에서 schema_table_statistics 및 x$schema_table_statistics 보기를 참조하세요.

   기본적으로 행은 총 대기 시간의 내림차순으로 정렬됩니다. 경합이 가장 많은 테이블이 먼저 표시됩니다. 출력은 읽기, 쓰기, 가져오기, 삽입, 업데이트 또는 삭제에 소요되는 시간 여부를 나타냅니다.

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
```

```
     fetch_latency: 39.14 m
     rows_inserted: 14833
    insert_latency: 5.78 m
      rows_updated: 30470
    update_latency: 5.39 m
      rows_deleted: 14833
    delete_latency: 3.81 m
  io_read_requests: NULL
           io_read: NULL
   io_read_latency: NULL
 io_write_requests: NULL
          io_write: NULL
  io_write_latency: NULL
  io_misc_requests: NULL
   io_misc_latency: NULL
1 row in set (0.11 sec)
```

다른 상호 연관된 대기 이벤트 확인

`synch/sxlock/innodb/btr_search_latch` 및 `io/table/sql/handler`가 DB 로드 이상에 가장 많이 기여하는 경우 `innodb_adaptive_hash_index` 변수가 활성화되어 있는지 확인합니다. 그렇다면 `innodb_adaptive_hash_index_parts` 파라미터 값을 늘리는 것이 좋습니다.

Adaptive Hash Index가 꺼져 있는 경우 켜는 것이 좋습니다. MySQL Adaptive Hash Index에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Percona 웹 사이트의 문서, InnoDB의 Adaptive Hash Index가 내 워크로드에 적합한가요?
- MySQL 참조 설명서의 Adaptive Hash Index
- Percona 웹 사이트의 문서, MySQL InnoDB에서 경합: Semaphores 섹션의 유용한 정보

> **Note**
>
> Adaptive Hash Index는 Aurora 리더 DB 인스턴스에서 지원되지 않습니다.
> 경우에 따라 `synch/sxlock/innodb/btr_search_latch` 및 `io/table/sql/handler`가 지배적이면 리더 인스턴스에서 성능이 저하될 수 있습니다. 그렇다면 워크로드를 일시적으로 작성기 DB 인스턴스로 리디렉션하고 Adaptive Hash Index를 켜는 것이 좋습니다.

synch/cond/innodb/row_lock_wait

`synch/cond/innodb/row_lock_wait` 이벤트는 한 세션이 업데이트를 위해 행을 잠그고 다른 세션이 동일한 행을 업데이트하려고 할 때 발생합니다. 자세한 내용은 MySQL 설명서의 [InnoDB 잠금](#)을 참조하십시오.

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 3

대기 증가의 가능한 원인

여러 데이터 조작 언어(DML) 문이 동시에 동일한 행이나 행에 액세스하고 있습니다.

작업

표시되는 다른 대기 이벤트에 따라 다른 작업을 권장합니다.

주제

- [이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.](#)
- [차단 세션 찾기 및 응답](#)

이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.

성능 개선 도우미를 사용하여 이 대기 이벤트를 담당하는 SQL 문을 식별합니다. 다음과 같은 전략을 생각해 보세요.

- 행 잠금이 지속적으로 문제가 되는 경우 낙관적 잠금을 사용하도록 애플리케이션을 다시 작성하는 것이 좋습니다.
- 다중 명령문을 사용합니다.
- 서로 다른 데이터베이스 객체에 워크로드를 분산합니다. 파티셔닝을 통해 이를 수행할 수 있습니다.
- `innodb_lock_wait_timeout` 파라미터 값을 확인하세요. 시간 초과 오류가 발생하기 전에 트랜잭션이 대기하는 시간을 제어합니다.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, [성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석](#)을 참조하세요.

차단 세션 찾기 및 응답

차단 세션이 유휴 상태인지 또는 활성 상태인지 확인합니다. 또한 세션이 애플리케이션 또는 활성 사용자에게서 왔는지 확인합니다.

잠금을 유지하는 세션을 식별하려면 `SHOW ENGINE INNODB STATUS`를 실행할 수 있습니다. 다음 예는 샘플 출력을 보여줍니다.

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

또는 다음 쿼리를 사용하여 현재 잠금에 대한 세부 정보를 추출할 수 있습니다.

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
```

```
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

세션을 식별할 때 옵션에는 다음이 포함됩니다.

- 애플리케이션 소유자 또는 사용자에게 문의하세요.
- 차단 세션이 유휴 상태인 경우 차단 세션을 종료하는 것이 좋습니다. 이 작업은 긴 롤백을 트리거할 수 있습니다. 세션을 종료하는 방법에 대한 내용은 세션 또는 쿼리 종료를 참조하세요.

차단 트랜잭션 식별에 대한 자세한 내용은 MySQL 설명서에서 Using InnoDB transaction and locking information을 참조하세요.

synch/cond/innodb/row_lock_wait_cond

synch/cond/innodb/row_lock_wait_cond 이벤트는 한 세션이 업데이트를 위해 행을 잠그고 다른 세션이 동일한 행을 업데이트하려고 할 때 발생합니다. 자세한 내용은 MySQL 설명서의 InnoDB 잠금을 참조하십시오.

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2

대기 증가의 가능한 원인

여러 데이터 조작 언어(DML) 문이 동시에 동일한 행이나 행에 액세스하고 있습니다.

작업

표시되는 다른 대기 이벤트에 따라 다른 작업을 권장합니다.

주제

- 이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.
- 차단 세션 찾기 및 응답

이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.

성능 개선 도우미를 사용하여 이 대기 이벤트를 담당하는 SQL 문을 식별합니다. 다음과 같은 전략을 생각해 보세요.

- 행 잠금이 지속적으로 문제가 되는 경우 낙관적 잠금을 사용하도록 애플리케이션을 다시 작성하는 것이 좋습니다.
- 다중 명령문을 사용합니다.
- 서로 다른 데이터베이스 객체에 워크로드를 분산합니다. 파티셔닝을 통해 이를 수행할 수 있습니다.
- `innodb_lock_wait_timeout` 파라미터 값을 확인하세요. 시간 초과 오류가 발생하기 전에 트랜잭션이 대기하는 시간을 제어합니다.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

차단 세션 찾기 및 응답

차단 세션이 유휴 상태인지 또는 활성 상태인지 확인합니다. 또한 세션이 애플리케이션 또는 활성 사용자에게서 왔는지 확인합니다.

잠금을 유지하는 세션을 식별하려면 SHOW ENGINE INNODB STATUS를 실행할 수 있습니다. 다음 예는 샘플 출력을 보여줍니다.

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

또는 다음 쿼리를 사용하여 현재 잠금에 대한 세부 정보를 추출할 수 있습니다.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
```

```
          ON it.trx_mysql_thread_id = p.id
        JOIN information_schema.innodb_trx it1
          ON ilw.requesting_trx_id = it1.trx_id
        JOIN information_schema.processlist p1
          ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

세션을 식별할 때 옵션에는 다음이 포함됩니다.

- 애플리케이션 소유자 또는 사용자에게 문의하세요.
- 차단 세션이 유휴 상태인 경우 차단 세션을 종료하는 것이 좋습니다. 이 작업은 긴 롤백을 트리거할 수 있습니다. 세션을 종료하는 방법에 대한 내용은 세션 또는 쿼리 종료를 참조하세요.

차단 트랜잭션 식별에 대한 자세한 내용은 MySQL 설명서에서 Using InnoDB transaction and locking information을 참조하세요.

synch/cond/sql/MDL_context::COND_wait_status

`synch/cond/sql/MDL_context::COND_wait_status` 이벤트는 테이블 메타데이터 잠금을 기다리는 스레드가 있는 경우 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트

- [대기 증가의 가능한 원인]
- [작업]

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

`synch/cond/sql/MDL_context::COND_wait_status` 이벤트는 테이블 메타데이터 잠금을 기다리는 스레드가 있음을 나타냅니다. 경우에 따라 한 세션이 테이블에 대해 메타데이터 잠금을 유지하고 다른 세션은 동일한 테이블에서 동일한 잠금을 가져오려고 시도합니다. 이 경우 두 번째 세션은 `synch/cond/sql/MDL_context::COND_wait_status` 대기 이벤트를 대기합니다.

MySQL은 메타데이터 잠금을 사용하여 데이터베이스 객체에 대한 동시 액세스를 관리하고 데이터 일관성을 보장합니다. 메타데이터 잠금은 `get_lock` 기능 및 저장된 프로그램을 통해 획득한 테이블, 스키마, 예약된 이벤트, 테이블스페이스 및 사용자 잠금에 적용됩니다. 저장된 프로그램에는 프로시저, 기능 및 트리거가 포함됩니다. 자세한 내용은 MySQL 설명서의 [메타데이터 잠금]을 참조하세요.

MySQL 프로세스 목록에는 이 세션이 `waiting for metadata lock` 상태로 표시됩니다. 성능 개선 도우미에서, `Performance_schema`가 켜져 있는 경우 `synch/cond/sql/MDL_context::COND_wait_status` 이벤트가 발생합니다.

메타데이터 잠금을 기다리는 쿼리의 기본 시간 초과는 기본값이 31,536,000초(365일)인 `lock_wait_timeout` 파라미터 값을 기반으로 합니다.

서로 다른 InnoDB 잠금 및 충돌을 일으킬 수 있는 잠금 유형에 대한 자세한 내용은 MySQL 설명서에서 [InnoDB 잠금]을 참조하세요.

대기 증가의 가능한 원인

`synch/cond/sql/MDL_context::COND_wait_status` 이벤트가 정상보다 많이 나타나 성능 문제를 나타내는 경우 일반적인 원인은 다음과 같습니다.

장기 실행 트랜잭션

> 하나 이상의 트랜잭션이 대량의 데이터를 수정하고 오랜 시간 동안 테이블 잠금을 유지하고 있습니다.

유휴 트랜잭션

하나 이상의 트랜잭션이 커밋되거나 롤백되지 않고 오랫동안 열려 있습니다.

대형 테이블의 DDL 문

하나 이상의 데이터 정의 언어(DDL) 문(예: `ALTER TABLE` 명령)이 매우 큰 테이블에서 실행되었습니다.

명시적 테이블 잠금

제때에 릴리스되지 않는 테이블에 대한 명시적 잠금이 있습니다. 예를 들어 애플리케이션이 `LOCK TABLE` 문을 부적절하게 실행할 수 있습니다.

작업

대기 이벤트의 원인과 Aurora MySQL DB 클러스터 버전에 따라 다른 작업을 권장합니다.

주제

- [이벤트를 일으키는 세션 및 쿼리 식별](#)
- [지난 이벤트 확인](#)
- [Aurora MySQL 버전 2에서 쿼리 실행](#)
- [차단 세션에 대한 응답](#)

이벤트를 일으키는 세션 및 쿼리 식별

성능 개선 도우미를 사용하여 `synch/cond/sql/MDL_context::COND_wait_status` 대기 이벤트에 의해 차단된 쿼리를 표시할 수 있습니다. 그러나 차단 세션을 식별하려면 DB 클러스터의 `performance_schema` 및 `information_schema`에서 메타데이터 테이블을 쿼리합니다.

일반적으로 보통 로드에서 중요 로드까지 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을 소비하는 위치를 확인합니다. 가장 높은 로드를 일으키는 대기 이벤트를 살펴보고 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

높은 로드에 책임이 있는 SQL 쿼리를 찾으려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.

3. DB 인스턴스를 선택합니다. 해당 DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결에 대한 유용한 개요는 AWS 데이터베이스 블로그 게시물, [성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석]을 참조하세요.

지난 이벤트 확인

이 대기 이벤트에 대한 인사이트를 확보하여 과거의 발생을 확인할 수 있습니다. 이렇게 하려면 다음 작업을 완료합니다.

- 데이터 조작 언어(DML), DDL 처리량 및 대기 시간을 확인하여 워크로드에 변경 사항이 있는지 확인합니다.

  성능 개선 도우미를 사용하여 문제 발생 시 이 이벤트에서 대기 중인 쿼리를 찾을 수 있습니다. 또한 문제 발생 시점에 실행되는 쿼리의 다이제스트도 볼 수 있습니다.
- DB 클러스터에 대해 감사 로그 또는 일반 로그가 켜져 있는 경우 대기 트랜잭션과 관련된 객체(schema.table)에서 실행되는 모든 쿼리를 확인할 수 있습니다. 트랜잭션 전에 실행이 완료된 쿼리를 확인할 수도 있습니다.

과거 이벤트 문제를 해결하는 데 사용할 수 있는 정보는 제한되어 있습니다. 이러한 검사를 수행해도 어떤 객체가 정보를 대기하고 있는지는 표시되지 않습니다. 그러나 이벤트 발생 시 로드가 많은 테이블 및 문제 발생 시 충돌의 원인인 자주 작동하는 행 집합을 식별할 수 있습니다. 그런 다음 이 정보를 사용하여 테스트 환경에서 문제를 재현하고 그 원인에 대한 인사이트를 제공할 수 있습니다.

Aurora MySQL 버전 2에서 쿼리 실행

Aurora MySQL 버전 2에서는 `performance_schema` 테이블 또는 `sys` 스키마 보기를 쿼리하여 차단된 세션을 직접 식별할 수 있습니다. 예를 들어 차단 쿼리 및 세션을 식별하려면 테이블을 쿼리하는 방법을 설명할 수 있습니다.

다음 프로세스 목록 출력에서 연결 ID 89는 메타 데이터 잠금을 기다리고 있으며 `TRUNCATE TABLE` 명령을 실행하고 있습니다. `performance_schema` 테이블 또는 `sys` 스키마 보기에 대한 쿼리에서 출력은 차단 세션이 76임을 표시합니다.

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.11.5           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------+------
+--------------------------------+------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                           |
+----+-----------------+--------------------+-----------+---------+------+------
+--------------------------------+------------------------------+
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                           |
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                           |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                           |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
```

```
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep    |    0 | NULL
                     | NULL                             |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query    |    0 | starting
                     | show processlist                 |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query    |   22 | User sleep
                     | select sleep(10000)              |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query    |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+----------------------------------+--------------------------------+
18 rows in set (0.00 sec)
```

다음으로, performance_schema 테이블 또는 sys 스키마 보기에 대한 쿼리는 차단 세션이 76임을 표시합니다.

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+----------------------
+--------------------------------+--------------------+------------------------------
+-----------------------------+-------------------+-------------
+------------------------------+--------------------+------------------------
+-------------------------+-------------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
           | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account             | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+----------------------
+--------------------------------+--------------------+------------------------------
+-----------------------------+-------------------+-------------
+------------------------------+--------------------+------------------------
+-------------------------+-------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                           0 |
                0 |                 108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
               |
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+----------------------
+--------------------------------+--------------------+------------------------------
```

```
+-----------------------------+--------------------+--------------
+------------------------------+--------------------+------------------------
+-------------------------+------------------------------+
1 row in set (0.00 sec)
```

차단 세션에 대한 응답

세션을 식별할 때 옵션에는 다음이 포함됩니다.

- 애플리케이션 소유자 또는 사용자에게 문의하세요.
- 차단 세션이 유휴 상태인 경우 차단 세션을 종료하는 것이 좋습니다. 이 작업은 긴 롤백을 트리거할 수 있습니다. 세션을 종료하는 방법에 대한 내용은 세션 또는 쿼리 종료를 참조하세요.

차단 트랜잭션 식별에 대한 자세한 내용은 MySQL 설명서에서 InnoDB 트랜잭션 및 잠금 정보 사용을 참조하세요.

synch/mutex/innodb/aurora_lock_thread_slot_futex

`synch/mutex/innodb/aurora_lock_thread_slot_futex` 이벤트는 한 세션이 업데이트를 위해 행을 잠그고 다른 세션이 동일한 행을 업데이트하려고 할 때 발생합니다. 자세한 내용은 MySQL 설명서의 InnoDB 잠금을 참조하세요.

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2

대기 증가의 가능한 원인

여러 데이터 조작 언어(DML) 문이 동시에 동일한 행이나 행에 액세스하고 있습니다.

작업

표시되는 다른 대기 이벤트에 따라 다른 작업을 권장합니다.

주제

- 이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.
- 차단 세션 찾기 및 응답

이 대기 이벤트를 담당하는 SQL 문을 찾아 응답합니다.

성능 개선 도우미를 사용하여 이 대기 이벤트를 담당하는 SQL 문을 식별합니다. 다음과 같은 전략을 생각해 보세요.

- 행 잠금이 지속적으로 문제가 되는 경우 낙관적 잠금을 사용하도록 애플리케이션을 다시 작성하는 것이 좋습니다.
- 다중 명령문을 사용합니다.
- 서로 다른 데이터베이스 객체에 워크로드를 분산합니다. 파티셔닝을 통해 이를 수행할 수 있습니다.
- `innodb_lock_wait_timeout` 파라미터 값을 확인하세요. 시간 초과 오류가 발생하기 전에 트랜잭션이 대기하는 시간을 제어합니다.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, [성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석](#)을 참조하세요.

차단 세션 찾기 및 응답

차단 세션이 유휴 상태인지 또는 활성 상태인지 확인합니다. 또한 세션이 애플리케이션 또는 활성 사용자에게서 왔는지 확인합니다.

잠금을 유지하는 세션을 식별하려면 `SHOW ENGINE INNODB STATUS`를 실행할 수 있습니다. 다음 예는 샘플 출력을 보여줍니다.

```
mysql> SHOW ENGINE INNODB STATUS;

---------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

또는 다음 쿼리를 사용하여 현재 잠금에 대한 세부 정보를 추출할 수 있습니다.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
```

```
               p1.host waiting_host,
               it1.trx_query waiting_query,
               ilw.requesting_trx_id waiting_transaction,
               ilw.blocking_lock_id blocking_lock,
               il.lock_mode blocking_mode,
               il.lock_type blocking_type,
               ilw.blocking_trx_id blocking_transaction,
               CASE it.trx_state
                 WHEN 'LOCK WAIT'
                 THEN it.trx_state
                 ELSE p.state
               END blocker_state,
               il.lock_table locked_table,
               it.trx_mysql_thread_id blocker_thread,
               p.user blocker_user,
               p.host blocker_host
        FROM information_schema.innodb_lock_waits ilw
        JOIN information_schema.innodb_locks il
          ON ilw.blocking_lock_id = il.lock_id
         AND ilw.blocking_trx_id = il.lock_trx_id
        JOIN information_schema.innodb_trx it
          ON ilw.blocking_trx_id = it.trx_id
        JOIN information_schema.processlist p
          ON it.trx_mysql_thread_id = p.id
        JOIN information_schema.innodb_trx it1
          ON ilw.requesting_trx_id = it1.trx_id
        JOIN information_schema.processlist p1
          ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
```

```
1 row in set (0.04 sec)
```

세션을 식별할 때 옵션에는 다음이 포함됩니다.

- 애플리케이션 소유자 또는 사용자에게 문의하세요.
- 차단 세션이 유휴 상태인 경우 차단 세션을 종료하는 것이 좋습니다. 이 작업은 긴 롤백을 트리거할 수 있습니다. 세션을 종료하는 방법에 대한 내용은 세션 또는 쿼리 종료를 참조하세요.

차단 트랜잭션 식별에 대한 자세한 내용은 MySQL 참조 설명서에서 InnoDB 트랜잭션 및 잠금 정보 사용을 참조하세요.

synch/mutex/innodb/buf_pool_mutex

`synch/mutex/innodb/buf_pool_mutex` 이벤트는 메모리의 페이지에 액세스하기 위해 스레드가 InnoDB 버퍼 풀에서 잠금을 획득한 경우 발생합니다.

주제

- 관련 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

관련 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2

컨텍스트

`buf_pool` 뮤텍스는 버퍼 풀의 제어 데이터 구조를 보호하는 단일 뮤텍스입니다.

자세한 내용은 MySQL 설명서의 성능 스키마를 사용하여 InnoDB 뮤텍스 대기 모니터링을 참조하세요.

대기 증가의 가능한 원인

이는 워크로드별 대기 이벤트입니다. 상위 대기 이벤트에 나타나는 `synch/mutex/innodb/buf_pool_mutex`에 대한 일반적인 원인은 다음과 같습니다.

- 버퍼 풀 크기는 작업 중인 데이터 세트를 보유하기에 충분하지 않습니다.
- 워크로드는 데이터베이스의 특정 테이블의 특정 페이지에 보다 구체적으로 지정되므로 버퍼 풀에서 경합이 발생할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 이벤트를 일으키는 세션 및 쿼리 식별
- 성능 개선 도우미 사용
- Aurora 복제본 생성
- 버퍼 풀 크기 검사
- 전역적 상태 이력 모니터링

이벤트를 일으키는 세션 및 쿼리 식별

일반적으로 보통 로드에서 중요 로드까지 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을 소비하는 위치를 확인합니다. 가장 높은 로드를 일으키는 대기 이벤트를 살펴보고 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

AWS 관리 콘솔에서 상위 SQL 차트를 보려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. DB 인스턴스를 선택합니다. DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 데이터베이스 로드(Database load) 차트에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

성능 개선 도우미 사용

이 이벤트는 워크로드와 관련이 있습니다. 성능 개선 도우미를 사용하여 다음을 수행할 수 있습니다.

- 대기 이벤트가 시작되는 시점과 애플리케이션 로그 또는 관련 소스에서 해당 시점에 워크로드가 변경되었는지 여부를 식별합니다.
- 이 대기 이벤트를 담당하는 SQL 문을 식별합니다. 쿼리의 실행 계획을 검토하여 이러한 쿼리가 최적화되고 적절한 인덱스를 사용하는지 확인합니다.

  대기 이벤트를 담당하는 상위 쿼리가 동일한 데이터베이스 객체 또는 테이블과 관련이 있는 경우 해당 객체 또는 테이블을 분할하는 것이 좋습니다.

Aurora 복제본 생성

Aurora 복제본을 생성하여 읽기 전용 트래픽을 처리할 수 있습니다. Aurora Auto Scaling을 사용하여 읽기 트래픽의 서지를 처리할 수도 있습니다. Aurora 복제본에서 예약된 읽기 전용 작업과 논리적 백업을 실행해야 합니다.

자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 단원을 참조하십시오.

버퍼 풀 크기 검사

`innodb_buffer_pool_wait_free` 지표를 검토하여 버퍼 풀 크기가 워크로드에 충분한지 확인합니다. 이 지표 값이 높고 지속적으로 증가하는 경우는 버퍼 풀의 크기가 작업 로드를 처리하기에 충분하지 않음을 나타냅니다. `innodb_buffer_pool_size`가 제대로 설정된 경우 `innodb_buffer_pool_wait_free`의 값은 작아야 합니다. 자세한 내용은 MySQL 설명서의 Innodb_buffer_pool_wait_free를 참조하세요.

DB 인스턴스에 세션 버퍼 및 운영 체제 작업을 위한 메모리가 충분한 경우 버퍼 풀 크기를 늘립니다. 그렇지 않으면 DB 인스턴스를 더 큰 DB 인스턴스 클래스로 변경하여 버퍼 풀에 할당할 수 있는 추가 메모리를 가져옵니다.

> **Note**
>
> Aurora MySQL은 구성된 `innodb_buffer_pool_size`에 따라 `innodb_buffer_pool_instances` 값을 자동으로 조정합니다.

전역적 상태 이력 모니터링

상태 변수의 변경률을 모니터링하여 DB 인스턴스에서 잠금 또는 메모리 문제를 감지할 수 있습니다. 아직 켜져 있지 않은 경우 전역적 상태 이력(GoSH)을 켭니다. GoSH에 대한 자세한 내용은 [전역적 상태 이력 관리]를 참조하세요.

상태 변수를 모니터링하기 위해 사용자 지정 Amazon CloudWatch 지표를 생성할 수도 있습니다. 자세한 내용은 [사용자 지정 지표 게시]를 참조하세요.

synch/mutex/innodb/fil_system_mutex

`synch/mutex/innodb/fil_system_mutex` 이벤트는 세션이 테이블스페이스 메모리 캐시에 액세스하려고 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

InnoDB는 테이블스페이스를 사용하여 테이블 및 로그 파일의 스토리지 영역을 관리합니다. 테이블스페이스 메모리 캐시(tablespace memory cache)는 테이블스페이스에 대한 정보를 유지 관리하는 전역적 메모리 구조입니다. MySQL은 `synch/mutex/innodb/fil_system_mutex` 대기를 사용하여 테이블스페이스 메모리 캐시에 대한 동시 액세스를 제어합니다.

`synch/mutex/innodb/fil_system_mutex` 이벤트는 현재 동일한 테이블스페이스의 테이블스페이스 메모리 캐시 정보를 검색하고 조작해야 하는 작업이 두 개 이상임을 나타냅니다.

대기 증가의 가능한 원인

`synch/mutex/innodb/fil_system_mutex` 이벤트가 정상보다 더 많이 나타나 성능 문제를 나타낼 수 있는 경우는 일반적으로 다음 조건이 모두 충족된 경우에 발생합니다.

- 동일한 테이블에서 데이터를 업데이트하거나 삭제하는 동시 데이터 조작 언어(DML) 작업이 늘어납니다.
- 이 테이블의 테이블스페이스는 매우 크며, 많은 데이터 페이지가 있습니다.
- 이러한 데이터 페이지의 채우기 비율은 낮습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 이벤트를 일으키는 세션 및 쿼리 식별
- 사용량이 적은 시간대에 대형 테이블 재구성

이벤트를 일으키는 세션 및 쿼리 식별

일반적으로 보통 로드에서 중요 로드까지 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을 소비하는 위치를 확인합니다. 가장 높은 로드를 일으키는 대기 이벤트를 살펴보고 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

높은 로드에 책임이 있는 SQL 쿼리를 찾으려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다. 해당 DB 인스턴스에 대해 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

다음 예와 같이 어떤 쿼리가 많은 `synch/mutex/innodb/fil_system_mutex` 대기를 일으키는지 알아내는 다른 방법은 `performance_schema`를 확인하는 것입니다.

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
```

```
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

사용량이 적은 시간대에 대형 테이블 재구성

프로덕션 시간대 이외의 유지 관리 기간 동안 많은 수의 `synch/mutex/innodb/fil_system_mutex` 대기 이벤트 소스로 식별되는 대형 테이블을 재구성합니다. 이렇게 하면 테이블에 대한 빠른 액세스가 중요한 경우 내부 테이블스페이스 맵 정리가 수행되지 않아야 합니다. 테이블 재구성에 대한 자세한 내용은 MySQL 참조의 OPTIMIZE TABLE 문을 참조하세요.

synch/mutex/innodb/trx_sys_mutex

`synch/mutex/innodb/trx_sys_mutex` 이벤트는 많은 트랜잭션 수가 있는 데이터베이스 작업이 많을 때 발생합니다.

주제

- 관련 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

관련 엔진 버전

이 대기 이벤트 정보는 다음 엔진 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

내부적으로 InnoDB 데이터베이스 엔진은 스냅샷을 통해 반복 가능한 읽기 격리 수준을 사용하여 읽기 일관성을 유지합니다. 이렇게 하면 스냅샷이 생성될 때 데이터베이스의 특정 시점 보기가 제공됩니다.

InnoDB에서는 변경 사항이 커밋되었는지 여부와 관계없이 모든 변경 사항이 도착하자마자 데이터베이스에 적용됩니다. 이 방법은 다중 버전 동시성 제어(MVCC)가 없으면 데이터베이스에 연결된 모든 사용자가 모든 변경 사항 및 최신 행을 볼 수 있음을 의미합니다. 따라서 InnoDB는 필요한 경우 롤백해야 할 내용을 이해하기 위해 변경 사항을 추적하는 방법이 필요합니다.

이렇게 하려면 InnoDB는 트랜잭션 시스템(`trx_sys`)을 사용하여 스냅샷을 추적합니다. 트랜잭션 시스템은 다음 작업을 수행합니다.

- 실행 취소 로그의 각 행에 대한 트랜잭션 ID를 추적합니다.
- `ReadView`라는 내부 InnoDB 구조를 사용하여 스냅샷에 표시되는 트랜잭션 ID를 식별하는 데 도움을 줄 수 있습니다.

대기 증가의 가능한 원인

트랜잭션 ID의 일관되고 제어된 처리(생성, 읽기, 업데이트 및 삭제)가 필요한 모든 데이터베이스 작업은 `trx_sys`의 호출을 뮤텍스로 생성합니다.

이러한 호출은 세 가지 기능 내에서 발생합니다.

- `trx_sys_mutex_enter` - 뮤텍스를 생성합니다.
- `trx_sys_mutex_exit` - 뮤텍스를 해제합니다.
- `trx_sys_mutex_own` - 뮤텍스 소유 여부를 테스트합니다.

InnoDB 성능 스키마 계측은 모든 `trx_sys` 뮤텍스 호출을 추적합니다. 추적에는 데이터베이스 시작 또는 종료, 롤백 작업, 정리 실행 취소, 행 읽기 액세스 및 버퍼 풀 로드 시 `trx_sys`의 관리가 포함되지만 이에 국한되지는 않습니다. 많은 수의 트랜잭션을 통한 데이터베이스 작업이 많으면 `synch/mutex/innodb/trx_sys_mutex`가 상위 대기 이벤트에 표시됩니다.

자세한 내용은 MySQL 설명서의 [성능 스키마를 사용하여 InnoDB 뮤텍스 대기 모니터링](#)을 참조하세요.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 이벤트를 일으키는 세션 및 쿼리 식별
- 다른 대기 이벤트 검사

이벤트를 일으키는 세션 및 쿼리 식별

일반적으로 보통 로드에서 중요 로드까지 있는 데이터베이스에는 대기 이벤트가 있습니다. 성능이 최적이면 대기 이벤트가 허용될 수 있습니다. 성능이 최적이 아닌 경우 데이터베이스가 가장 많은 시간을 소비하는 위치를 확인합니다. 가장 큰 로드에 기여하는 대기 이벤트를 살펴보세요. 데이터베이스와 애플리케이션을 최적화하여 이러한 이벤트를 줄일 수 있는지 확인합니다.

AWS Management Console에서 상위 SQL 차트를 보려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. DB 인스턴스를 선택합니다. DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 데이터베이스 로드(Database load) 차트에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결의 개요는 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

다른 대기 이벤트 검사

`synch/mutex/innodb/trx_sys_mutex` 대기 이벤트와 연결된 다른 대기 이벤트를 검사합니다. 이렇게 하면 워크로드의 특성에 대해 추가 정보를 얻을 수 있습니다. 트랜잭션이 많으면 처리량이 줄어들 수 있지만 워크로드로 인해 이 작업이 필요할 수도 있습니다.

트랜잭션 최적화 방법에 대한 자세한 내용은 MySQL 설명서에서 InnoDB 트랜잭션 관리 최적화를 참조하세요.

synch/sxlock/innodb/hash_table_locks

`synch/sxlock/innodb/hash_table_locks` 이벤트는 버퍼 풀에서 찾을 수 없는 페이지를 스토리지에서 읽어야 할 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 다음 버전에서 지원됩니다.

- Aurora MySQL 버전 2 및 3

컨텍스트

`synch/sxlock/innodb/hash_table_locks` 이벤트는 워크로드가 버퍼 풀에 저장되지 않은 데이터에 자주 액세스하고 있음을 나타냅니다. 이 대기 이벤트는 버퍼 풀에서 새 페이지 추가 및 이전 데이터 제거와 연결됩니다. 버퍼 풀에 저장된 데이터는 오래된 데이터와 새 데이터를 캐시해야 하므로 오래된 페이지는 새 페이지의 캐싱을 허용하도록 제거됩니다. MySQL은 최소최근사용(LRU) 알고리즘을 사용하여 버퍼 풀에서 페이지를 제거합니다. 워크로드가 버퍼 풀에 로드되지 않은 데이터 또는 버퍼 풀에서 제거된 데이터에 액세스하려고 합니다.

이 대기 이벤트는 워크로드가 디스크의 파일에 있는 데이터에 액세스해야 하거나 블록이 버퍼 풀의 LRU 목록에서 해제되거나 추가될 때 발생합니다. 이러한 작업은 공유 배타 잠금(SX-lock)을 획득하기 위해 대기합니다. 이 SX-lock은 버퍼 풀 액세스 성능을 향상하도록 설계된 메모리의 테이블인 해시 테이블(hash table)을 통한 동기화에 사용됩니다.

자세한 내용은 MySQL 설명서의 버퍼 풀을 참조하세요.

대기 증가의 가능한 원인

`synch/sxlock/innodb/hash_table_locks` 대기 이벤트가 정상보다 많이 나타나 성능 문제를 나타낼 수 있는 경우 일반적인 원인은 다음과 같습니다.

크기가 작은 버퍼 풀

버퍼 풀의 크기가 너무 작아서 자주 액세스하는 모든 페이지를 메모리에 보관할 수 없습니다.

많은 워크로드

워크로드로 인해 자주 제거되고 버퍼 캐시에서 데이터 페이지가 다시 로드됩니다.

페이지를 읽는 중 오류

버퍼 풀에서 페이지를 읽는 중에 오류가 발생하여 데이터 손상을 나타낼 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 버퍼 풀의 크기 늘리기
- 데이터 액세스 패턴 개선
- 전체 테이블 스캔 감소 또는 방지
- 오류 로그에서 페이지 손상 확인

버퍼 풀의 크기 늘리기

버퍼 풀의 크기가 워크로드에 맞게 적절한지 확인합니다. 이렇게 하면 버퍼 풀 캐시 적중률을 확인할 수 있습니다. 일반적으로 값이 95% 미만으로 떨어지면 버퍼 풀 크기를 늘리는 것이 좋습니다. 버퍼 풀이 클수록 자주 액세스하는 페이지를 메모리에 더 오래 유지할 수 있습니다. 버퍼 풀의 크기를 늘리려면 `innodb_buffer_pool_size` 파라미터 값을 수정합니다. 이 파라미터의 기본값은 DB 인스턴스 클래스 크기를 기반으로 합니다. 자세한 내용은 Amazon Aurora MySQL 데이터베이스 구성에 대한 모범 사례를 참조하세요.

데이터 액세스 패턴 개선

이 대기 및 실행 계획의 영향을 받는 쿼리를 확인합니다. 데이터 액세스 패턴을 개선하는 것이 좋습니다. 예를 들어 mysqli_result::fetch_array를 사용 중인 경우 배열 가져오기 크기를 늘릴 수 있습니다.

성능 개선 도우미를 사용하여 `synch/sxlock/innodb/hash_table_locks` 대기 이벤트를 일으킬 수 있는 쿼리 및 세션을 표시할 수 있습니다.

높은 로드에 책임이 있는 SQL 쿼리를 찾으려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Performance Insights(성능 개선 도우미)를 선택합니다.
3. DB 인스턴스를 선택합니다. DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(Database load) 차트에서 대기별 조각(Slice by wait)을 선택합니다.
5. 페이지 하단에서 상위 SQL(Top SQL)을 선택합니다.

   차트에는 로드에 책임이 있는 SQL 쿼리가 나열됩니다. 목록의 맨 위에 있는 쿼리가 가장 큰 책임이 있습니다. 병목 현상을 해결하려면 다음 명령문에 집중하세요.

성능 개선 도우미를 통한 문제 해결에 대한 유용한 개요는 AWS 데이터베이스 블로그 게시물, 성능 개선 도우미를 통해 Amazon Aurora MySQL 워크로드 분석을 참조하세요.

전체 테이블 스캔 감소 또는 방지

워크로드를 모니터링하여 전체 테이블 스캔이 실행되고 있는지 확인하고, 있는 경우 워크로드를 줄이거나 방지할 수 있습니다. 예를 들어 `Handler_read_rnd_next`와 같은 상태 변수를 모니터링할 수 있습니다. 자세한 내용은 MySQL 설명서에서 서버 상태 변수를 참조하세요.

오류 로그에서 페이지 손상 확인

mysql-error.log에서 문제 발생 시점에 감지된 손상 관련 메시지가 있는지 확인할 수 있습니다. 문제를 해결하기 위해 작업할 수 있는 메시지는 오류 로그에 있습니다. 손상된 것으로 보고된 객체를 다시 작성해야 할 수 있습니다.

스레드 상태로 Aurora MySQL 튜닝

다음 테이블에는 Aurora MySQL에 대한 가장 일반적인 스레드 상태가 요약되어 있습니다.

일반 스레드 상태	설명
???	이 스레드 상태는 스레드가 데이터를 정렬하기 위해 내부 임시 테이블을 사용해야 하는 SELECT 문을 처리하고 있음을 나타냅니다.
???	이 스레드 상태는 스레드가 올바른 결과 집합을 결정하기 위해 쿼리의 행을 읽고 필터링하고 있음을 나타냅니다.

정렬 인덱스 생성

`creating sort index` 스레드 상태는 스레드가 데이터를 정렬하기 위해 내부 임시 테이블을 사용해야 하는 SELECT 문을 처리하고 있음을 나타냅니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 스레드 상태 정보는 다음 버전에서 지원됩니다.

- Aurora MySQL 버전 2에서 최대 2.09.2까지

컨텍스트

`creating sort index` 상태는 ORDER BY 또는 GROUP BY 절이 있는 쿼리가 기존 인덱스를 사용하여 작업을 수행할 수 없는 경우 표시됩니다. 이 경우 MySQL은 더 비싼 `filesort` 작업을 수행해야 합니다. 이 작업은 일반적으로 결과 집합이 너무 크지 않은 경우 메모리에서 수행됩니다. 그렇지 않은 경우 디스크에 파일을 만드는 작업이 포함됩니다.

대기 증가의 가능한 원인

`creating sort index`의 발생 그 자체로는 문제를 나타내지 않습니다. 성능이 좋지 않고 `creating sort index` 인스턴스가 자주 표시되는 경우 가장 가능성 있는 원인은 `ORDER BY` 또는 `GROUP BY` 연산자를 통한 속도가 느린 쿼리입니다.

작업

일반적인 지침은 `creating sort index` 상태의 증가와 연결된 `ORDER BY` 또는 `GROUP BY` 절이 있는 쿼리를 찾는 것입니다. 그런 다음 인덱스를 추가하거나 정렬 버퍼 크기를 늘리면 문제가 해결되는지 확인합니다.

주제

- 성능 스키마가 켜져 있지 않으면 성능 스키마를 켭니다.
- 문제 쿼리 식별
- filesort 사용에 대한 설명 계획 검토
- 정렬 버퍼 크기 늘리기

성능 스키마가 켜져 있지 않으면 성능 스키마를 켭니다.

성능 개선 도우미는 성능 스키마 도구가 켜져 있지 않은 경우에만 스레드 상태를 보고합니다. 성능 스키마 도구가 켜져 있으면 성능 개선 도우미는 대신 대기 이벤트를 보고합니다. 성능 스키마 도구는 잠재적인 성능 문제를 조사하는 경우 추가적인 인사이트와 더 나은 도구를 제공합니다. 따라서 성능 스키마를 켜는 것이 좋습니다. 자세한 내용은 Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요 단원을 참조하십시오.

문제 쿼리 식별

`creating sort index` 상태의 증가를 일으키는 현재 쿼리를 식별하려면 `show processlist`를 실행한 다음 쿼리에 `ORDER BY` 또는 `GROUP BY`가 있는지 확인합니다. 선택적으로 `N`이 `filesort`가 있는 쿼리의 프로세스 목록 ID인 `explain for connection N`을 실행합니다.

이러한 증가를 유발하는 이전 쿼리를 식별하려면 느린 쿼리 로그를 켜고 `ORDER BY`로 해당 쿼리를 찾습니다. 느린 쿼리에서 `EXPLAIN`을 실행한 다음 'filesort 사용'을 찾으세요. 자세한 내용은 filesort 사용에 대한 설명 계획 검토 단원을 참조하십시오.

filesort 사용에 대한 설명 계획 검토

`ORDER BY` 또는 `GROUP BY` 절이 있는 명령문을 식별하면 `creating sort index` 상태로 됩니다.

다음 예에서는 쿼리에서 explain을 실행하는 방법을 보여줍니다. Extra 열은 이 쿼리가 filesort를 사용하는 것을 보여줍니다.

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

다음 예에서는 c1 열에 인덱스가 생성된 후 동일한 쿼리에서 EXPLAIN을 실행한 결과를 보여줍니다.

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

정렬 순서 최적화를 위한 인덱스 사용에 대한 자세한 내용은 MySQL 설명서의 ORDER BY 최적화를 참조하세요.

정렬 버퍼 크기 늘리기

특정 쿼리에 디스크에 파일을 만든 `filesort` 프로세스가 필요한지 여부를 확인하려면 쿼리를 실행한 후 `sort_merge_passes` 변수 값을 확인합니다. 다음은 그 한 예입니다.

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

`sort_merge_passes`의 값이 큰 경우 정렬 버퍼 크기를 늘리는 것이 좋습니다. 세션 수준에서 증가를 적용하세요. 전역적으로 늘리면 MySQL의 RAM 사용량이 크게 늘어날 수 있기 때문입니다. 다음 예에서는 쿼리를 실행하기 전에 정렬 버퍼 크기를 변경하는 방법을 보여줍니다.

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

데이터 전송

`sending data` 스레드 상태는 스레드가 올바른 결과 집합을 결정하기 위해 쿼리의 행을 읽고 필터링하고 있음을 나타냅니다. 이름은 상태가 나중에 전송할 데이터를 수집 및 준비하지 않고 데이터를 전송하고 있음을 의미하기 때문에 오해의 소지가 있습니다.

주제

- [지원되는 엔진 버전]

- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 스레드 상태 정보는 다음 버전에서 지원됩니다.

- Aurora MySQL 버전 2에서 최대 2.09.2까지

컨텍스트

대부분 스레드 상태는 오래 지속되지 않습니다. `sending data` 도중 발생하는 작업은 많은 수의 디스크 또는 캐시 읽기를 수행하는 경향이 있습니다. 따라서 `sending data`는 주어진 쿼리의 수명 동안 가장 긴 실행 상태인 경우가 많습니다. 이 상태는 Aurora MySQL에서 다음을 수행할 때 표시됩니다.

- `SELECT` 문의 행 읽기 및 처리
- 디스크 또는 메모리에서 많은 수의 읽기 수행
- 특정 쿼리에서 모든 데이터에 대한 전체 읽기 완료
- 테이블, 인덱스 또는 저장 프로시저의 작업에서 데이터 읽기
- 데이터 정렬, 그룹화 또는 순서 지정

`sending data` 상태가 데이터 준비를 완료한 다음의 `writing to net` 스레드 상태는 클라이언트로 데이터 반환을 나타냅니다. 일반적으로 `writing to net`은 결과 집합이 매우 크거나 네트워크 대기 시간이 심각해 전송 속도가 느려지는 경우에만 캡처됩니다.

대기 증가의 가능한 원인

`sending data`의 발생 그 자체로는 문제를 나타내지 않습니다. 성능이 좋지 않고 `sending data` 인스턴스가 자주 표시되는 경우 가장 가능성 있는 원인은 다음과 같습니다.

주제

- 비효율적인 쿼리
- 최적이 아닌 서버 구성

비효율적인 쿼리

대부분의 경우 이 상태를 담당하는 것은 특정 쿼리의 결과 집합을 찾기 위해 적절한 인덱스를 사용하지 않는 쿼리입니다. 예를 들어 상태 열이 인덱싱되지 않거나 제대로 인덱싱되지 않은 캘리포니아에 배치된 모든 주문에 대해 1천만 개의 레코드 테이블을 읽는 쿼리를 가정해 보겠습니다. 후자의 경우 인덱스가 존재할 수도 있지만 최적화 프로그램은 카디널리티가 낮아 인덱스를 무시합니다.

최적이 아닌 서버 구성

`sending data` 상태에 여러 쿼리가 표시되는 경우 데이터베이스 서버가 제대로 구성되지 않을 수 있습니다. 특히 서버에 다음과 같은 문제가 있을 수 있습니다.

- 데이터베이스 서버에는 디스크 I/O, 디스크 유형 및 속도, CPU 또는 CPU 수와 같은 컴퓨팅 용량이 충분하지 않습니다.
- 서버는 InnoDB 테이블의 InnoDB 버퍼 풀 또는 MyISAM 테이블의 키 버퍼와 같이 할당된 리소스가 부족합니다.
- 스레드별 메모리 설정(예:`sort_buffer`, `read_buffer` 및 `join_buffer`)은 필요한 것보다 많은 RAM을 소비하고 메모리 리소스의 물리적 서버가 부족합니다.

작업

일반적인 지침은 성능 스키마를 확인하여 많은 수의 행을 반환하는 쿼리를 찾는 것입니다. 인덱스를 사용하지 않는 로깅 쿼리가 켜져 있는 경우 느린 로그의 결과를 검사할 수도 있습니다.

주제

- [성능 스키마가 켜져 있지 않으면 성능 스키마를 켭니다.]
- [메모리 설정 검사]
- [인덱스 사용량에 대한 설명 계획 검토]
- [반환된 데이터의 양 확인]
- [동시성 문제 확인]
- [요청 구조 확인]

성능 스키마가 켜져 있지 않으면 성능 스키마를 켭니다.

성능 개선 도우미는 성능 스키마 도구가 켜져 있지 않은 경우에만 스레드 상태를 보고합니다. 성능 스키마 도구가 켜져 있으면 성능 개선 도우미는 대신 대기 이벤트를 보고합니다. 성능 스키마 도구는 잠

재적인 성능 문제를 조사하는 경우 추가적인 인사이트와 더 나은 도구를 제공합니다. 따라서 성능 스키마를 켜는 것이 좋습니다. 자세한 내용은 [Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요](#) 단원을 참조하십시오.

메모리 설정 검사

기본 버퍼 풀의 메모리 설정을 검사합니다. 이러한 풀의 크기가 워크로드에 적절한지 확인합니다. 데이터베이스에서 여러 버퍼 풀 인스턴스를 사용하는 경우 여러 개의 작은 버퍼 풀로 분할되지 않았는지 확인합니다. 스레드는 한 번에 하나의 버퍼 풀만 사용할 수 있습니다.

각 스레드에 사용되는 다음 메모리 설정의 크기가 적절한지 확인합니다.

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

설정을 수정할 특별한 이유가 없는 한 기본값을 사용합니다.

인덱스 사용량에 대한 설명 계획 검토

`sending data` 스레드 상태에 대한 쿼리의 경우 계획을 검토하여 적절한 인덱스가 사용되는지 확인합니다. 쿼리가 유용한 인덱스를 사용하지 않는 경우 `USE INDEX` 또는 `FORCE INDEX`와 같은 힌트를 추가하는 것이 좋습니다. 힌트는 쿼리를 실행하는 데 걸리는 시간을 크게 늘리거나 줄일 수 있으므로 추가하기 전에 주의해야 합니다.

반환된 데이터의 양 확인

쿼리되는 테이블과 테이블에 포함된 데이터의 양을 확인합니다. 이 데이터를 모두 아카이브할 수 있나요? 대부분의 경우 부족한 쿼리 실행 시간의 원인은 쿼리 계획의 결과가 아니라 처리할 데이터 볼륨입니다. 많은 개발자가 데이터베이스에 데이터를 매우 효율적으로 추가하지만, 설계 및 개발 단계에서는 데이터 세트 수명 주기를 거의 고려하지 않습니다.

작은 볼륨의 데이터베이스에서는 잘 수행되지만 현재 시스템에서는 제대로 수행되지 않는 쿼리를 찾습니다. 특정 쿼리를 설계하는 개발자는 때로는 이러한 쿼리가 350,000개의 행을 반환한다는 사실을 인식하지 못할 수도 있습니다. 개발자는 프로덕션 환경보다 작은 데이터 세트를 사용하여 볼륨이 작은 환경에서 쿼리를 개발했을 수 있습니다.

동시성 문제 확인

동일한 유형의 여러 쿼리가 동시에 실행되고 있는지 확인합니다. 일부 형식의 쿼리는 단독으로 실행될 때 효율적으로 실행됩니다. 그러나 유사한 형식의 쿼리가 함께 실행되거나 많은 볼륨으로 실행되면 동시성 문제가 발생할 수 있습니다. 데이터베이스가 임시 테이블을 사용하여 결과를 렌더링할 때 이러한 문제가 발생하는 경우가 많습니다. 제한적인 트랜잭션 격리 수준은 동시성 문제를 일으킬 수도 있습니다.

테이블을 동시에 읽고 쓰는 경우 데이터베이스가 잠금을 사용하고 있을 수 있습니다. 성능 저하 기간을 식별하려면 대규모 배치 프로세스를 통해 데이터베이스 사용을 검토합니다. 최근 잠금 및 롤백을 확인하려면 `SHOW ENGINE INNODB STATUS` 명령의 결과를 검토합니다.

요청 구조 확인

이러한 상태에서 캡처된 쿼리가 하위 쿼리를 사용하는지 확인합니다. 이러한 유형의 쿼리는 데이터베이스가 내부적으로 결과를 컴파일한 다음 데이터를 렌더링하기 위해 쿼리로 다시 대체하기 때문에 성능이 저하되는 경우가 많습니다. 이 프로세스는 데이터베이스에 대한 추가 단계입니다. 대부분의 경우 이 단계는 동시 로드 조건에서 성능이 저하될 수 있습니다.

또한 쿼리가 많은 수의 `ORDER BY` 및 `GROUP BY` 절을 사용하는지 확인합니다. 이러한 작업에서는 데이터베이스가 먼저 메모리에 전체 데이터 세트를 구성해야 하는 경우가 많습니다. 그런 다음 클라이언트로 반환하기 전에 특정 방식으로 주문하거나 그룹화해야 합니다.

Amazon DevOps Guru의 사전 예방 인사이트를 활용하여 Aurora MySQL 조정

DevOps Guru의 사전 예방 인사이트는 Aurora MySQL DB 클러스터에서 알려진 문제 상태가 발생하기 전에 이를 감지합니다. DevOps Guru는 다음과 같은 작업을 수행할 수 있습니다.

- 데이터베이스 구성을 일반적인 권장 설정과 교차 검사하여 여러 가지 일반적인 데이터베이스 문제를 방지합니다.
- 확인하지 않은 상태로 두면 나중에 더 큰 문제로 이어질 수 있는 플릿의 심각한 문제를 알려줍니다.
- 새로 발견된 문제를 알려줍니다.

모든 사전 예방 인사이트에는 문제의 원인에 대한 분석과 수정 조치를 위한 권장 사항이 포함됩니다.

주제

InnoDB 기록 목록 길이가 크게 늘어남

##부터 행 변경에 대한 기록 목록이 *db-instance*의 ##까지 크게 늘어났습니다. 이 증가는 쿼리 및 데이터베이스 종료 성능에 영향을 줍니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 이 문제의 잠재적 원인
- 작업
- 관련 지표

지원되는 엔진 버전

이 인사이트 정보는 Aurora MySQL의 모든 버전에서 지원됩니다.

컨텍스트

InnoDB 트랜잭션 시스템은 다중 버전 동시성 제어(MVCC)를 유지합니다. 행이 수정되면 수정 중인 데이터의 수정 전 버전이 실행 취소 로그에 실행 취소 레코드로 저장됩니다. 모든 실행 취소 레코드에는 이전 실행 취소 레코드에 대한 참조가 있어 연결된 목록을 형성합니다.

InnoDB 기록 목록은 커밋된 트랜잭션에 대한 실행 취소 로그의 전체 목록입니다. MySQL 트랜잭션에 더 이상 기록이 필요하지 않은 경우 기록 목록을 사용하여 레코드 및 로그 페이지를 제거합니다. 기록 목록 길이는 기록 목록에 수정 사항이 포함된 실행 취소 로그의 총 수입니다. 각 로그에는 하나 이상의 수정이 포함되어 있습니다. InnoDB 기록 목록 길이가 너무 길어지면 이전 행 버전이 많다는 것을 의미하며 쿼리 및 데이터베이스 종료 속도가 느려집니다.

이 문제의 잠재적 원인

긴 기록 목록의 일반적인 원인은 다음과 같습니다.

- 장기 실행 트랜잭션(읽기 또는 쓰기)
- 무거운 쓰기 로드

작업

인사이트의 원인에 따라 다른 조치를 취할 것을 권장합니다.

주제

- [InnoDB 기록 목록이 줄어들 때까지 데이터베이스 종료와 관련된 작업을 시작하지 않음]
- [장기 실행 트랜잭션 확인 및 종료]
- [성능 개선 도우미를 사용하여 상위 호스트와 상위 사용자를 확인합니다.]

InnoDB 기록 목록이 줄어들 때까지 데이터베이스 종료와 관련된 작업을 시작하지 않음

InnoDB 기록 목록이 길면 데이터베이스 종료 속도가 느려지므로 데이터베이스 종료와 관련된 작업을 시작하기 전에 목록 크기를 줄이세요. 이러한 작업에는 메이저 버전 데이터베이스 업그레이드가 포함됩니다.

장기 실행 트랜잭션 확인 및 종료

`information_schema.innodb_trx` 쿼리를 통해 장기 실행 트랜잭션을 찾을 수 있습니다.

> **Note**
> 또한 읽기 전용 복제본에서 장기 실행 트랜잭션을 찾아보는 것도 잊지 마세요.

장기 실행 트랜잭션을 확인하는 방법

1. SQL 클라이언트에서 다음 쿼리를 실행합니다.

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
```

```
ORDER BY trx_started
```

2. 저장 프로시저 [mysql.rds_kill](#)을 사용하여 장기 실행 트랜잭션을 종료합니다.

성능 개선 도우미를 사용하여 상위 호스트와 상위 사용자를 확인합니다.

많은 수의 수정된 행이 즉시 커밋되도록 트랜잭션을 최적화합니다.

관련 지표

이 인사이트와 관련된 지표는 다음과 같습니다.

- `trx_rseg_history_len` - 이 카운터 지표는 `INFORMATION_SCHEMA.INNODB_METRICS` 테이블뿐만 아니라 성능 개선 도우미에서도 볼 수 있습니다. 자세한 내용은 MySQL 설명서의 InnoDB INFORMATION_SCHEMA metrics table을 참조하세요.
- `RollbackSegmentHistoryListLength` - 이 Amazon CloudWatch 지표는 삭제 표시 레코드로 커밋된 트랜잭션을 로깅하는 실행 취소 로그를 측정합니다. 이러한 레코드는 InnoDB 제거 작업에 의해 처리되도록 예약됩니다. `trx_rseg_history_len` 지표는 `RollbackSegmentHistoryListLength`과 동일한 값을 갖습니다.
- `PurgeBoundary` - InnoDB 제거가 허용되는 최대 트랜잭션 수입니다. 이 CloudWatch 지표가 장기간 변화하지 않으면 장기 실행 트랜잭션으로 인해 InnoDB 제거가 차단되었음을 나타내는 좋은 지표로 해석할 수 있습니다. 이를 조사하려면 Aurora MySQL DB 클러스터의 활성 트랜잭션을 확인합니다. 이 지표는 Aurora MySQL 버전 2.11 이상 및 버전 3.08 이상에서만 사용할 수 있습니다.
- `PurgeFinishedPoint` - InnoDB 제거가 수행되는 최대 트랜잭션 수입니다. 이 CloudWatch 지표를 사용하면 InnoDB 제거가 얼마나 빠르게 진행되고 있는지 확인할 수 있습니다. 이 지표는 Aurora MySQL 버전 2.11 이상 및 버전 3.08 이상에서만 사용할 수 있습니다.
- `TransactionAgeMaximum` - 가장 오랫동안 실행 중인 활성 트랜잭션의 연령입니다. 이 CloudWatch 지표는 Aurora MySQL 버전 3.08 이상에서 사용할 수 있습니다.
- `TruncateFinishedPoint` - 실행 취소 잘라내기가 수행될 때의 트랜잭션 수입니다. 이 CloudWatch 지표는 Aurora MySQL 버전 2.11 이상 및 버전 3.08 이상에서만 사용할 수 있습니다.

CloudWatch 지표에 대한 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 섹션을 참조하세요.

데이터베이스가 디스크에 임시 테이블을 만들고 있음

최근 온디스크 임시 테이블 사용량이 최대 *###*까지 크게 증가했습니다. 데이터베이스는 초당 약 *##*의 임시 테이블을 생성하고 있습니다. 이는 성능에 영향을 미치고 *db-instance*의 디스크 작업을 증가시킬 수 있습니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 이 문제의 잠재적 원인
- 작업
- 관련 지표

지원되는 엔진 버전

이 인사이트 정보는 Aurora MySQL의 모든 버전에서 지원됩니다.

컨텍스트

MySQL 서버가 쿼리를 처리하는 동안 내부 임시 테이블을 만들어야 하는 경우가 있습니다. Aurora MySQL은 내부 임시 테이블을 메모리에 보관할 수 있으며, 이 메모리는 TempTable 또는 MEMORY 스토리지 엔진에서 처리되거나 InnoDB에 의해 디스크에 저장될 수 있습니다. 자세한 내용은 MySQL 참조 설명서의 MySQL의 내부 임시 테이블 사용을 참조하세요.

이 문제의 잠재적 원인

온디스크 임시 테이블의 증가는 복잡한 쿼리를 사용함을 나타냅니다. 구성된 메모리가 메모리에 임시 테이블을 저장하기에 충분하지 않은 경우 Aurora MySQL이 디스크에 테이블을 생성합니다. 이는 성능에 영향을 미치고 디스크 작업을 증가시킬 수 있습니다.

작업

인사이트의 원인에 따라 다른 조치를 취할 것을 권장합니다.

- Aurora MySQL 버전 3의 경우 TempTable 스토리지 엔진을 사용하는 것이 좋습니다.
- 필요한 열만 선택하여 반환되는 데이터가 적도록 쿼리를 최적화합니다.

모든 statement 도구가 활성화되고 시간이 설정된 상태에서 성능 스키마를 켜면 SYS.statements_with_temp_tables를 쿼리하여 임시 테이블을 사용하는 쿼리 목록을 검색할 수 있습니다. 자세한 내용은 MySQL 설명서의 sys 스키마 사용 전제 조건을 참조하세요.

- 정렬 및 그룹화 작업과 관련된 열을 인덱싱하는 것을 고려해 보세요.
- BLOB 및 TEXT 열을 피하여 쿼리를 다시 작성합니다. 해당 열은 항상 디스크를 사용합니다.
- tmp_table_size 및 max_heap_table_size 데이터베이스 파라미터를 튜닝합니다.

  이러한 파라미터의 기본값은 16MiB입니다. 메모리 내 임시 테이블에 MEMORY 스토리지 엔진을 사용하는 경우 최대 크기는 tmp_table_size 또는 max_heap_table_size 값 중 더 작은 값으로 정의됩니다. 이 최대 크기에 도달하면 MySQL는 메모리 내 내부 임시 테이블을 InnoDB 온디스크 내부 임시 테이블로 자동 변환합니다. 자세한 내용은 Amazon RDS for MySQL 및 Amazon Aurora MySQL의 TempTable 스토리지 엔진 사용을 참조하세요.

> **Note**
>
> CREATE TABLE을 사용하여 MEMORY 테이블을 명시적으로 만들 때는 max_heap_table_size 변수만이 테이블의 최대 크기를 결정합니다. 온디스크 형식으로의 변환도 없습니다.

관련 지표

이 인사이트와 관련된 성능 개선 도우미 지표는 다음과 같습니다.

- Created_tmp_disk_tables
- Created_tmp_tables

자세한 내용은 MySQL 설명서에서 Created_tmp_disk_tables를 참조하세요.

Amazon Aurora MySQL용 병렬 쿼리

이 주제에서는 Amazon Aurora MySQL 호환 버전의 병렬 쿼리 성능 최적화에 대해 설명합니다. 이 기능은 특정 데이터 집약적인 쿼리를 위한 특수한 처리 경로를 사용하면서 Aurora 공유 스토리지 아키텍처를 활용합니다. 병렬 쿼리는 수만 개의 행이 포함된 테이블이 있는 Aurora MySQL DB 클러스터 및 완료되기까지 몇 분 또는 몇 시간이 걸리는 분석 쿼리에서 가장 잘 작동합니다.

주제

Aurora MySQL용 Parallel Query 개요

Aurora MySQL 병렬 쿼리는 데이터 집약적인 쿼리 처리에 수반되는 I/O 및 컴퓨팅의 일부를 병렬화하는 최적화입니다. 병렬화되는 작업은 스토리지로부터 행 검색, 열 값 추출, 어떤 행이 `WHERE` 절 및 JOIN 절의 조건과 일치하는지 판단을 포함합니다. 이 데이터 집약적인 작업은 Aurora 분산 스토리지 계층의 여러 노드에 위임됩니다(데이터베이스 최적화 용어로는 푸시 다운됩니다). 병렬 쿼리가 없으면, 각 쿼리가 스캔한 모든 데이터를 Aurora MySQL 클러스터(헤드 노드) 내의 단일 노드로 가져오고 거기에서 모든 쿼리 처리를 수행합니다.

> **Tip**
>
> PostgreSQL 데이터베이스 엔진에도 '병렬 쿼리'라는 기능이 있습니다. 해당 기능은 Aurora 병렬 쿼리와 관련이 없습니다.

병렬 쿼리 기능을 설정하면 Aurora MySQL 엔진이 힌트 또는 테이블 속성과 같은 SQL 변경의 필요 없이 쿼리가 혜택을 얻을 수 있는 경우를 자동으로 결정합니다. 다음 단원에는서 병렬 쿼리가 쿼리에 적용되는 경우에 대한 설명을 확인할 수 있습니다. 또한 병렬 쿼리가 가장 많은 혜택을 제공하는 경우에 적용되게 하는 방법도 확인할 수 있습니다.

> **Note**
> 병렬 쿼리 최적화는 완료하는 데 몇 분 또는 몇 시간이 걸리는 장기 실행 쿼리에 가장 큰 이점을 제공합니다. Aurora MySQL은 일반적으로 비용이 적게 드는 쿼리에 대해 병렬 쿼리 최적화를 수행하지 않습니다. 또한 쿼리 캐싱, 버퍼 풀 캐싱, 인덱스 조회를 비롯한 다른 최적화 기술이 더 적합한 경우에도 일반적으로 병렬 쿼리 최적화를 수행하지 않습니다. 병렬 쿼리가 사용될 것으로 예상될 때 사용되고 있지 않은 경우 어떤 문이 Aurora MySQL의 병렬 쿼리를 사용하는지 확인 단원을 참조하세요.

주제

- 이점
- 아키텍처
- 사전 조건
- 제한 사항
- 병렬 쿼리를 사용할 경우 I/O 비용

이점

병렬 쿼리를 사용하면 Aurora MySQL 테이블에 대해 데이터 집약적인 분석 쿼리를 실행할 수 있으며, 대부분의 경우 쿼리 처리를 위한 전통적인 작업 분할에 비해 획기적인 성능 향상이 나타날 수 있습니다.

병렬 쿼리의 이점은 다음과 같습니다.

- 여러 스토리지 노드에 걸친 물리적 읽기 요청을 병렬화하여 I/O 성능 개선
- 네트워크 트래픽 감소. Aurora는 전체 데이터 페이지를 스토리지 노드로부터 헤드 노드로 전송한 다음 그 후에 불필요한 행과 열을 필터링하지 않습니다. 대신, Aurora은 결과 집합에 필요한 열 값만 포함된 간소화된 튜플을 전송합니다.
- 푸시 다운 기능 처리, 행 필터링 및 `WHERE` 절에 대한 열 프로젝션으로 인한 헤드 노드에 대한 CPU 사용량의 감소.
- 버퍼 풀에서의 메모리 압력 감소. 병렬 쿼리에 의해 처리된 페이지는 버퍼 풀에 추가되지 않으므로 데이터 집약적인 스캔 중 버퍼 풀에서 자주 사용되는 데이터가 제거될 가능성이 감소됩니다.
- 기존 데이터에 대한 장기 실행 분석 쿼리 수행이 유용해진 덕분에 추출, 변환, 로드(ETL) 파이프라인에서 데이터 중복의 잠재적 감소.

아키텍처

병렬 쿼리 기능은 Aurora MySQL의 주요 아키텍처 원칙을 사용합니다. 스토리지 하위 시스템으로부터 데이터베이스 엔진의 결합 해제 및 통신 프로토콜을 간소화하여 네트워크 트래픽을 감소합니다. Aurora MySQL은 이러한 기법을 사용하여 다시 실행 로그 처리와 같은 쓰기 집약적인 작업의 속도를 높입니다. 병렬 쿼리는 읽기 작업에도 동일한 원칙을 적용합니다.

> **Note**
>
> Aurora MySQL 병렬 쿼리의 아키텍처는 다른 데이터베이스 시스템에서 이름이 유사한 기능의 아키텍처와는 다릅니다. Aurora MySQL 병렬 쿼리는 SMP(대칭적 다중 처리)를 포함하지 않으며 따라서 데이터베이스 서버의 CPU 용량에 의존하지 않습니다. 병렬 처리는 쿼리 조정자 역할을 하는 Aurora MySQL 서버와는 독립적인 스토리지 계층에서 일어납니다.

기본적으로, 병렬 쿼리를 사용하지 않을 경우 Aurora 쿼리에 대한 처리는 원시 데이터를 Aurora 클러스터 내 단일 노드(헤드 노드)로 전송합니다. 그런 다음 Aurora는 해당 단일 노드의 단일 스레드에서 해당 쿼리에 대해 추가되는 모든 처리를 수행합니다. 병렬 쿼리를 사용할 경우, 이러한 I/O 집약적이고 CPU 집약적인 작업의 대부분이 스토리지 계층의 노드로 위임됩니다. 행은 이미 필터링되고 열 값은 이미 추출되어 전송된 상태로, 결과 집합의 간소화된 행만 다시 헤드 노드로 전송됩니다. 성능 혜택은 네트워크 트래픽의 감소, 헤드 노드에서 CPU 사용량의 감소, 및 스토리지 노드 전체에서 I/O 병렬화로부터 비롯됩니다. 병렬 I/O, 필터링 및 프로젝션의 양은 쿼리를 실행하는 Aurora 클러스터의 DB 인스턴스 수와 무관합니다.

사전 조건

병렬 쿼리의 모든 기능을 사용하려면 버전 2.09 이상을 실행하는 Aurora MySQL DB 클러스터가 필요합니다. 병렬 쿼리와 함께 사용하려는 클러스터가 이미 있는 경우 호환 가능한 버전으로 업그레이드하고 나중에 병렬 쿼리를 설정할 수 있습니다. 이 경우 해당 최신 버전에서는 구성 설정 이름 및 기본값이 다르기 때문에 병렬 쿼리를 위한 업그레이드 고려 사항의 업그레이드 절차를 따라야 합니다.

클러스터의 DB 인스턴스는 `db.r*` 인스턴스 클래스를 사용해야 합니다.

클러스터에 대해 해시 조인 최적화가 설정되어 있는지 확인합니다. 자세한 방법은 병렬 쿼리 클러스터에 대한 해시 조인 설정 단원을 참조하십시오.

`aurora_parallel_query` 및 `aurora_disable_hash_join`과 같은 파라미터를 사용자 지정하려면 클러스터와 함께 사용하는 사용자 지정 파라미터 그룹이 있어야 합니다. DB 파라미터 그룹을 사용하여 각 DB 인스턴스에 대해 이러한 파라미터를 개별적으로 지정할 수 있습니다. 그러나 DB 클러스터

파라미터 그룹에서 지정하는 것이 좋습니다. 이렇게 하면 클러스터의 모든 DB 인스턴스가 이러한 파라미터에 대해 동일한 설정을 상속합니다.

제한 사항

다음 제한 사항이 병렬 쿼리 기능에 적용됩니다.

- 병렬 쿼리는 Aurora I/O-Optimized DB 클러스터 스토리지 구성에서 지원되지 않습니다.
- db.t2 또는 db.t3 인스턴스 클래스에는 병렬 쿼리를 사용할 수 없습니다. 이 제한은 `aurora_pq_force` 세션 변수를 사용하여 병렬 쿼리를 요청하는 경우에도 적용됩니다.
- 병렬 쿼리는 `COMPRESSED` 또는 `REDUNDANT` 행 형식을 사용하는 테이블에는 적용되지 않습니다. 병렬 쿼리에 사용하려는 테이블에는 `COMPACT` 또는 `DYNAMIC` 행 형식을 사용합니다.
- Aurora는 비용 기반 알고리즘을 사용하여 각 SQL 문에 대해 병렬 쿼리 메커니즘을 사용할지 여부를 결정합니다. 명령문에서 특정 SQL 구문을 사용하면 병렬 쿼리가 방지되거나 해당 명령문에 대해 병렬 쿼리가 실행되지 않을 수 있습니다. SQL 구문과 병렬 쿼리의 호환성에 대한 자세한 내용은 Aurora MySQL의 병렬 쿼리를 위한 SQL 구성 단원을 참조하세요.
- 각 Aurora DB 인스턴스는 한 번에 일정한 수의 병렬 쿼리 세션만 실행할 수 있습니다. 쿼리에 하위 쿼리, 조인 또는 `UNION` 연산자 같은 병렬 쿼리를 사용하는 여러 부분이 있는 경우, 그러한 단계가 순차적으로 실행됩니다. 구문은 한 시점에 단일 병렬 쿼리 세션으로만 계산됩니다. 병렬 쿼리 상태 변수를 사용하여 활성 세션의 수를 모니터링할 수 있습니다. 상태 변수 `Aurora_pq_max_concurrent_requests`를 쿼리하여 주어진 DB 인스턴스의 동시 세션에 대한 제한을 확인할 수 있습니다.
- 병렬 쿼리는 Aurora가 지원되는 모든 AWS 리전에서 사용할 수 있습니다. 대부분의 AWS 리전에서 병렬 쿼리를 사용하는 데 필요한 최소 Aurora MySQL 버전은 2.09입니다.
- 병렬 쿼리는 데이터 집약적인 쿼리의 성능을 향상하도록 설계되었습니다. 경량 쿼리용으로 설계되지 않았습니다.
- SELECT 문, 특히 데이터 집약적인 문에는 리더 노드를 사용하는 것이 좋습니다.

병렬 쿼리를 사용할 경우 I/O 비용

Aurora MySQL 클러스터가 병렬 쿼리를 사용하는 경우 `VolumeReadIOPS` 값이 증가할 수 있습니다. 병렬 쿼리는 버퍼 풀을 사용하지 않습니다. 따라서 쿼리는 빠르지만 이렇게 최적화된 프로세싱은 읽기 작업 및 관련 비용을 증가시킬 수 있습니다.

쿼리에 대한 병렬 쿼리 I/O 비용은 스토리지 계층에서 측정되며 병렬 쿼리를 켠 상태에서 같거나 더 커집니다. 이를 통해 얻을 수 있는 이점은 쿼리 성능이 향상된다는 것입니다. 병렬 쿼리를 사용할 경우 I/O 비용이 높아질 수 있는 두 가지 이유가 있습니다.

- 테이블의 일부 데이터가 버퍼 풀에 있더라도 병렬 쿼리는 스토리지 계층에서 모든 데이터를 스캔해야 하므로 I/O 비용이 발생합니다.
- 병렬 쿼리를 실행해도 버퍼 풀이 워밍업되지 않습니다. 결과적으로 동일한 병렬 쿼리를 연속으로 실행하면 전체 I/O 비용이 발생합니다.

Aurora MySQL에서 병렬 쿼리 DB 클러스터 생성

병렬 쿼리를 사용하여 Aurora MySQL 클러스터를 생성하려면, 새 인스턴스를 추가하거나 다른 Aurora MySQL 클러스터에서 수행하는 것과 동일한 AWS Management Console 및 AWS CLI 기법을 사용하는 다른 관리 작업을 수행합니다. 병렬 쿼리에서 작동하는 새 클러스터를 생성할 수 있습니다. 또한 MySQL과 호환되는 Aurora DB 클러스터의 스냅샷에서 복원하여 병렬 쿼리에서 작동하는 DB 클러스터도 생성할 수 있습니다. 새 Aurora MySQL 클러스터를 생성하는 절차에 익숙하지 않은 경우, Amazon Aurora DB 클러스터 생성에서 배경 정보를 확인할 수 있습니다.

Aurora MySQL 엔진 버전을 선택할 때는 최신 버전을 선택하는 것이 좋습니다. 현재 사용 가능한 모든 Aurora MySQL 버전이 병렬 쿼리를 지원합니다. 최신 버전을 사용하면 병렬 쿼리를 설정 및 해제하거나 기존 클러스터에서 병렬 쿼리를 사용할 수 있는 유연성이 높아집니다.

새 클러스터를 생성하든 스냅샷에서 복원하든, 새 DB 인스턴스를 추가하기 위해 다른 Aurora MySQL 클러스터에서 수행하는 것과 동일한 기법을 사용합니다.

Amazon RDS 콘솔 또는 AWS CLI를 사용하여 병렬 쿼리 클러스터를 생성할 수 있습니다.

목차

콘솔을 사용하여 병렬 쿼리 클러스터 생성

다음 설명에 따라 콘솔을 사용하여 새 병렬 쿼리 클러스터를 생성할 수 있습니다.

AWS Management Console을 사용하여 병렬 쿼리 클러스터를 생성하려면

1. AWS Management Console의 일반 Amazon Aurora DB 클러스터 생성 절차를 따르십시오.
2. 엔진 유형으로 Aurora MySQL을 선택합니다.
3. 추가 구성에서 DB 클러스터 파라미터 그룹에 대해 생성한 파라미터 그룹을 선택합니다. Aurora MySQL 2.09 이상에서는 이러한 사용자 지정 파라미터 그룹을 사용해야 합니

다. DB 클러스터 파라미터 그룹에서 파라미터 설정 aurora_parallel_query=ON 및 aurora_disable_hash_join=OFF를 지정합니다. 이렇게 하면 클러스터에 대한 병렬 쿼리가 설정되고 병렬 쿼리와 함께 작동하는 해시 조인 최적화가 설정됩니다.

새 클러스터가 병렬 쿼리를 사용할 수 있는지 확인하려면

1. 앞에 나온 방법을 사용하여 클러스터를 생성합니다.
2. (Aurora MySQL 버전 2 또는 3의 경우) aurora_parallel_query 구성 설정이 참인지 확인합니다.

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (Aurora MySQL 버전 2의 경우) aurora_disable_hash_join 설정이 거짓인지 확인합니다.

```
mysql> select @@aurora_disable_hash_join;
+----------------------------+
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. 일부 대형 테이블과 데이터 집약적 쿼리의 경우 쿼리 계획을 검토하여 일부 쿼리에 병렬 쿼리 최적화가 사용되고 있는지 확인합니다. 이 작업을 수행하려면 의 절차를 수행합니다[어떤 문이 Aurora MySQL의 병렬 쿼리를 사용하는지 확인](#)

CLI를 사용하여 병렬 쿼리 클러스터 생성

다음 설명에 따라 CLI를 사용하여 새 병렬 쿼리 클러스터를 생성할 수 있습니다.

AWS CLI을 사용하여 병렬 쿼리 클러스터를 생성하려면

1. (선택 사항) 병렬 쿼리 클러스터와 호환되는 Aurora MySQL 버전을 확인합니다. 이를 위해 describe-db-engine-versions 명령을 사용하고 SupportsParallelQuery 필드 값을 확인합니다. 관련 예제는 [병렬 쿼리를 위한 Aurora MySQL 버전 호환성 확인](#) 섹션을 참조하세요

2. (선택 사항) aurora_parallel_query=ON 및 aurora_disable_hash_join=OFF 설정으로 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 다음과 같은 명령을 사용합니다.

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-mysql8.0 --db-cluster-parameter-group-name pq-enabled-80-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-enabled-80-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-enabled-80-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-reboot
```

이 단계를 수행하는 경우 이후 --db-cluster-parameter-group-name *my_cluster_parameter_group* 문에서 create-db-cluster 옵션을 지정합니다. 고유한 해당 파라미터 그룹 이름으로 대치합니다. 이 단계를 생략하는 경우 Aurora MySQL에서 병렬 쿼리 설정 및 해제에 설명된 대로 파라미터 그룹을 생성하고 나중에 클러스터에 연결합니다.

3. AWS CLI의 일반 Amazon Aurora DB 클러스터 생성 절차를 따르십시오.

4. 다음 옵션 세트를 지정하십시오.

   - --engine 옵션의 경우 aurora-mysql를 사용합니다. 이러한 값은 MySQL 5.7 또는 8.0과 호환되는 병렬 쿼리 클러스터를 생성합니다.
   - --db-cluster-parameter-group-name 옵션에 생성한 DB 클러스터 파라미터 그룹의 이름을 지정하고 파라미터 값 aurora_parallel_query=ON을 지정합니다. 이 옵션을 생략하는 경우 기본 파라미터 그룹을 사용하여 클러스터를 생성한 다음 나중에 해당 사용자 지정 파라미터 그룹을 사용하도록 수정할 수 있습니다.
   - --engine-version 옵션에 병렬 쿼리와 호환되는 Aurora MySQL 버전을 사용합니다. 필요한 경우 Aurora MySQL에서 병렬 쿼리 최적화의 절차를 사용하여 버전 목록을 가져옵니다.

   다음 코드 예제에서는 작업 방법을 보여줍니다. *$CLUSTER_ID*와 같은 각 환경 변수를 고유한 해당 값으로 대치합니다. 이 예제에서는 마스터 사용자 암호를 생성하고 이를 Secrets Manager에서 관리하는 --manage-master-user-password 옵션도 지정합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 단원을 참조하십시오. 또는 --master-password 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

   ```
   aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
   ```

```
  --engine aurora-mysql --engine-version 8.0.mysql_aurora.3.04.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. 생성하거나 복원한 클러스터에서 병렬 쿼리 기능을 사용할 수 있는지 확인합니다.

   aurora_parallel_query 구성 설정이 존재하는지 확인합니다. 이 설정의 값이 1이면 병렬 쿼리를 사용할 수 있습니다. 이 설정의 값이 0이면 먼저 1로 설정해야 병렬 쿼리를 사용할 수 있습니다. 어느 경우이든 클러스터에서 병렬 쿼리를 수행할 수 있습니다.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

AWS CLI을 사용하여 스냅샷을 병렬 쿼리 클러스터에 복원하려면

1. 병렬 쿼리 클러스터와 호환되는 Aurora MySQL 버전을 확인합니다. 이를 위해 describe-db-engine-versions 명령을 사용하고 SupportsParallelQuery 필드 값을 확인합니다. 관련 예제는 병렬 쿼리를 위한 Aurora MySQL 버전 호환성 확인 섹션을 참조하세요 복원된 클러스터에 사용할 버전을 결정합니다.
2. Aurora MySQL 호환 클러스터 스냅샷을 찾습니다.
3. AWS CLI의 일반 DB 클러스터 스냅샷에서 복원 절차를 따르십시오.

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. 생성하거나 복원한 클러스터에서 병렬 쿼리 기능을 사용할 수 있는지 확인합니다. CLI를 사용하여 병렬 쿼리 클러스터 생성에 나온 것과 동일한 확인 절차를 사용합니다.

Aurora MySQL에서 병렬 쿼리 설정 및 해제

병렬 쿼리가 설정되면 Aurora MySQL이 각 쿼리의 런타임 시 병렬 쿼리를 사용할지 여부를 결정합니다. 조인, 통합, 하위 쿼리 등의 경우에는 Aurora MySQL이 각 쿼리 블록의 실행 시간에 병렬 쿼리를 사용할지 여부를 결정합니다. 자세한 내용은 어떤 문이 Aurora MySQL의 병렬 쿼리를 사용하는지 확인 및 Aurora MySQL의 병렬 쿼리를 위한 SQL 구성 단원을 참조하십시오.

aurora_parallel_query 옵션을 사용하여 DB 인스턴스에 대하여 전역 수준과 세션 수준에서 모두 동적으로 병렬 쿼리를 설정 및 해제할 수 있습니다. DB 클러스터 그룹의 `aurora_parallel_query` 설정을 변경하여 병렬 쿼리를 기본적으로 설정하거나 해제할 수 있습니다.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

세션 수준에서 `aurora_parallel_query` 파라미터를 전환하려면 표준 방법을 사용하여 클라이언트 구성 설정을 변경합니다. 예를 들어 `mysql` 명령줄을 통해 또는 JDBC 또는 ODBC 애플리케이션 내에서 이렇게 할 수 있습니다. 표준 MySQL 클라이언트에서는 명령이 `set session aurora_parallel_query = {'ON'/'OFF'}`입니다. 또한 JDBC 구성 또는 애플리케이션 코드에 세션 수준 파라미터를 추가하여 동적으로 병렬 쿼리를 설정하거나 해제할 수 있습니다.

특정 DB 인스턴스 또는 전체 클러스터에 대해 `aurora_parallel_query` 파라미터 설정을 영구적으로 변경할 수 있습니다. DB 파라미터 그룹에서 파라미터 값을 지정하면 해당 값은 클러스터의 특정 DB 인스턴스에만 적용됩니다. DB 클러스터 파라미터 그룹에서 파라미터 값을 지정하면 클러스터의 모든 DB 인스턴스가 동일한 설정을 상속합니다. `aurora_parallel_query` 파라미터를 전환하려면 Amazon Aurora의 파라미터 그룹에 설명된 대로 파라미터 그룹으로 작업하기 위한 기술을 사용합니다. 다음 단계를 따릅니다.

1. 사용자 지정 클러스터 파라미터 그룹(권장) 또는 사용자 지정 DB 파라미터 그룹을 생성합니다.
2. 이 파라미터 그룹에서 원하는 값으로 `parallel_query`를 업데이트합니다.
3. DB 클러스터 파라미터 그룹을 생성했는지 아니면 DB 파라미터 그룹을 생성했는지에 따라 파라미터 그룹을 Aurora 클러스터에 연결하거나 병렬 쿼리 기능을 사용할 특정 DB 인스턴스에 연결합니다.

> **Tip**
>
> aurora_parallel_query는 동적 파라미터이므로 이 설정을 변경한 후에 클러스터를 다시 시작할 필요가 없습니다. 그러나 옵션을 토글하기 전에 병렬 쿼리를 사용하고 있던 모든 연결은 연결이 닫히거나 인스턴스가 재부팅될 때까지 계속됩니다.

ModifyDBClusterParameterGroup 또는 ModifyDBParameterGroup API 작업이나 AWS Management Console을 사용하여 병렬 쿼리 파라미터를 수정할 수 있습니다.

병렬 쿼리 클러스터의 해시 조인을 설정하고, Amazon RDS 콘솔 또는 AWS CLI를 사용하여 병렬 쿼리를 설정 및 해제하고, 병렬 쿼리 최적화 프로그램을 재정의할 수 있습니다.

목차

병렬 쿼리 클러스터에 대한 해시 조인 설정

병렬 쿼리는 해시 조인 최적화에서 이익을 얻는 리소스 집약적인 쿼리 유형에 일반적으로 사용됩니다. 따라서 병렬 쿼리를 사용하려는 클러스터에 대해 해시 조인을 설정하는 것이 좋습니다. 해시 조인을 효과적으로 사용하는 방법에 대한 자세한 내용은 해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화 섹션을 참조하세요.

콘솔을 사용하여 병렬 쿼리 설정 및 해제

파라미터 그룹으로 작업하여 DB 인스턴스 수준 또는 DB 클러스터 수준에서 병렬 쿼리를 설정하거나 해제할 수 있습니다.

AWS Management Console을 사용하여 DB 클러스터에 대한 병렬 쿼리를 설정하거나 해제하려면

1. Amazon Aurora의 파라미터 그룹에 설명된 대로 사용자 지정 파라미터 그룹을 생성합니다.
2. aurora_parallel_query를 1(활성화됨) 또는 0(비활성화됨)으로 업데이트합니다. 병렬 쿼리 기능을 사용할 수 있는 클러스터에서는 aurora_parallel_query가 기본적으로 해제되어 있습니다.

3. 사용자 지정 클러스터 파라미터 그룹을 사용하는 경우 병렬 쿼리 기능을 사용할 Aurora DB 클러스터에 해당 그룹을 연결합니다. 사용자 지정 DB 파라미터 그룹을 사용하는 경우 클러스터에 있는 하나 이상의 DB 인스턴스에 해당 그룹을 연결합니다. 클러스터 파라미터 그룹을 사용하는 것이 좋습니다. 이렇게 하면 클러스터의 모든 DB 인스턴스가 병렬 쿼리 및 해시 조인과 같은 관련 기능에 대해 동일한 설정을 갖게 됩니다.

CLI를 사용하여 병렬 쿼리 설정 및 해제

`modify-db-cluster-parameter-group` 또는 `modify-db-parameter-group` 명령을 사용하여 병렬 쿼리 파라미터를 수정할 수 있습니다. `aurora_parallel_query`의 값을 DB 클러스터 파라미터 그룹을 통해 지정하는지 아니면 DB 파라미터 그룹을 통해 지정하는지에 따라 적절한 명령을 선택합니다.

CLI를 사용하여 DB 클러스터에 대한 병렬 쿼리를 설정하거나 해제하려면

- `modify-db-cluster-parameter-group` 명령을 사용하여 병렬 쿼리 파라미터를 수정합니다. 다음과 같은 명령을 사용합니다. 사용자 지정 파라미터 그룹의 해당 이름으로 대치합니다. `ON` 옵션의 `OFF` 부분을 `ParameterValue` 또는 `--parameters`로 대치합니다.

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

또한 세션 수준에서도(예를 들어 `mysql` 명령줄을 통해 혹은 JDBC 또는 ODBC 애플리케이션 내에서) 병렬 쿼리를 설정하거나 해제할 수 있습니다. 그렇게 하려면, 표준 메서드를 사용하여 클라이언트 구성 설정을 변경합니다. 예를 들어, 표준 MySQL 클라이언트에서는 Aurora MySQL의 경우 명령이 `set session aurora_parallel_query = {'ON'/'OFF'}`이고,

또한 JDBC 구성 또는 애플리케이션 코드에 세션 수준 파라미터를 추가하여 동적으로 병렬 쿼리를 설정하거나 해제할 수 있습니다.

병렬 쿼리 옵티마이저 재정의

aurora_pq_force 세션 변수를 사용하여 병렬 쿼리 옵티마이저 프로그램을 재정의하고 모든 쿼리에 대해 병렬 쿼리를 요청할 수 있습니다. 이 작업은 테스트 목적으로만 하는 것이 좋습니다. 다음 예제는 세션에서 aurora_pq_force를 사용하는 방법을 보여줍니다.

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

재정의를 해제하려면 다음을 수행합니다.

```
set SESSION aurora_pq_force = OFF;
```

Aurora MySQL에서 병렬 쿼리 최적화

병렬 쿼리에 대해 DB 클러스터를 최적화하려면 병렬 쿼리의 이점을 누릴 수 있는 DB 클러스터와 병렬 쿼리 업그레이드 여부를 고려해야 합니다. 그런 다음 워크로드를 조정하고 병렬 쿼리를 위한 스키마 객체를 생성합니다.

목차

병렬 쿼리 클러스터 계획

병렬 쿼리가 설정된 DB 클러스터를 계획하려면 몇 가지 선택이 필요합니다. 여기에는 설정 단계(전체 Aurora MySQL 클러스터 생성 또는 복원)를 수행하고 DB 클러스터에서 병렬 쿼리를 설정할 범위를 결정하는 작업이 포함됩니다.

계획 과정에서 다음과 같은 사항을 고려해야 합니다.

- MySQL 5.7과 호환되는 Aurora MySQL을 사용하는 경우 프로비저닝된 클러스터를 생성해야 합니다. 그런 다음 `aurora_parallel_query` 파라미터를 사용하여 병렬 쿼리를 설정합니다.

  기존 Aurora MySQL 클러스터가 있는 경우 병렬 쿼리를 사용하기 위해 새 클러스터를 생성할 필요가 없습니다. 클러스터 또는 클러스터의 특정 DB 인스턴스를 `aurora_parallel_query` 파라미터가 설정된 파라미터 그룹에 연결할 수 있습니다. 이렇게 하면 병렬 쿼리와 함께 사용할 관련 데이터를 설정하는 데 드는 시간과 노력을 줄일 수 있습니다.
- 테이블에 액세스할 때 병렬 쿼리를 사용할 수 있도록 재구성해야 하는 대형 테이블이 있는지 확인하고 조치를 취합니다. 병렬 쿼리가 유용한 일부 대형 테이블의 경우 새 버전을 만들어야 할 수도 있습니다. 예를 들어 전체 텍스트 검색 인덱스를 제거해야 할 수 있습니다. 세부 정보는 병렬 쿼리를 활용하기 위해 스키마 객체 생성을 참조하세요.

병렬 쿼리를 위한 Aurora MySQL 버전 호환성 확인

병렬 쿼리 클러스터와 호환되는 Aurora MySQL 버전을 확인하려면 `describe-db-engine-versions` AWS CLI 명령을 사용하고 `SupportsParallelQuery` 필드의 값을 확인합니다. 다음 코드 예제는 지정된 AWS 리전에서 병렬 쿼리 클러스터에 사용할 수 있는 조합을 확인하는 방법을 보여줍니다. 한 줄에 전체 `--query` 파라미터 문자열을 지정해야 합니다.

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

위의 명령은 다음과 비슷한 출력을 생성합니다. 출력은 지정된 AWS 리전에서 사용 가능한 Aurora MySQL 버전에 따라 달라질 수 있습니다.

```
5.7.mysql_aurora.2.11.1
5.7.mysql_aurora.2.11.2
5.7.mysql_aurora.2.11.3
5.7.mysql_aurora.2.11.4
5.7.mysql_aurora.2.11.5
5.7.mysql_aurora.2.11.6
5.7.mysql_aurora.2.12.0
5.7.mysql_aurora.2.12.1
5.7.mysql_aurora.2.12.2
5.7.mysql_aurora.2.12.3
5.7.mysql_aurora.2.12.4
8.0.mysql_aurora.3.04.0
8.0.mysql_aurora.3.04.1
8.0.mysql_aurora.3.04.2
8.0.mysql_aurora.3.04.3
```

```
8.0.mysql_aurora.3.05.2
8.0.mysql_aurora.3.06.0
8.0.mysql_aurora.3.06.1
8.0.mysql_aurora.3.07.0
8.0.mysql_aurora.3.07.1
```

클러스터에서 병렬 쿼리를 사용하기 시작한 후 성능을 모니터링하고 병렬 쿼리 사용에 대한 장애 요소를 제거할 수 있습니다. 해당 지침은 병렬 쿼리를 위한 성능 튜닝 단원을 참조하십시오.

병렬 쿼리를 위한 업그레이드 고려 사항

병렬 쿼리 클러스터를 업그레이드할 때 원래 버전과 대상 버전에 따라 병렬 쿼리가 최적화할 수 있는 쿼리 유형을 개선할 수 있습니다. 병렬 쿼리에 특수 엔진 모드 파라미터를 지정하지 않아도 될 수 있습니다. 다음 섹션에서는 병렬 쿼리가 설정된 클러스터를 업그레이드하는 경우 고려 사항에 대해 설명합니다.

Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드

여러 SQL 문, 절 및 데이터 유형에는 Aurora MySQL 버전 3부터 새로운 병렬 쿼리 지원 또는 향상된 병렬 쿼리 지원이 있습니다. 버전 3 이전 릴리스에서 업그레이드하는 경우 추가 쿼리가 병렬 쿼리 최적화를 통해 이점을 얻을 수 있는지 확인합니다. 이러한 병렬 쿼리 개선 사항에 대한 자세한 내용은 열 데이터 유형, 분할된 테이블 및 집계 함수, GROUP BY 절 및 HAVING 절 섹션을 참조하세요.

Aurora MySQL 2.08 이하에서 병렬 쿼리 클러스터를 업그레이드하는 경우 병렬 쿼리를 설정하는 방법의 변경 사항에 대해서도 알아봅니다. 이렇게 하려면 Aurora MySQL 2.09 이상으로 업그레이드 섹션을 참조하세요.

Aurora MySQL 버전 3에서 해시 조인 최적화는 기본적으로 설정되어 있습니다. 이전 버전의 `aurora_disable_hash_join` 구성 옵션은 사용되지 않습니다.

Aurora MySQL 2.09 이상으로 업그레이드

Aurora MySQL 버전 2.09 이상에서는 병렬 쿼리가 프로비저닝된 클러스터에 대해 작동하며 `parallelquery` 엔진 모드 파라미터가 필요하지 않습니다. 따라서 이러한 버전에서 병렬 쿼리를 사용하기 위해 새 클러스터를 생성하거나 기존 스냅샷에서 복원할 필요가 없습니다. Aurora MySQL DB 클러스터의 부 버전 또는 패치 수준 업그레이드에 설명된 업그레이드 절차를 사용하여 클러스터를 이러한 버전으로 업그레이드할 수 있습니다. 병렬 쿼리 클러스터인지 또는 프로비저닝된 클러스터인지에 관계없이 이전 클러스터를 업그레이드할 수 있습니다. 엔진 버전 메뉴에서 선택 항목 수를 줄이려면 병렬 쿼리 기능을 지원하는 버전 표시를 선택하여 해당 메뉴의 항목을 필터링할 수 있습니다. 그런 다음 Aurora MySQL 2.09 이상을 선택합니다.

이전 병렬 쿼리 클러스터를 Aurora MySQL 2.09 이상으로 업그레이드한 후 업그레이드된 클러스터에서 병렬 쿼리를 설정합니다. 이러한 버전에서는 병렬 쿼리가 기본적으로 해제되어 있으며 이를 활성화하는 절차가 다릅니다. 해시 조인 최적화도 기본적으로 해제되어 있으므로 별도로 설정해야 합니다. 따라서 업그레이드 후 이러한 설정을 다시 설정하도록 합니다. 관련 지침은 Aurora MySQL에서 병렬 쿼리 설정 및 해제 및 병렬 쿼리 클러스터에 대한 해시 조인 설정 단원을 참조하세요.

특히 `aurora_parallel_query=ON` 및 `aurora_disable_hash_join=OFF` 대신 `aurora_pq_supported` 및 `aurora_pq` 구성 파라미터를 사용하여 병렬 쿼리를 설정합니다. `aurora_pq_supported` 및 `aurora_pq` 파라미터는 최신 Aurora MySQL 버전에서는 더 이상 사용되지 않습니다.

업그레이드된 클러스터에서 `EngineMode` 속성은 `provisioned` 대신 `parallelquery` 값을 갖습니다. 지정된 엔진 버전에 대해 병렬 쿼리를 사용할 수 있는지 여부를 확인하려면 이제 `SupportsParallelQuery` `describe-db-engine-versions` 명령의 출력에서 AWS CLI 필드 값을 확인합니다. 이전 Aurora MySQL 버전에서는 `parallelquery` 목록에 `SupportedEngineModes`가 있는지 확인했습니다.

Aurora MySQL 버전 2.09 이상으로 업그레이드한 후 다음 기능을 이용할 수 있습니다. 이러한 기능은 이전 Aurora MySQL 버전을 실행하는 병렬 쿼리 클러스터에는 사용할 수 없습니다.

- Performance Insights. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.
- 역추적. 자세한 내용은 Aurora DB 클러스터 역추적 섹션을 참조하세요.
- 클러스터 중지 및 시작. 자세한 내용은 Amazon Aurora DB 클러스터 중지 및 시작 섹션을 참조하세요.

병렬 쿼리를 위한 성능 튜닝

병렬 쿼리로 워크로드의 성능을 관리하려면, 이 최적화가 가장 도움이 되는 쿼리에 병렬 쿼리가 사용되어야 합니다.

그렇게 하기 위해 다음을 수행할 수 있습니다.

- 최대 규모의 테이블이 병렬 쿼리와 호환되는지 확인합니다. 이러한 테이블에 대한 쿼리에 병렬 쿼리 최적화가 활용될 수 있도록 테이블 속성을 변경하거나 일부 테이블을 다시 생성할 수 있습니다. 자세한 방법은 병렬 쿼리를 활용하기 위해 스키마 객체 생성 단원을 참조하십시오.
- 어떤 쿼리가 병렬 쿼리를 사용하는지 모니터링합니다. 자세한 방법은 Aurora MySQL용 병렬 쿼리 모니터링 단원을 참조하십시오.

- 병렬 쿼리가 데이터 집약적인 장기 실행 쿼리에서 사용되는지 그리고 적절한 수준의 워크로드 동시성에서 사용되는지 확인합니다. 자세한 방법은 어떤 문이 Aurora MySQL의 병렬 쿼리를 사용하는지 확인 단원을 참조하십시오.
- SQL 코드를 세부 조정하여 예상한 쿼리에 병렬 쿼리가 적용되도록 설정합니다. 자세한 방법은 Aurora MySQL의 병렬 쿼리를 위한 SQL 구성 단원을 참조하십시오.

병렬 쿼리를 활용하기 위해 스키마 객체 생성

병렬 쿼리에 사용할 테이블을 생성하거나 수정하기 전에 사전 조건 및 제한 사항에 설명된 요구 사항을 숙지해야 합니다.

병렬 쿼리는 테이블이 `ROW_FORMAT=Compact` 또는 `ROW_FORMAT=Dynamic` 설정을 사용해야 하기 때문에 Aurora 구성 설정에 `INNODB_FILE_FORMAT` 구성 옵션에 대한 변경 사항이 있는지 확인합니다. `SHOW TABLE STATUS` 문을 실행하여 데이터베이스의 모든 테이블에 대한 행 형식을 확인합니다.

스키마를 변경하여 병렬 쿼리가 더 많은 테이블에서 작동하도록 설정하기 전에 테스트해야 합니다. 테스트에서 병렬 쿼리로 인해 해당 테이블의 성능이 실제로 향상되는지 확인해야 합니다. 또한, 병렬 쿼리를 위한 스키마 요구 사항이 원하는 목표와 다른 식으로 양립될 수 있는지 확인합니다.

예를 들어, `ROW_FORMAT=Compressed`에서 `ROW_FORMAT=Compact` 또는 `ROW_FORMAT=Dynamic`으로 전환하기 전에 원본 테이블과 새 테이블에 대한 워크로드의 성능을 테스트합니다. 또한 데이터 볼륨 증가와 같은 다른 잠재적 영향도 고려합니다.

어떤 문이 Aurora MySQL의 병렬 쿼리를 사용하는지 확인

일반적인 작업에서는 병렬 쿼리를 이용하기 위해 어떤 특별한 조치를 수행할 필요가 없습니다. 쿼리가 병렬 쿼리를 위한 필수적 요구 사항을 충족한 후에는, 쿼리 옵티마이저가 각 특정 쿼리에 대하여 병렬 쿼리를 사용할지 여부를 자동으로 결정합니다.

개발 환경 또는 테스트 환경에서 실험을 실행하는 경우, 테이블에 행의 수 또는 전체 데이터 볼륨이 너무 작아서 병렬 쿼리가 사용되지 않을 수도 있습니다. 특히 실험을 수행하기 위해 최근에 만든 테이블의 경우, 테이블의 데이터가 버퍼 풀에서 전부 사용될 수도 있습니다.

클러스터 성능을 모니터링하거나 튜닝할 때, 병렬 쿼리가 적절한 컨텍스트에서 사용되고 있는지 여부를 확인해야 합니다. 이 기능을 활용하기 위해 데이터베이스 스키마, 설정, SQL 쿼리 또는 심지어 클러스터 토폴로지 및 애플리케이션 연결 설정을 조정할 수도 있습니다.

쿼리가 병렬 쿼리를 사용하고 있는지 확인하려면, EXPLAIN 문을 실행하여 쿼리 계획("실행 계획"이라고도 함)을 확인합니다. SQL 문, 절 및 표현식이 병렬 쿼리의 `EXPLAIN` 출력에 어떠한 영향을 미치는지에 대한 예는 Aurora MySQL의 병렬 쿼리를 위한 SQL 구성를 참조하십시오.

다음 예제는 기존의 쿼리 계획과 병렬 쿼리 계획 간의 차이점을 보여줍니다. 이 설명 계획은 TPC-H 벤치마크의 쿼리 3에서 나온 것입니다. 이 단원의 곳곳에 나오는 샘플 쿼리의 대부분이 TPC-H 데이터 세트의 테이블을 사용합니다. [TPC-H 웹 사이트]에서 샘플 데이터를 생성하는 dbgen 프로그램, 테이블 정의 및 쿼리를 얻을 수 있습니다.

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

기본적으로 쿼리에 다음과 같은 계획이 있을 수 있습니다. 쿼리 계획에 해시 조인이 사용된 것으로 표시되지 않으면 먼저 최적화가 설정되어 있는지 확인합니다.

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

Aurora MySQL 버전 3의 경우 다음 문을 실행하여 세션 수준에서 해시 조인을 설정합니다.

```
SET optimizer_switch='block_nested_loop=on';
```

Aurora MySQL 버전 2.09 이상에서는 aurora_disable_hash_join DB 파라미터 또는 DB 클러스터 파라미터를 0(비활성화됨)으로 설정합니다. aurora_disable_hash_join을 비활성화하면 optimizer_switch의 값이 hash_join=on이 됩니다.

해시 조인을 활성화한 후 EXPLAIN 문을 다시 실행해 보세요. 해시 조인을 효과적으로 사용하는 방법에 대한 자세한 내용은 해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화 섹션을 참조하세요.

해시 조인이 설정되어 있지만 병렬 쿼리가 해제되어 있으면, 쿼리가 다음과 같은 계획(병렬 쿼리가 아니라 해시 조인을 사용)을 사용할 수 있습니다.

```
+----+-------------+----------+...+-----------
+-----------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-----------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-----------------------------------------------------------------+
```

병렬 쿼리가 설정된 후에는 이 쿼리 계획의 두 단계가 Extra 출력의 EXPLAIN 열 아래에 표시된 대로 병렬 쿼리 최적화를 사용할 수 있습니다. 이 두 단계를 위한 I/O 집약적이고 CPU 집약적인 처리는 스토리지 계층으로 푸시 다운됩니다.

```
+----+...
+------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                  |
+----+...
+------------------------------------------------------------------------------------------
+
```

```
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

병렬 쿼리 및 병렬 쿼리가 적용될 수 있는 SQL 문의 부분에 대한 EXPLAIN 출력을 해석하는 방법에 대한 내용은 Aurora MySQL의 병렬 쿼리를 위한 SQL 구성을 참조하십시오.

다음 예제 출력은 콜드 버퍼 풀이 있는 db.r4.2xlarge 인스턴스에서 이전 쿼리를 실행한 결과를 보여줍니다. 병렬 쿼리 사용 시 쿼리가 상당히 더 빠르게 실행됩니다.

Note

타이밍은 많은 환경 요인에 의존하기 때문에 결과가 다를 수 있습니다. 본인의 고유한 환경, 워크로드 등에서의 결과를 확인하기 위해 항상 본인의 성능 테스트를 수행하십시오.

```
-- Without parallel query
+------------+-------------+------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
```

```
+------------+------------+------------+----------------+
10 rows in set (1 min 49.91 sec)
```

이 단원의 곳곳에 나오는 샘플 쿼리의 대부분은 이 TPC-H 데이터 세트의 테이블을 사용합니다(특히, 2천만 개의 행과 다음 정의가 포함된 PART 테이블).

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

개별 SQL 문이 병렬 쿼리를 활용할 수 있는지 가늠하기 위해 본인의 워크로드로 시험하십시오. 그런 다음, 시간이 지남에 따라 실제 워크로드에서 병렬 쿼리가 얼마나 자주 사용되는지 확인할 수 있도록 다음 모니터링 기법을 사용합니다. 실제 워크로드의 경우, 동시성 한도와 같은 추가 요인이 적용됩니다.

Aurora MySQL용 병렬 쿼리 모니터링

Aurora MySQL 클러스터가 병렬 쿼리를 사용하는 경우 `VolumeReadIOPS` 값이 증가할 수 있습니다. 병렬 쿼리는 버퍼 풀을 사용하지 않습니다. 따라서 쿼리는 빠르지만 이렇게 최적화된 프로세싱은 읽기 작업 및 관련 비용을 증가시킬 수 있습니다.

Amazon RDS 콘솔에서 지표 보기에 설명된 Amazon CloudWatch 지표 외에도 Aurora는 다른 전역적 상태 변수를 제공합니다. 이러한 전역 상태 변수를 사용하여 병렬 쿼리 실행을 모니터링할 수 있습니다. 이를 통해 최적화 프로그램이 주어진 상황에서 병렬 쿼리를 사용하거나 사용하지 않는 이유를 파악할 수 있습니다. 이러한 변수에 액세스하기 위해 `SHOW GLOBAL STATUS` 명령을 사용할 수 있습니다. 또한 아래에 나열된 변수도 찾을 수 있습니다.

병렬 쿼리 세션은 데이터베이스에서 수행한 쿼리와 반드시 일대일 매핑되는 것은 아닙니다. 예를 들어, 쿼리 계획에 병렬 쿼리를 사용하는 두 단계가 있다고 가정해 봅시다. 이 경우에, 쿼리는 두 병렬 세션 및 시도된 요청을 위한 카운터가 필요하고 성공한 요청은 2씩 증가합니다.

EXPLAIN 문을 실행하여 병렬 쿼리로 시험할 때, 쿼리가 실제로 실행 중이지 않은 경우에도 "선택되지 않음"으로 지정된 카운터에서 증가가 있을 것으로 예상합니다. 프로덕션에서 병렬 쿼리로 작업할 때 "선택되지 않은" 카운터가 예상보다 더 빠르게 증가하고 있는지 확인할 수 있습니다. 이때 예상한 쿼리에 대해 병렬 쿼리가 실행되도록 조정할 수 있습니다. 이렇게 하려면 클러스터 설정, 쿼리 조합, 병렬 쿼리가 설정된 DB 인스턴스 등을 변경할 수 있습니다.

이러한 카운터는 DB 인스턴스 수준에서 추적됩니다. 서로 다른 엔드포인트에 연결된 경우, 각 DB 인스턴스가 자체의 고유한 병렬 쿼리 집합을 실행하기 때문에 서로 다른 지표가 표시될 수도 있습니다. 또한 리더 엔드포인트가 각 세션마다 서로 다른 DB 인스턴스에 연결된 경우에도 서로 다른 지표가 표시될 수 있습니다.

이름	설명
Aurora_pq_bytes_returned	병렬 쿼리 동안 헤드 노드에 전송된 튜플 데이터 구조를 위한 바이트 수입니다. Aurora_pq_pages_pushed_down 과 비교하기 위해 16,384로 나눕니다.
Aurora_pq_max_concurrent_requests	이 Aurora DB 인스턴스에서 동시에 실행될 수 있는 병렬 쿼리 세션의 최대 수입니다. 이 수는 AWS DB 인스턴스 클래스에 따라 결정되는 고정된 수입니다.
Aurora_pq_pages_pushed_down	병렬 쿼리가 헤드 노드로 네트워크 전송을 회피한 데이터 페이지의 수입니다(각각 16 KiB의 고정 크기).
Aurora_pq_request_attempted	요청된 병렬 쿼리 세션의 수입니다. 이 값은 하위 쿼리 및 조인과 같은 SQL 구조에 따라, 쿼리당 두 개 이상의 세션을 나타낼 수 있습니다.
Aurora_pq_request_executed	병렬 쿼리 세션의 수가 성공적으로 실행됩니다.
Aurora_pq_request_failed	클라이언트에 오류를 반환한 병렬 쿼리 세션의 수입니다. 일부 경우에 병렬 쿼리를 위한 요청이 실패할 수도 있습니다(예를 들어, 스토리지 계층의 문제로 인해). 이러한 경우에는 실패한 쿼리 부분이 비병렬 쿼리 메커니즘을 사용하여 다시 시도됩니다. 다시 시도된 쿼리 또한 실패하는 경

이름	설명
	우, 오류가 클라이언트에 반환되고 이 카운터가 증가합니다.
`Aurora_pq_request_in_progress`	병렬 쿼리 세션의 수가 현재 진행 중입니다. 이 수는 전체 Aurora DB 클러스터가 아니라 연결되어 있는 특정 Aurora DB 인스턴스에 적용됩니다. DB 인스턴스가 동시성 한도에 근접했는지 확인하려면, 이 값을 `Aurora_pq_max_concurrent_requests` 와 비교합니다.
`Aurora_pq_request_not_chosen`	병렬 쿼리가 쿼리를 충족시키기 위해 선택되지 않은 횟수입니다. 이 값은 여러 개의 다른 더 세분화된 카운터의 합계입니다. 쿼리가 실제로 수행되지 않더라도 `EXPLAIN` 문을 통해 이 카운터가 증가할 수 있습니다.
`Aurora_pq_request_not_chosen_below_min_rows`	테이블에 있는 행의 수로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 쿼리가 실제로 수행되지 않더라도 `EXPLAIN` 문을 통해 이 카운터가 증가할 수 있습니다.
`Aurora_pq_request_not_chosen_column_bit`	예상 열 목록에서 지원되지 않는 데이터 형식으로 인해 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
`Aurora_pq_request_not_chosen_column_geometry`	테이블에 `GEOMETRY` 데이터 형식의 열이 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 제한 사항이 적용되지 않는 Aurora MySQL 버전에 대한 자세한 정보는 [Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드](#) 섹션을 참조하세요.

이름	설명
Aurora_pq_request_not_chosen_column_lob	테이블에 LOB 데이터 형식의 열이 있거나 VARCHAR 선언된 길이로 인해 외부에 저장된 열이 있으므로에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 제한 사항이 제거된 Aurora MySQL 버전에 대한 자세한 정보는 [Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드](#) 섹션을 참조하세요.
Aurora_pq_request_not_chosen_column_virtual	테이블에 가상 열이 포함되어 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_custom_charset	테이블에 사용자 지정 문자 집합이 있는 열이 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_fast_ddl	테이블이 현재 빠른 DDL ALTER 문에 의해 변경되고 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	테이블 데이터 중 95% 미만이 버퍼 풀에 있는 경우에도, 버퍼링되지 않은 테이블 데이터가 병렬 쿼리가 가치 있을 만큼 충분하지 않았기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다.
Aurora_pq_request_not_chosen_full_text_index	테이블에 전체 텍스트 인덱스가 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_high_buffer_pool_pct	테이블 데이터 중 많은 양이(현재, 95% 이상) 이미 버퍼 풀에 있었기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다. 이러한 경우, 옵티마이저는 버퍼 풀에서 데이터 읽기가 더 효율적이라고 결정합니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.

이름	설명
Aurora_pq_request_not_chosen_index_hint	쿼리에 인덱스 힌트가 포함되어 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_innodb_table_format	테이블이 지원되지 않는 InnoDB 행 형식을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. Aurora 병렬 쿼리는 COMPACT, REDUNDANT 및 DYNAMIC 행 형식에만 적용됩니다.
Aurora_pq_request_not_chosen_long_trx	장기 실행 트랜잭션 내부에서 쿼리가 시작되는 중이어서 비병렬 쿼리 처리 경로를 사용한 병렬 쿼리 요청의 수입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.
Aurora_pq_request_not_chosen_no_where_clause	쿼리에 WHERE 절이 포함되어 있지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_range_scan	쿼리가 인덱스에서 범위 스캔을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_row_length_too_long	모든 열의 총 결합 길이가 너무 길기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_small_table	행의 수 및 평균 행 길이에 따라 결정된 테이블의 전체 크기로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.

이름	설명
Aurora_pq_request_not_chosen_temporary_table	쿼리가 지원되지 않는 MyISAM 또는 memory 테이블 형식을 사용하는 임시 테이블을 참조하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_tx_isolation	쿼리가 지원되지 않는 트랜잭션 격리 수준을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 읽기 전용 DB 인스턴스에서는 병렬 쿼리가 REPEATABLE READ 및 READ COMMITTED 격리 수준에만 적용됩니다.
Aurora_pq_request_not_chosen_update_delete_stmts	쿼리가 UPDATE 또는 DELETE 문의 일부이기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_unsupported_access	WHERE 절이 병렬 쿼리를 위한 기준에 부합되지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 결과는 쿼리에 데이터 집약적인 스캔이 필요하지 않은 경우이거나 쿼리가 DELETE 또는 UPDATE 문인 경우에 발생할 수 있습니다.
Aurora_pq_request_not_chosen_unsupported_storage_type	Aurora MySQL DB 클러스터가 지원되는 Aurora 클러스터 스토리지 구성을 사용하지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청 수입니다. 이 파라미터는 Aurora MySQL 버전 3.04 이상에서 사용할 수 있습니다. 자세한 내용은 제한 사항 단원을 참조하십시오.
Aurora_pq_request_throttled	동시 병렬 쿼리의 최대 수가 이미 특정 Aurora DB 인스턴스에서 실행 중이기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다.

Aurora MySQL의 병렬 쿼리를 위한 SQL 구성

다음 단원에서는 특정 SQL 문이 병렬 쿼리를 사용하거나 사용하지 않는 이유와 Aurora MySQL 기능이 병렬 쿼리와 상호 작용하는 방법에 대한 자세한 내용을 확인할 수 있습니다. 이러한 정보는 병렬 쿼리를 사용하는 클러스터의 성능 문제를 진단하고 병렬 쿼리가 특정 워크로드에 적용되는 방법을 이해하는 데 도움이 될 수 있습니다.

병렬 쿼리를 사용하려는 결정은 문이 실행되는 시점에 발생하는 여러 요인에 의존합니다. 따라서 병렬 쿼리는 특정 쿼리를 위해 항상 사용되거나 절대 사용되지 않거나 특정 조건에서만 사용될 수 있습니다.

> **Tip**
>
> HTML에서 이러한 예제를 볼 때 각 코드 목록의 오른쪽 위에 있는 복사 위젯을 사용해 SQL 코드를 복사하여 직접 시도할 수 있습니다. 복사 위젯을 사용하면 `mysql>` 프롬프트 및 `->` 연속 줄 주위에 추가 문자가 복사되지 않습니다.

주제

- [전체 텍스트 검색(FTS) 인덱스]
- [가상 열]
- [내장된 캐싱 메커니즘]
- [최적화 프로그램 힌트]
- [MyISAM 임시 테이블]

EXPLAIN 문

이 섹션의 곳곳에 나오는 예제에서 보듯이, EXPLAIN 문은 쿼리의 각 단계가 병렬 쿼리를 위해 현재 적격한지 여부를 나타냅니다. 또한 쿼리의 어떤 부분이 스토리지 계층으로 푸시 다운될 수 있는지를 나타냅니다. 다음은 쿼리 계획에서 가장 중요한 항목입니다.

- NULL 열의 key 외에 다른 값은 쿼리가 인덱스 조회를 사용하여 효율적으로 수행될 수 있어서 병렬 쿼리가 사용될 가능성이 낮음을 암시합니다.
- rows 열의 작은 값(즉, 값이 수백만이 아님)은 쿼리가 병렬 쿼리가 가치 있을 만큼 충분한 데이터에 액세스하고 있지 않음을 암시합니다. 즉, 병렬 쿼리가 발생할 가능성이 낮습니다.
- Extra 열은 병렬 쿼리가 사용될 것으로 예상되는 경우를 표시합니다. 이 출력은 다음 예제와 비슷합니다.

  ```
  Using parallel query (A columns, B filters, C exprs; D extra)
  ```

  columns 수는 쿼리 블록에서 조회되는 열의 수를 나타냅니다.

  filters 수는 상수에 대한 열 값의 단순 비교를 나타내는 WHERE 조건자의 수를 나타냅니다. 등식, 부등식 또는 범위에 대해 비교할 수 있습니다. Aurora는 이러한 종류의 조건자를 가장 효율적으로 병렬화할 수 있습니다.

  exprs 수는 함수 호출, 연산자, 또는 필터 조건만큼 효율적이지는 않지만 병렬화될 수 있는 다른 표현식 등과 같은 표현식의 수를 나타냅니다.

  extra 수는 푸시 다운될 수 없고 헤드 노드에 의해 수행되는 표현식의 수를 나타냅니다.

예를 들어, 다음 EXPLAIN 출력을 고려해 보십시오.

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
```

```
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+--------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+--------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+--------------------------------------------------------------------------+
```

Extra 열의 정보는 쿼리 조건을 평가하고 결과 집합을 구성하기 위해 5개의 열이 각 행으로부터 추출되는 것을 보여줍니다. 한 개의 WHERE 조건자는 필터(즉, WHERE 절에서 직접 테스트되는 열)를 포함합니다. 두 WHERE 절은 더 복잡한 표현식(이 경우에는 함수 호출 포함)의 평가가 필요합니다. 0 extra 필드는 WHERE 절의 모든 작업이 병렬 쿼리 처리의 부분으로 스토리지 계층으로 푸시 다운됨을 확인합니다.

병렬 쿼리가 선택되지 않은 경우에는, 일반적으로 EXPLAIN 출력의 다른 열에서 이유를 추론할 수 있습니다. 예를 들어, rows 값이 너무 작을 수 있거나 possible_keys 열은 쿼리가 데이터 집약적인 스캔 대신 인덱스 조회를 사용할 수 있음을 나타낼 수 있습니다. 다음 예제에서는 쿼리가 적은 수의 행만 스캔할 것이라고 최적화 프로그램에서 추정할 수 있는 쿼리를 보여줍니다. 이는 기본 키의 특성을 기반으로 합니다. 이 경우에는 병렬 쿼리가 필요하지 않습니다.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
```

병렬 쿼리의 사용 여부를 보여주는 출력은 EXPLAIN 문이 실행되는 시점에 사용할 수 있는 모든 요인을 고려합니다. 그 동안에 상황이 변경된 경우, 옵티마이저는 쿼리가 실제로 실행될 때 다른 선택을 할 수도 있습니다. 예를 들어, EXPLAIN은 문이 병렬 쿼리를 사용할 것이라고 보고할 수 있습니다. 그러나 나중에 쿼리가 실제로 실행될 때에는, 그 당시 조건을 기반으로 병렬 쿼리를 사용하지 않을 수도 있습

니다. 그러한 조건에는 여러 다른 병렬 쿼리가 동시에 실행 중, 행이 테이블에서 삭제되는 중, 새 인덱스가 생성되는 중, 열린 트랜잭션 내에서 너무 많은 시간 경과 등이 포함될 수 있습니다.

WHERE 절

쿼리가 병렬 쿼리 최적화를 사용하기 위해서는 반드시 WHERE 절을 포함해야 합니다.

병렬 쿼리 최적화는 WHERE 절에 사용되는 다양한 유형의 표현식 속도를 높입니다.

- 상수에 대한 열 값의 단순 비교(필터라고 알려짐). 이러한 비교는 스토리지 계층으로 푸시 다운되는 경우에 가장 많은 이점이 있습니다. 쿼리에 있는 필터 표현식의 수가 EXPLAIN 출력에 보고됩니다.
- WHERE 절에 있는 다른 유형의 표현식 또한 가능한 경우 스토리지 계층으로 푸시 다운됩니다. 쿼리에 있는 그러한 표현식의 수가 EXPLAIN 출력에 보고됩니다. 이러한 표현식은 함수 호출, LIKE 연산자, CASE 표현식 등이 해당될 수 있습니다.
- 특정 함수 및 연산자는 현재 병렬 쿼리에 의해 푸시 다운되지 않습니다. 쿼리에 있는 그러한 표현식의 수는 extra 출력에서 EXPLAIN 카운터로 보고됩니다. 나머지 쿼리도 여전히 병렬 쿼리를 사용할 수 있습니다.
- 선택 목록의 표현식이 푸시 다운되지 않은 동안, 그러한 함수를 포함한 쿼리는 병렬 쿼리의 중간 결과를 위한 네트워크 트래픽 감소로부터 여전히 이점을 얻을 수 있습니다. 예를 들어, 선택 목록의 집계 함수를 호출하는 쿼리는 집계 함수가 푸시 다운되지 않는 경우에도 병렬 쿼리로부터 이점을 얻을 수 있습니다.

예를 들어, 다음 쿼리는 전체 테이블 스캔을 수행하고 P_BRAND 열의 모든 값을 처리합니다. 하지만, 쿼리가 WHERE 절을 하나도 포함하고 있지 않기 때문에 병렬 쿼리를 사용하지 않습니다.

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                           |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

반면에, 다음 쿼리는 결과를 필터링하는 WHERE 조건자를 포함하고 있으므로 병렬 쿼리가 적용될 수 있습니다.

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+----------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                             |
+----+...+----------
+----------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+----------------------------------------------------------------------------------------------
+
```

옵티마이저가 쿼리 블록에 대해 반환되는 행의 수가 적다고 예상하는 경우, 병렬 쿼리가 해당 쿼리 블록에 사용되지 않습니다. 다음은 기본 키 열의 "~보다 큼" 연산자가 수백만의 행에 적용되고, 그로 인해 병렬 쿼리가 사용되는 사례를 보여주는 예제입니다. 정반대의 "~보다 작음" 테스트는 소수의 행에만 적용될 것으로 예상되므로 병렬 쿼리를 사용하지 않습니다.

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+--------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+--------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+--------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

데이터 정의 언어(DDL)

Aurora MySQL 버전 2에서 병렬 쿼리는 대기 중인 빠른 데이터 정의 언어(DDL) 작업이 없는 테이블에서만 사용할 수 있습니다. Aurora MySQL 버전 3에서는 인스턴트 DDL 작업과 동시에 테이블의 병렬 쿼리를 사용할 수 있습니다.

Aurora MySQL 버전 3의 인스턴트 DDL은 Aurora MySQL 버전 2의 빠른 DDL 기능을 대체합니다. 인스턴트 DDL에 대한 자세한 내용은 [인스턴트 DDL(Aurora MySQL 버전 3)](#) 섹션을 참조하세요.

열 데이터 유형

Aurora MySQL 버전 3에서 병렬 쿼리는 `TEXT`, `BLOB`, `JSON` 및 `GEOMETRY` 데이터 유형이 있는 열을 포함하는 테이블에서 작업할 수 있습니다. 또한 선언된 최대 길이가 768바이트보다 긴 `VARCHAR` 및 `CHAR` 열에서 작업할 수 있습니다. 쿼리가 이처럼 대형 객체 유형을 포함하는 모든 열을 참조하는 경우 추가로 검색 작업을 수행하면 쿼리 처리에 일부 오버헤드가 추가됩니다. 이 경우 쿼리가 해당 열에 대한 참조를 생략할 수 있는지 확인합니다. 그렇지 않은 경우 벤치마크를 실행하여 병렬 쿼리가 설정되거나 해제된 상태에서 이러한 쿼리가 더 빠르게 수행되는지 확인합니다.

Aurora MySQL 버전 2에는 대형 객체 유형에 대해 병렬 쿼리에 다음과 같은 제한이 있습니다.

- `TEXT`, `BLOB`, `JSON` 및 `GEOMETRY` 데이터 유형은 병렬 쿼리에서 지원되지 않습니다. 이러한 유형의 열을 참조하는 쿼리는 병렬 쿼리를 사용할 수 없습니다.
- 가변 길이 열(`VARCHAR` 및 `CHAR` 데이터 유형)은 768바이트의 최대 선언된 길이까지 병렬 쿼리와 호환됩니다. 더 긴 최대 길이로 선언된 유형의 열을 참조하는 쿼리는 병렬 쿼리를 사용할 수 없습니다. 멀티바이트 문자 집합을 사용하는 열의 경우, 바이트 제한이 문자 집합의 최대 바이트 수를 고려합니다. 예를 들어, `utf8mb4` 문자 집합(최대 문자 길이 = 4바이트)의 경우 `VARCHAR(192)` 열은 병렬 쿼리와 호환되지만 `VARCHAR(193)` 열은 호환되지 않습니다.

분할된 테이블

Aurora MySQL 버전 3에서 병렬 쿼리에서 파티션된 테이블을 사용할 수 있습니다. 파티션된 테이블은 내부적으로 여러 개의 작은 테이블로 표시되므로 파티션되지 않은 테이블에서 병렬 쿼리를 사용하는 쿼리는 동일하게 파티션된 테이블에서 병렬 쿼리를 사용할 수 없습니다. Aurora MySQL은 전체 테이블의 크기를 평가하는 대신 각 파티션이 병렬 쿼리 최적화에 적합할 만큼 충분히 큰지 여부를 고려합니다. 파티션된 테이블의 쿼리가 예상될 때 병렬 쿼리를 사용하지 않는 경우 `Aurora_pq_request_not_chosen_small_table` 상태 변수가 증가하는지 여부를 확인합니다.

예를 들어 한 테이블을 `PARTITION BY HASH (column) PARTITIONS 2`로 파티션하고 다른 테이블을 `PARTITION BY HASH (column) PARTITIONS 10`으로 파티션하는 것을 고려합니다. 두

개의 파티션이 있는 테이블의 파티션은 10개의 파티션이 있는 테이블보다 5배 더 큽니다. 따라서 병렬 쿼리는 파티션이 적은 테이블에 대한 쿼리에 사용될 가능성이 높습니다. 다음 예에서는 테이블 PART_BIG_PARTITIONS에는 두 개의 파티션이 있으며 PART_SMALL_PARTITIONS는 10개의 파티션이 있습니다. 동일한 데이터를 사용하면 병렬 쿼리는 큰 파티션이 적은 테이블에 사용될 가능성이 높습니다.

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+------------
+----------------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                          |
+----+-------------+---------------------+------------
+----------------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+---------------------+------------
+----------------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+-----------------------+-------------------------------
+------------------------------+
| id | select_type | table                 | partitions                    | Extra
                   |
+----+-------------+-----------------------+-------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+-----------------------+-------------------------------
+------------------------------+
```

집계 함수, GROUP BY 절 및 HAVING 절

집계 함수를 포함하는 쿼리는 대용량 테이블 내에 많은 수의 행 스캔이 필요하기 때문에, 대개 병렬 쿼리를 위한 좋은 후보가 됩니다.

Aurora MySQL 3에서 병렬 쿼리는 선택 목록 및 HAVING 절에서 집계 함수 호출을 최적화할 수 있습니다.

Aurora MySQL 3 이전에서 선택 목록 또는 HAVING 절에 있는 집계 함수 호출은 스토리지 계층으로 푸시 다운되지 않습니다. 하지만, 병렬 쿼리는 집계 함수에서도 그러한 쿼리의 성능을 여전히 향상시킬 수 있습니다. 그렇게 하기 위해서 먼저 스토리지 계층에서 병렬적으로 원시 데이터 페이지로부터 열 값을 추출합니다. 그런 다음 이러한 값을 전체 데이터 페이지가 아니라 간소화된 튜플 형식으로 다시 헤드 노드로 전송합니다. 항상 그렇듯이, 쿼리는 병렬 쿼리를 위해 최소 하나 이상의 WHERE조건자가 활성화되어야 합니다.

다음은 병렬 쿼리로부터 이점을 얻을 수 있는 집계 쿼리의 유형을 보여주는 간단한 예제입니다. 중간 결과를 간소화된 형식으로 헤드 노드에 반환하거나, 중간 결과에서 일치하지 않는 행을 필터링하거나, 혹은 둘 다를 사용함으로써 이점을 얻습니다.

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+----------------------------------------------------------------------------+
| id |...| Extra                                                                      |
+----+...+----------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+----------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-------------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------------------
+
```

WHERE 절의 함수 호출

Aurora는 병렬 쿼리 최적화를 WHERE 절의 대부분 내장 함수에 대한 호출에 적용할 수 있습니다. 이러한 함수 호출의 병렬화는 헤드 노드에서 일부 CPU 작업을 오프로드합니다. 가장 빠른 쿼리 단계 동안 조건자 함수를 병렬적으로 평가하면 Aurora가 전송되어 이후 단계에서 처리되는 데이터의 양을 최소화하는데 도움이 됩니다.

현재, 병렬화는 선택 목록의 함수 호출에 적용되지 않습니다. 이러한 함수는 동일한 함수 호출이 WHERE 절에 나타날지라도 헤드 노드에 의해 평가됩니다. 관련 열의 원래 값은 스토리지 노드에서 다시 헤드 노드로 전송되는 튜플에 포함되어 있습니다. 헤드 노드는 결과 집합의 최종 값을 생성하기 위해 UPPER, CONCATENATE 등과 같은 변환을 수행합니다.

다음 예제에서는, LOWER에 대한 호출이 WHERE 절에 나타나기 때문에 병렬 쿼리가 이 호출을 병렬화합니다. SUBSTR 및 UPPER에 대한 호출이 선택 목록에 나타나기 때문에 병렬 쿼리가 이러한 호출에 영향을 주지 않습니다.

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

동일한 고려 사항이 CASE 표현식 또는 LIKE 연산자와 같은 다른 표현식에 적용됩니다. 예를 들어, 다음 예제에서는 병렬 쿼리가 CASE 절의 LIKE 표현식 및 WHERE 연산자를 평가하는 것을 보여줍니다.

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
```

```
    -> group by p_retailprice;
+----+...
+-----------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
```

LIMIT 절

현재, 병렬 쿼리는 LIMIT 절을 포함하는 쿼리 블록에는 사용되지 않습니다. 병렬 쿼리는 GROUP by, ORDER BY 또는 조인이 포함된 이전 쿼리 단계에 여전히 사용될 수 있습니다.

비교 연산자

옵티마이저는 비교 연산자를 평가하기 위해 스캔해야 하는 행의 수를 추산하고, 그 추산을 기반으로 병렬 쿼리를 사용할지 여부를 결정합니다.

아래에 첫 번째 예제는 기본 키 열에 대한 "같음" 비교가 병렬 쿼리 없이도 효율적으로 수행될 수 있음을 보여줍니다. 아래에 두 번째 예제는 인덱싱되지 않은 열에 대한 "더 작음" 비교에 수백만 개 행의 스캔이 필요하며 따라서 병렬 쿼리로부터 이점을 얻을 수 있음을 보여줍니다.

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+--------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+--------------------------------------------------------------------------+
```

```
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-----------------------------------------------------------------------------+
```

동일한 고려 사항이 "같지 않음" 테스트에는 적용되지 않고 ~보다 작음/~보다 큼/같음 또는 BETWEEN과 같은 범위 비교에 적용됩니다. 옵티마이저는 스캔할 행의 수를 추산하고 I/O의 전체 볼륨을 기반으로 병렬 쿼리가 가치 있는지 여부를 결정합니다.

조인

대용량 테이블에서의 조인 쿼리는 일반적으로 병렬 쿼리 최적화로부터 이점을 얻는 데이터 집약적인 작업을 포함합니다. 여러 테이블 간에 열 값의 비교는(즉, 조인 조건자 자체) 현재 병렬화되지 않습니다. 하지만, 병렬 쿼리는 해시 조인 동안 Bloom 필터 생성과 같은 다른 조인 단계를 위한 내부 처리의 일부를 푸시 다운할 수 있습니다. 병렬 쿼리는 WHERE 절 없이도 조인 쿼리에 적용될 수 있습니다. 그러므로, 조인 쿼리는 병렬 쿼리를 사용하기 위해 WHERE 절이 필요한 규칙에는 예외입니다.

조인 처리의 각 단계는 병렬 쿼리에 적격한지 여부를 확인하기 위해 평가됩니다. 둘 이상의 단계에서 병렬 쿼리를 사용할 수 있는 경우 이러한 단계는 순차적으로 수행됩니다. 따라서, 각 조인 쿼리는 동시성 한도 면에서 단일 병렬 쿼리세션으로 계산됩니다.

예를 들어, 조인 쿼리에 조인된 테이블 중 한 테이블에서 행을 필터링하기 위한 WHERE 조건자가 포함된 경우, 해당 필터링 옵션은 병렬 쿼리를 사용할 수 있습니다. 다른 예로서 예를 들어, 큰 테이블을 작은 테이블과 조인하기 위해 조인 쿼리가 해시 조인 메커니즘을 사용한다고 가정해 봅시다. 이 사례에서는, Bloom 필터 데이터 구조를 생성하기 위한 테이블 스캔이 병렬 쿼리를 사용할 수도 있습니다.

> **Note**
>
> 병렬 쿼리는 해시 조인 최적화에서 이익을 얻는 리소스 집약적인 쿼리 유형에 일반적으로 사용됩니다. 해시 조인 최적화를 설정하는 방법은 Aurora MySQL 버전에 따라 다릅니다. 각 버전에 대한 자세한 내용은 [병렬 쿼리 클러스터에 대한 해시 조인 설정](#) 섹션을 참조하세요. 해시 조인을 효과적으로 사용하는 방법에 대한 자세한 내용은 [해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화](#) 섹션을 참조하세요.

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+-------+---------------+-------------+...+-----------
+-----------------------------------------------------------------------------------------
+
```

```
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+-------+---------------+-------------+...+-----------
+-----------------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+---------------+-------------+...+-----------
+-----------------------------------------------------------------------------------------------------
+
```

중첩 루프 메커니즘을 사용하는 조인 쿼리의 경우, 가장 바깥쪽 중첩 루프 블록은 병렬 쿼리를 사용할 수도 있습니다. 병렬 쿼리의 사용은 평상시처럼 동일한 요인(예: WHERE 절에 추가 필터 조건의 유무)에 의존합니다.

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+----------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                 |
+----+-------------+----------+...+----------
+----------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                        |
+----+-------------+----------+...+----------
+----------------------------------------------------------------------------+
```

하위 쿼리

외부 쿼리 블록 및 내부 하위 쿼리 블록은 각각 병렬 쿼리를 사용하거나 사용하지 않을 수 있습니다. 이는 각 블록에 대한 테이블의 일반적인 특성, WHERE 절 등을 기반으로 합니다. 예를 들어, 다음 쿼리는 하위 쿼리 블록에 대해서는 병렬 쿼리를 사용하지만 외부 블록에 대해서는 사용하지 않습니다.

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-----------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                         |
+----+-------------+...+----------
+-----------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                         |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-----------------------------------------------------------------------------+
```

현재, 상관관계가 있는 하위 쿼리는 병렬 쿼리 최적화를 사용할 수 없습니다.

UNION

UNION 쿼리의 각 쿼리 블록은 WHERE의 각 부분에 대하여 테이블의 일반적인 특성, UNION 절 등을 기반으로 병렬 쿼리를 사용할 수 있거나 사용할 수 없습니다.

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+----------------+...+----------
+-----------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                            |
+----+----------------+...+----------
+-----------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                        |
+----+--------------+...+----------
+-----------------------------------------------------------------------------+
```

> **Note**
>
> 쿼리 내의 각 UNION 절은 순차적으로 실행됩니다. 쿼리가 모두 병렬 쿼리를 사용하는 여러 단계를 포함한 경우에도, 한 번에 하나의 병렬 쿼리만 실행합니다. 그러므로, 복잡한 다단계 쿼리조차도 동시 병렬 쿼리의 한도에 1로만 가산됩니다.

보기

옵티마이저는 기본 테이블을 사용하는 더 긴 쿼리로 보기를 사용하는 쿼리를 재작성합니다. 따라서, 병렬 쿼리는 테이블 참조가 보기이든 실제 테이블이든 관계 없이 동일하게 작동합니다. 쿼리에 병렬 쿼리를 사용할지 여부 및 어떤 부분을 푸시 다운할지에 대해 모두 동일한 고려 사항이 최종적으로 재작성된 쿼리에 적용됩니다.

예를 들어, 다음 쿼리 계획은 일반적으로 병렬 쿼리를 사용하지 않는 보기 정의를 보여줍니다. 보기가 추가 WHERE 절로 쿼리되는 경우 Aurora MySQL은 병렬 쿼리를 사용합니다.

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-----------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-----------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-----------------------------------------------------------------------------+
```

데이터 조작 언어(DML) 문

INSERT 부분이 병렬 쿼리를 위한 다른 조건에 부합하는 경우, SELECT 문이 처리의 SELECT 단계에 병렬 쿼리를 사용할 수 있습니다.

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-----------------------------------------------------------------------------+
```

```
| id |...| rows     | Extra
         |
+----+...+----------
+----------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+----------------------------------------------------------------------------+
```

> **Note**
>
> 일반적으로, INSERT 문 뒤에 새로 삽입된 행을 위한 데이터는 버퍼 풀에 있습니다. 그러므로, 테이블이 많은 수의 행을 삽입한 직후에는 병렬 쿼리에 적격하지 않을 수도 있습니다. 나중에 데이터가 정상 작동 중에 버퍼 풀에서 제거된 후, 테이블에 대한 쿼리가 다시 병렬 쿼리를 사용하기 시작할 수 있습니다.

문의 CREATE TABLE AS SELECT 부분이 다른 식으로 병렬 쿼리에 적격할 수 있는 경우에도 SELECT 문은 병렬 쿼리를 사용하지 않습니다. 이 문의 DDL 부분으로 인하여 병렬 쿼리 처리와 호환되지 않게 됩니다. 반면, INSERT ... SELECT 문에서는 SELECT 부분이 병렬 쿼리를 사용할 수 있습니다.

테이블의 크기 및 DELETE 절의 조건자에 관계 없이 UPDATE 또는 WHERE 문에는 병렬 쿼리가 사용되지 않습니다.

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

트랜잭션 및 잠금

Aurora 기본 인스턴스에서 모든 격리 수준을 사용할 수 있습니다.

Aurora 리더 DB 인스턴스의 경우 REPEATABLE READ 격리 수준에서 수행되는 문에 병렬 쿼리가 적용됩니다. 또한 Aurora MySQL 버전2.09 이상에서는 리더 DB 인스턴스에서 READ COMMITTED 격리 수준을 사용할 수 있습니다. REPEATABLE READ는 Aurora 리더 DB 인스턴스에 대한 기본 격리 수준입니다. 읽기 전용 DB 인스턴스에서 READ COMMITTED 격리 수준을 사용하려면 세션 수준에서 aurora_read_replica_read_committed 구성 옵션을 설정해야 합니다. 리더 인스턴스의 READ

COMMITTED 격리 수준은 SQL 표준 동작을 준수합니다. 그러나 리더 인스턴스의 격리는 쿼리가 작성자 인스턴스에서 READ COMMITTED 격리를 사용하는 경우보다 덜 엄격합니다.

Aurora 격리 수준, 특히 작성자 인스턴스와 리더 인스턴스 간 READ COMMITTED 차이점에 대한 자세한 내용은 Aurora MySQL 격리 수준 섹션을 참조하세요.

대형 트랜잭션이 완료된 후에는 테이블 통계가 기한 경과될 수 있습니다. Aurora가 행의 수를 정확하게 추산할 수 있으려면 먼저 그러한 기한 경과 통계에 ANALYZE TABLE 문이 필요할 수 있습니다. 대규모 DML 문 또한 테이블 데이터의 상당 부분을 버퍼 풀로 가져올 수 있습니다. 이러한 데이터가 버퍼 풀에 있으면 데이터가 풀에서 제거될 때까지 해당 테이블에 대해 병렬 쿼리가 덜 자주 선택될 수 있습니다.

세션이 장기 실행 트랙잭션(기본적으로 10분) 내부에 있는 경우, 해당 세션 내부의 추가 쿼리는 병렬 쿼리를 사용하지 않습니다. 또한 단일 장기 실행 쿼리 동안 시간 초과가 발생할 수 있습니다. 병렬 쿼리 처리가 시작되기 전에 쿼리가 최대 간격(현재 10분)보다 더 오래 실행되는 경우 이러한 유형의 시간 초과가 발생할 수 있습니다.

임시(1회) 쿼리를 수행하는 autocommit=1 세션에 mysql을 설정하여 실수로 장기 실행 트랜잭션을 시작할 가능성을 줄일 수 있습니다. 읽기 보기를 만들면 테이블에 대한 SELECT 문도 트랜잭션을 시작합니다. 읽기 보기는 트랜잭션이 커밋될 때까지 남아있는 후속 쿼리를 위한 일관된 데이터 집합입니다. Aurora에서 JDBC 또는 ODBC 애플리케이션을 사용할 때도 이러한 제한에 유의하십시오. 이러한 애플리케이션은 autocommit 설정이 꺼진 상태에서도 실행될 수 있기 때문입니다.

다음은 autocommit 설정이 꺼진 상태에서, 테이블에 대한 쿼리 실행이 트랜잭션을 암시적으로 시작하는 읽기 보기를 생성하는 방법을 보여주는 예제입니다. 그 후에 짧게 실행되는 쿼리는 여전히 병렬 쿼리를 사용할 수 있습니다. 하지만, 몇 분간 일시 정지 후에는 쿼리가 병렬 쿼리에 더 이상 적격하지 않습니다. COMMIT 또는 ROLLBACK으로 트랜잭션을 종료하면 병렬 쿼리 자격을 복원합니다.

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+----------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+----------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+----------------------------------------------------------------------------+
```

```
mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+---------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+---------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+---------------------------------------------------------------------------+
```

쿼리가 장기 실행 트랜잭션 내부에 있었기 때문에 병렬 쿼리에 적격하지 않은 횟수를 확인하려면 상태 변수 `Aurora_pq_request_not_chosen_long_trx`를 확인합니다.

```
mysql> show global status like '%pq%trx%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+-------------------------------+-------+
```

`SELECT` 또는 `SELECT FOR UPDATE` 구문과 같은 잠금을 획득하는 `SELECT LOCK IN SHARE MODE` 문은 병렬 쿼리를 사용할 수 없습니다.

병렬 쿼리는 `LOCK TABLES` 문에 의해 잠겨진 테이블을 위해 작동할 수 있습니다.

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------------+
| id |...| rows      | Extra
           |
+----+...+-----------
+-------------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

B-트리 인덱스

ANALYZE TABLE 문에 의해 수집된 통계는 옵티마이저가 각 열에 대한 데이터의 특성을 기반으로 병렬 쿼리 또는 인덱스 조회를 사용하는 경우를 결정하도록 도와줍니다. 테이블 내의 데이터에 상당한 변경을 적용하는 DML 작업 후에 ANALYZE TABLE을 실행하여 통계를 현재 상태로 유지하십시오.

인덱스 조회가 데이터 집약적인 스캔 없이도 효율적으로 쿼리를 수행할 수 있는 경우 Aurora은 인덱스 조회를 사용할 수도 있습니다. 그렇게 하면 병렬 쿼리 처리의 오버헤드를 피할 수 있습니다. 또한 Aurora DB 클러스터에서 동시에 실행될 수 있는 병렬 쿼리의 수에 대한 동시성 제한도 있습니다. 가장 빈번하게 가장 많이 사용되는 동시 쿼리가 인덱스 조회를 사용하도록, 테이블 인덱싱을 위한 모범 사례를 사용하십시오.

전체 텍스트 검색(FTS) 인덱스

현재, 병렬 쿼리는 전체 텍스트 검색 인덱스를 포함한 테이블에 사용되지 않습니다(쿼리가 그러한 인덱싱된 열을 참조하든 MATCH 연산자를 사용하든 관계 없음).

가상 열

현재 쿼리가 가상 열을 참조하는지 여부에 관계없이 가상 열이 포함된 테이블에는 병렬 쿼리가 사용되지 않습니다.

내장된 캐싱 메커니즘

Aurora에는 내장된 캐싱 메커니즘(즉, 버퍼 풀 및 쿼리 캐시)이 포함되어 있습니다. Aurora 옵티마이저는 어떤 것이 특정 쿼리에 가장 효과적인지에 따라서 이러한 캐싱 메커니즘과 병렬 쿼리 중에서 선택합니다.

병렬 쿼리가 행을 필터링하고 열 값을 변환 및 추출할 때, 데이터는 데이터 페이지가 아니라 튜플로 다시 헤드 노드에 전송됩니다. 그러므로, 병렬 쿼리를 실행해도 버퍼 풀에 페이지가 추가되거나 버퍼 풀에 이미 존재하는 페이지가 제거되지 않습니다.

Aurora는 버퍼 풀에 있는 테이블 데이터의 페이지 수와 해당 숫자가 나타내는 테이블 데이터의 비율을 확인합니다. Aurora는 이 정보를 사용하여 병렬 쿼리(및 버퍼 풀의 데이터 우회)를 사용하는 것이 더 효율적인지 여부를 결정합니다. 또는 Aurora은 버퍼 풀에 캐시된 데이터를 사용하는 비병렬 쿼리 처리 경로를 사용할 수도 있습니다. 어떤 페이지가 캐시되고 데이터 집약적인 쿼리가 캐싱 및 제거에 어떠한 영향을 미치는지는 버퍼 풀에 관련된 구성 설정에 따라 다릅니다. 따라서 선택은 버퍼 풀 내에서 끊임없이 변하는 데이터에 달려 있기 때문에, 특정 쿼리가 병렬 쿼리를 사용할지 여부를 예측하기는 어려울 수 있습니다.

또한, Aurora은 병렬 쿼리에 동시성 한도를 적용합니다. 모든 쿼리가 병렬 쿼리를 사용하는 것은 아니기 때문에, 여러 쿼리에 의해 동시에 액세스되는 테이블은 일반적으로 버퍼 풀에 데이터의 상당 부분이 있습니다. 따라서, Aurora은 대개 병렬 쿼리를 위해 이러한 테이블은 선택하지 않습니다.

동일한 테이블에서 비병렬 쿼리의 시퀀스를 실행하는 경우, 첫 번째 쿼리는 데이터가 버퍼 풀에 없어서 오래 걸릴 수 있습니다. 그런 다음 두 번째 및 이후 쿼리는 버퍼 풀이 이제 "워밍업"되어서 훨씬 더 빨라집니다. 병렬 쿼리는 일반적으로 테이블에 대한 맨 첫 번째 쿼리부터 일관된 성능을 보여줍니다. 성능 테스트를 수행할 때는 콜드(cold) 버퍼 풀과 웜(warm) 버퍼 풀에서 모두 비병렬 쿼리를 벤치마크합니다. 일부 경우, 웜 버퍼 풀에서의 결과는 병렬 쿼리 시간과 잘 비교될 수 있습니다. 이러한 경우 해당 테이블에 대한 쿼리 빈도와 같은 요소를 고려합니다. 또한 버퍼 풀에 해당 테이블의 데이터를 유지하는 것이 가치가 있는지 여부를 고려해야 합니다.

동일한 쿼리가 제출되고 기본 테이블 데이터가 변경되지 않은 경우 쿼리 캐시는 쿼리 재실행을 방지합니다. 병렬 쿼리 기능에 의해 최적화된 쿼리는 쿼리 캐시로 이동할 수 있어, 쿼리가 다시 실행될 때 즉각적으로 실행되므로 효율적입니다.

> **Note**
>
> 성능 비교를 수행할 때, 쿼리 캐시가 인위적으로 낮은 시간 지정 숫자를 생성할 수 있습니다. 따라서, 벤치마크 같은 상황에서는 `sql_no_cache` 힌트를 사용할 수 있습니다. 이 힌트는 이전에 동일한 쿼리가 실행된 경우에도, 결과가 쿼리 캐시로부터 제공되지 않도록 합니다. 힌트는

쿼리의 SELECT 문 바로 뒤에 옵니다. 병렬 쿼리가 설정 및 해제된 쿼리의 버전들 간에 쿼리 시간이 비교 가능해지도록 이 주제의 많은 병렬 쿼리 예에 이 힌트가 포함되어 있습니다. 병렬 쿼리의 프로덕션용으로 이동할 때는 원본에서 이 힌트를 제거해야 합니다.

최적화 프로그램 힌트

최적화 프로그램을 제어하는 또 다른 방법은 개별 문 내에서 지정할 수 있는 최적화 프로그램 힌트를 사용하는 것입니다. 예를 들어, 문 안에 있는 한 테이블에 대해 최적화를 설정한 다음 다른 테이블에 대한 최적화를 끌 수 있습니다. 이러한 힌트에 대한 자세한 내용은 MySQL 참조 설명서의 최적화 프로그램 힌트를 참조하세요.

Aurora MySQL 쿼리와 함께 SQL 힌트를 사용하여 성능을 미세 조정할 수 있습니다. 또한 힌트를 사용하여 예기치 않은 조건으로 인해 중요한 쿼리에 대한 실행 계획이 변경되는 것을 방지할 수 있습니다.

쿼리 계획에 대한 최적화 프로그램 선택 항목을 제어할 수 있도록 SQL 힌트 기능을 확장했습니다. 이러한 힌트는 병렬 쿼리 최적화를 사용하는 쿼리에 적용됩니다. 자세한 내용은 Aurora MySQL 힌트 단원을 참조하십시오.

MyISAM 임시 테이블

병렬 쿼리 최적화는 InnoDB 테이블에만 적용됩니다. Aurora MySQL은 임시 테이블을 위해 배후에서 MyISAM을 사용하기 때문에, 임시 테이블을 포함하는 내부 쿼리 단계는 병렬 쿼리를 사용합니다. 이러한 쿼리 단계는 `Using temporary` 출력에서 EXPLAIN로 표시됩니다.

Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용

Amazon Aurora MySQL에서 고성능 고급 감사 기능을 사용하여 데이터베이스 활동을 감사할 수 있습니다. 이렇게 하려면 여러 DB 클러스터 파라미터를 설정하여 감사 로그 모음을 활성화합니다. 고급 감사가 활성화되면 이 기능을 사용하여 지원되는 이벤트의 모든 조합을 기록할 수 있습니다.

감사 로그를 보거나 다운로드하여 한 번에 하나의 DB 인스턴스에 대한 감사 정보를 검토할 수 있습니다. 이를 위해 Amazon Aurora 로그 파일 모니터링의 절차를 사용할 수 있습니다.

> Tip
>
> 여러 DB 인스턴스가 포함된 Aurora DB 클러스터의 경우 클러스터의 모든 인스턴스에 대한 감사 로그를 검사하는 것이 더 편리할 수 있습니다. 이를 위해 CloudWatch Logs를 사용할 수 있습니다. 클러스터 수준에서 설정을 켜서 Aurora MySQL 감사 로그 데이터를 CloudWatch의 로그 그룹에 게시할 수 있습니다. 그런 다음 CloudWatch 인터페이스를 통해 감사 로그를 보고, 필터링하고, 검색할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.

고급 감사 활성화

이 단원에서 설명하는 파라미터를 사용하여 DB 클러스터에 대해 고급 감사를 활성화하고 구성할 수 있습니다.

`server_audit_logging` 파라미터를 사용하여 감사 로그를 사용 설정하거나 사용 중지합니다.

`server_audit_events` 파라미터를 사용하여 로그할 이벤트를 지정합니다.

감사를 받을 사용자를 지정하려면 `server_audit_incl_users` 및 `server_audit_excl_users` 파라미터를 사용합니다. 기본적으로 모든 사용자가 감사됩니다. 하나 또는 둘 다 비어 있거나 둘 다에 동일한 사용자 이름이 지정된 경우 이러한 파라미터가 작동하는 방식에 대한 자세한 내용은 server_audit_incl_users 및 server_audit_excl_users 섹션을 참조하세요.

DB 클러스터가 사용하는 파라미터 그룹에서 다음 파라미터를 설정하여 고급 감사를 구성합니다. Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원에서 설명하는 절차에 따라 AWS Management Console을 사용하여 DB 클러스터 파라미터를 수정할 수 있습니다. modify-db-cluster-parameter-group AWS CLI 명령 또는 ModifyDBClusterParameterGroup Amazon RDS API 작업을 사용하여 DB 클러스터 파라미터를 프로그래밍 방식으로 수정할 수 있습니다.

파라미터 그룹이 클러스터와 이미 연결되어 있을 때 이들 파라미터를 수정할 때 DB 클러스터를 다시 시작할 필요가 없습니다. 파라미터 그룹을 클러스터와 처음 연결하는 경우 클러스터를 다시 시작해야 합니다.

주제

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

고급 감사를 활성화 또는 비활성화합니다. 이 파라미터는 OFF가 기본값입니다. 고급 감사를 활성화하려면 ON으로 설정합니다.

`server_audit_events` 파라미터를 사용하여 감사할 이벤트 유형을 하나 이상 정의하지 않는 한 감사 데이터가 로그에 표시되지 않습니다.

DB 인스턴스에 대한 감사 데이터가 기록되었는지 확인하려면 해당 인스턴스에 대한 일부 로그 파일의 이름이 `audit/audit.log.`*`other_identifying_information`* 형식인지 확인합니다. 로그 파일의 이름을 보려면 데이터베이스 로그 파일 보기 및 나열의 절차를 따릅니다.

server_audit_events

기록할 이벤트의 쉼표로 구분된 목록을 포함합니다. 이벤트는 모두 대문자로 지정해야 하며 목록 항목 사이에 공백이 없어야 합니다. 예: `CONNECT,QUERY_DDL`. 이 파라미터는 빈 문자열이 기본값입니다.

다음 이벤트의 모든 조합을 기록할 수 있습니다.

- CONNECT – 성공 및 실패한 연결을 모두 기록하고 연결 해제도 기록합니다. 이 이벤트는 사용자 정보를 포함합니다.
- QUERY – 모든 쿼리를 일반 텍스트로 기록합니다(구문 또는 권한 오류로 인해 실패한 쿼리를 포함).

> **Tip**
>
> 이 이벤트 유형을 설정하면 감사 데이터에는 Aurora가 자동으로 수행하는 지속적인 모니터링 및 상태 확인 정보가 포함됩니다. 특정 종류의 작업에만 관심이 있다면 보다 구체적인 종

류의 이벤트를 사용할 수 있습니다. CloudWatch 인터페이스를 사용하여 로그에서 특정 데이터베이스, 테이블 또는 사용자와 관련된 이벤트를 검색할 수도 있습니다.

- QUERY_DCL – 쿼리 이벤트와 유사하지만 데이터 제어 언어(DCL) 쿼리(GRANT, REVOKE 등)만 반환합니다.
- QUERY_DDL – 쿼리 이벤트와 유사하지만 데이터 정의 언어(DDL) 쿼리(CREATE, ALTER 등)만 반환합니다.
- Query_DML – 쿼리 이벤트와 유사하지만 데이터 조작 언어(DML) 쿼리(INSERT, UPDATE 등 및 SELECT)만 반환합니다.
- TABLE – 쿼리 실행의 영향을 받은 테이블을 기록합니다.

> **Note**
>
> Aurora에는 감사 로그에서 특정 쿼리를 제외하는 필터가 없습니다. `SELECT` 쿼리를 제외하려면 모든 DML 문을 제외해야 합니다.
> 특정 사용자가 감사 로그에서 이러한 내부 `SELECT` 쿼리를 보고하는 경우, server_audit_excl_users DB 클러스터 파라미터를 설정하여 해당 사용자를 제외할 수 있습니다. 그러나 해당 사용자가 다른 활동에도 사용되어 생략할 수 없는 경우에는 `SELECT` 쿼리를 제외할 수 있는 다른 옵션이 없습니다.

server_audit_incl_users

활동이 기록된 사용자의 쉼표로 구분된 사용자 이름 목록을 포함합니다. 목록 항목 사이에 공백이 없어야 합니다. 예: `user_3,user_4`. 이 파라미터는 빈 문자열이 기본값입니다. 최대 길이는 1024자입니다. 지정된 사용자 이름이 `User` 테이블 내 `mysql.user` 열의 해당 이름과 일치해야 합니다. 사용자 이름에 대한 자세한 정보는 MySQL 설명서의 계정 사용자 이름 및 암호를 참조하세요.

`server_audit_incl_users` 및 `server_audit_excl_users`가 모두 비어 있는 경우(기본값) 모든 사용자가 감사됩니다.

`server_audit_incl_users`에 사용자를 추가하고 `server_audit_excl_users`는 비워둘 경우 해당 사용자만 감사됩니다.

`server_audit_excl_users`에 사용자를 추가하고 `server_audit_incl_users`는 비워둘 경우 `server_audit_excl_users`에 나열된 사용자를 제외한 모든 사용자가 감사됩니다.

server_audit_excl_users 및 server_audit_incl_users 모두에 사용자를 추가할 경우 해당 사용자가 감사됩니다. 동일한 사용자가 두 설정에 모두 나열되어 있는 경우 server_audit_incl_users의 우선순위가 높습니다.

Connect 및 Disconnect 이벤트는 이 변수의 영향을 받지 않습니다. 지정할 경우 이들 이벤트는 항상 기록됩니다. 사용자가 server_audit_excl_users 파라미터에도 지정되었더라도 server_audit_incl_users의 우선 순위가 더 높으므로 해당 사용자가 기록됩니다.

server_audit_excl_users

활동이 기록되지 않는 사용자의 쉼표로 구분된 사용자 이름 목록을 포함합니다. 목록 항목 사이에 공백이 없어야 합니다. 예: rdsadmin,user_1,user_2. 이 파라미터는 빈 문자열이 기본값입니다. 최대 길이는 1024자입니다. 지정된 사용자 이름이 User 테이블 내 mysql.user 열의 해당 이름과 일치해야 합니다. 사용자 이름에 대한 자세한 정보는 MySQL 설명서의 [계정 사용자 이름 및 암호를](#) 참조하세요.

server_audit_incl_users 및 server_audit_excl_users가 모두 비어 있는 경우(기본값) 모든 사용자가 감사됩니다.

server_audit_excl_users에 사용자를 추가하고 server_audit_incl_users는 비워둘 경우 server_audit_excl_users에 나열하는 해당 사용자만 제외하고 다른 모든 사용자가 감사됩니다.

server_audit_excl_users 및 server_audit_incl_users 모두에 사용자를 추가할 경우 해당 사용자가 감사됩니다. 동일한 사용자가 두 설정에 모두 나열되어 있는 경우 server_audit_incl_users의 우선순위가 높습니다.

Connect 및 Disconnect 이벤트는 이 변수의 영향을 받지 않습니다. 지정할 경우 이들 이벤트는 항상 기록됩니다. 사용자가 server_audit_incl_users 파라미터에도 지정된 경우 이 설정의 우선 순위가 server_audit_excl_users보다 높으므로 해당 사용자가 기록됩니다.

감사 로그 보기

콘솔을 사용하여 감사 로그를 확인하고 다운로드할 수 있습니다. Databases(데이터베이스) 페이지에서 DB 인스턴스를 선택하여 세부 정보를 표시한 다음, 로그 섹션으로 스크롤합니다. 고급 감사 기능에 의해 생성된 감사 로그는 audit/audit.log.*other_identifying_information* 형식의 이름을 갖습니다.

로그 파일을 다운로드하려면 로그 섹션에서 해당 파일을 선택한 다음 다운로드를 선택합니다.

또한 [describe-db-log-files](#) AWS CLI 명령을 사용하여 로그 파일 목록을 가져올 수 있습니다. [download-db-log-file-portion](#) AWS CLI 명령을 사용하여 로그 파일의 내용을 다운로드할 수 있습니다.

자세한 내용은 데이터베이스 로그 파일 보기 및 나열 및 데이터베이스 로그 파일 다운로드 단원을 참조하십시오.

감사 로그 세부 정보

로그 파일은 UTF-8 형식의 쉼표로 구분된 변수(CSV) 파일로 표시됩니다. 쿼리는 작은따옴표(')로도 묶여 있습니다.

감사 로그는 각 Aurora MySQL DB 인스턴스의 로컬 스토리지에 별도로 저장됩니다. 각 인스턴스는 한 번에 4개의 로그 파일에 쓰기를 분산합니다. 최대 로그 파일 크기는 100MB입니다. 이 구성 불가능한 한도에 도달하면 Aurora는 파일을 교체하고 새 파일을 생성합니다.

> **Tip**
>
> 로그 파일 항목은 순서대로 나열되지 않습니다. 항목의 순서를 정하려면 타임스탬프 값을 사용합니다. 최신 이벤트를 보기 위해 모든 로그 파일을 확인해야 하는 경우가 있을 수 있습니다. 로그 데이터를 보다 유연하게 정렬하고 검색하려면 CloudWatch에 감사 로그를 업로드하고 CloudWatch 인터페이스를 사용하여 보는 설정을 켜세요.
> 더 많은 유형의 필드와 JSON 형식의 출력이 있는 감사 데이터를 보기 위해 데이터베이스 활동 스트림 기능을 사용할 수도 있습니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 단원을 참조하십시오.

감사 로그 파일은 지정된 순서대로 다음의 쉼표로 구분된 정보를 행에 포함합니다.

필드	설명
타임스탬프	기록된 이벤트에 대한 Unix 타임스탬프(마이크로초 단위)입니다.
serverhost	이벤트가 기록되는 인스턴스의 이름입니다.
사용자 이름	사용자의 연결된 사용자 이름입니다.
host	사용자가 연결한 호스트입니다.
connectionid	기록된 작업의 연결 ID 번호입니다.
queryid	관계형 테이블 이벤트 및 관련 쿼리를 검색하는 데 사용할 수 있는 쿼리 ID 번호입니다. TABLE 이벤트의 경우 여러 줄이 추가됩니다.

필드	설명
작업을 통해 처리 속도를 높일 수 있습니다	기록된 작업 유형입니다. 가능한 값은 CONNECT, QUERY, READ, WRITE, CREATE, ALTER, RENAME 및 DROP입니다.
데이터베이스	USE 명령에 의해 설정된 활성 데이터베이스입니다.
객체	QUERY 이벤트의 경우 이 값은 데이터베이스에서 수행한 쿼리를 나타냅니다. TABLE 이벤트의 경우 이 값은 테이블 이름을 나타냅니다.
retcode	기록된 작업의 반환 코드입니다.

Amazon Aurora MySQL을 사용한 복제

Aurora MySQL 복제 기능은 클러스터의 고가용성과 성능을 위한 핵심 기능입니다. Aurora에서는 최대 15개의 Aurora 복제본을 사용하여 클러스터를 손쉽게 생성하거나 크기를 조정할 수 있습니다.

모든 복제본은 동일한 기반 데이터에서 작동합니다. 일부 데이터베이스 인스턴스가 오프라인으로 전환되는 경우 다른 인스턴스는 쿼리를 계속 처리하거나 필요한 경우 라이터 역할을 대신할 수 있습니다. Aurora는 읽기 전용 연결을 여러 데이터베이스 인스턴스에 자동으로 분산시켜 Aurora 클러스터에서 쿼리 집약적인 워크로드를 지원할 수 있도록 지원합니다.

다음 주제에서는 Aurora MySQL 복제의 작동 방식과 최적의 가용성 및 성능을 위해 복제 설정을 미세 조정하는 방법에 대한 정보를 확인할 수 있습니다.

주제

Aurora 복제본 사용

Aurora 복제본은 Aurora DB 클러스터의 독립 엔드포인트로서, 읽기 작업을 조정하고 가용성을 높이기 위해 사용하기에 가장 적합합니다. 최대 15개의 Aurora 복제본을 AWS 리전 내 DB 클러스터에 포함된 가용 영역에 배포할 수 있습니다. DB 클러스터 볼륨은 DB 클러스터의 여러 데이터 사본으로 구성되어 있지만, DB 클러스터의 기본 인스턴스 및 Aurora 복제본에는 클러스터 볼륨 데이터가 단 하나의 논리 볼륨으로 표시됩니다. Aurora 복제본에 대한 자세한 내용은 Aurora 복제본 단원을 참조하십시오.

Aurora 복제본은 클러스터 볼륨의 읽기 연산에 전적으로 사용되므로 읽기 조정에 유용합니다. 쓰기 연산은 기본 인스턴스에서 관리합니다. 클러스터 볼륨은 Aurora MySQL DB 클러스터의 모든 인스턴스가 공유하기 때문에 각 Aurora 복제본의 데이터 사본을 추가로 복제할 필요는 없습니다. 이와는 대조적

으로 MySQL 읽기 전용 복제본은 소스 DB 인스턴스부터 로컬 데이터 스토어에 이르는 모든 쓰기 작업을 단일 스레드에서 재실행해야 합니다. 이러한 제한은 대용량 읽기 트래픽을 지원하는 MySQL 읽기 전용 복제본의 기능에 영향을 끼칠 수 있습니다.

Aurora MySQL을 사용하여 Aurora 복제본이 삭제될 때 인스턴스 엔드포인트가 즉시 제거되고, Aurora 복제본은 리더(Reader) 엔드포인트에서 제거됩니다. 삭제되는 Aurora 복제본을 실행하는 문이 있는 경우 3분의 유예 기간이 있습니다. 기존 문은 유예 기간 동안 정상적으로 완료할 수 있습니다. 유예 기간이 종료되면 Aurora 복제본이 종료 및 삭제됩니다.

> **Important**
>
> Aurora MySQL용 Aurora 복제본은 InnoDB 테이블의 작업에 대해 항상 기본 트랜잭션 격리 수준 `REPEATABLE READ`를 사용합니다. `SET TRANSACTION ISOLATION LEVEL` 명령을 사용하여 Aurora MySQL DB 클러스터의 기본 인스턴스에 대한 트랜잭션 수준만 변경할 수 있습니다. 이렇게 제한함에 따라 Aurora 복제본의 사용자 수준 잠금이 방지되고, 복제 지연 시간을 최소화하는 동시에 Aurora 복제본을 확장하여 수천 개의 활성 사용자 연결을 지원할 수 있습니다.

> **Note**
>
> 기본 인스턴스에서 실행되는 DDL 문은 연결된 Aurora 복제본의 데이터베이스 연결을 중단할 수 있습니다. Aurora 복제본 연결에서 테이블 등 데이터베이스 객체를 적극적으로 사용 중이고 DDL 문을 사용하여 해당 객체를 기본 인스턴스에서 수정하는 경우 Aurora 복제본 연결이 중단됩니다.

> **Note**
>
> 중국(닝샤) 리전은 리전 간 읽기 전용 복제본을 지원하지 않습니다.

Amazon Aurora MySQL의 복제 옵션

다음 옵션들 중에서 복제를 설정할 수 있습니다.

- Aurora MySQL DB 클러스터의 교차 리전 읽기 전용 복제본을 생성하여 다른 AWS 리전에 Aurora MySQL DB 클러스터 2개를 설정합니다.

자세한 내용은 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 단원을 참조하십시오.

- MySQL 이진 로그(binlog) 복제를 사용하여 동일한 AWS 리전에 있는 Aurora MySQL DB 클러스터 2개를 설정합니다.

  자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.

- RDS for MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 소스인 RDS for MySQL DB 인스턴스와 Aurora MySQL DB 클러스터를 설정합니다.

  이 접근 방식을 사용하여 Aurora로 마이그레이션하는 동안 기존 및 지속적인 데이터 변경 사항을 Aurora MySQL로 가져올 수 있습니다. 자세한 내용은 Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 단원을 참조하십시오.

  이 방법을 사용하여 데이터에 대한 읽기 쿼리의 확장성을 높일 수도 있습니다. 읽기 전용 Aurora MySQL 클러스터 내에 있는 하나 이상의 DB 인스턴스를 사용하여 데이터를 쿼리하면 됩니다. 자세한 내용은 Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 규모 조정 단원을 참조하십시오.

- Aurora Global Database를 생성하여 하나의 AWS 리전에 Aurora MySQL DB 클러스터와, 다른 리전에 최대 5개의 Aurora 읽기 전용 Aurora MySQL DB 클러스터를 설정합니다.

  Aurora Global Database를 사용하여 전 세계에 설치할 공간이 있는 애플리케이션을 지원할 수 있습니다. 기본 Aurora MySQL DB 클러스터에는 라이터 인스턴스와 최대 15개의 Aurora 복제본이 있습니다. 읽기 전용 보조 Aurora MySQL DB 클러스터는 각각 16개의 Aurora 복제본으로 구성될 수 있습니다. 자세한 내용은 Amazon Aurora Global Database 사용 섹션을 참조하세요.

> **Note**
>
> Amazon Aurora DB 클러스터의 기본 인스턴스를 재부팅하면 DB 클러스터 전체에서 읽기/쓰기 일관성을 보장하는 진입점을 다시 설정하기 위해 해당 DB 클러스터에 대한 Aurora 복제본도 자동으로 재부팅됩니다.

Amazon Aurora MySQL 복제를 위한 성능 고려 사항

다음 기능을 사용하여 Aurora MySQL 복제본의 성능을 세부 조정할 수 있습니다.

복제본 로그 압축 기능은 복제 메시지에 대한 네트워크 대역폭을 자동으로 줄입니다. 각 메시지는 Aurora 복제본에 전송되기 때문에 크기가 큰 클러스터에 매우 유용한 기능입니다. 이 기능은 압축을 수행하기 위해 라이터(Writer) 노드에서 약간의 CPU 오버헤드가 발생합니다. Aurora MySQL 버전 2와 버전 3에서는 항상 활성화되어 있습니다.

binlog 필터링 기능은 복제 메시지에 대한 네트워크 대역폭을 자동으로 줄입니다. Aurora 복제본은 복제 메시지에 포함된 binlog 정보를 사용하지 않기 때문에 해당 노드로 전송된 메시지에서 이 데이터가 제외됩니다.

Aurora MySQL 버전 2에서 `aurora_enable_repl_bin_log_filtering` 파라미터를 변경하여 이 기능을 제어할 수 있습니다. 이 파라미터는 기본적으로 활성화됩니다. 이 최적화는 투명성을 위한 것이기 때문에 복제 관련 문제에 대한 진단이나 문제 해결 중에만 이 설정을 비활성화할 수 있습니다. 예를 들어 이 기능이 제공되지 않은 이전 버전 Aurora MySQL 클러스터의 동작과 일치시키기 위해 이 설정을 비활성화할 수 있습니다.

Aurora MySQL 버전 3에서는 Binlog 필터링이 항상 활성화되어 있습니다.

Amazon Aurora MySQL의 제로 다운타임 다시 시작(ZDR)

제로 다운타임 다시 시작(ZDR) 기능은 특정 종류의 다시 시작 중에 DB 인스턴스에 대한 활성 연결의 일부 또는 전부를 보관할 수 있습니다. ZDR은 Aurora에서 오류 조건(예: 복제본이 소스보다 너무 멀리 지연되기 시작하는 경우)을 해결하기 위해 자동으로 수행하는 다시 시작에 적용됩니다.

> **Important**
>
> ZDR 메커니즘은 최선의 노력을 기반으로 작동합니다. Aurora MySQL 버전, 인스턴스 클래스, 오류 조건, 호환되는 SQL 작업 및 ZDR이 적용되는 위치를 결정하는 기타 요소는 언제든지 변경될 수 있습니다.

Aurora MySQL 2.x의 ZDR에는 버전 2.10 이상이 필요합니다. ZDR은 Aurora MySQL 3.x의 모든 마이너 버전에서 사용할 수 있습니다. Aurora MySQL 버전 2 및 3에서는 ZDR 메커니즘이 기본적으로 설정되어 있으며 Aurora에는 `aurora_enable_zdr` 파라미터가 사용되지 않습니다.

Aurora는 제로 다운타임 재시작과 관련된 활동을 이벤트 페이지에 보고합니다. Aurora는 ZDR 메커니즘을 사용하여 재시작을 시도할 때 이벤트를 기록합니다. 이 이벤트에는 Aurora에서 다시 시작을 수행하는 이유가 명시됩니다. 다시 시작이 완료되면 Aurora는 다른 이벤트를 기록합니다. 이 최종 이벤트는 프로세스의 소요 시간과 다시 시작 중에 유지되거나 삭제된 연결 수를 보고합니다. 데이터베이스 오류 로그를 참조하여 다시 시작 중에 발생한 활동에 대한 자세한 내용을 확인할 수 있습니다.

성공적인 ZDR 작업 후 연결은 그대로 유지되지만 일부 변수와 기능은 다시 초기화됩니다. 다음 유형의 정보는 제로 다운타임 다시 시작으로 인한 다시 시작 중에 보관되지 않습니다.

- 글로벌 변수 Aurora는 세션 변수를 복원하지만 다시 시작 후 글로벌 변수를 복원하지 않습니다.
- 상태 변수. 특히 엔진 상태에 의해 보고된 가동 시간 값은 재설정됩니다.
- `LAST_INSERT_ID`.
- 테이블의 인 메모리 `auto_increment` 상태. 인 메모리 자동 증분 상태는 다시 초기화됩니다. 자동 증분 값에 대한 자세한 내용은 [MySQL 참조 매뉴얼](#)을 참조하세요.
- `INFORMATION_SCHEMA` 및 `PERFORMANCE_SCHEMA` 테이블의 진단 정보. 이 진단 정보는 `SHOW PROFILE` 및 `SHOW PROFILES`와 같은 명령 출력에도 표시됩니다.

다음 표에는 버전, 인스턴스 역할 및 클러스터의 DB 인스턴스를 다시 시작할 때 Aurora에서 ZDR 메커니즘을 사용할 수 있는지 여부를 결정하는 기타 상황이 나와 있습니다.

Aurora MySQL version	라이터에 ZDR 적용 여부	리더에 ZDR 적용 여부	ZDR 상시 활성화 여부	참고
2.x, 2.10.0 미만	아니요	아니요	N/A	이러한 버전에서는 ZDR을 사용할 수 없습니다.
2.10.0~2.11.0	예	예	예	Aurora는 활성 연결에서 진행 중인 모든 트랜잭션을 롤백합니다. 애플리케이션에서 트랜잭션을 다시 시도해야 합니다. Aurora는 TLS/SSL, 임시 테이블, 테이블 잠금 또는 사용자 잠금을 사용하는 모든 연결을 취소합니다.
2.11.1 이상	예	예	예	Aurora는 활성 연결에서 진행 중인 모든 트랜잭션을 롤백합니다. 애플리케이션에서 트랜잭션을 다시 시도해야 합니다. Aurora는 임시 테이블, 테이블 잠금 또는 사용자 잠금을 사용하는 모든 연결을 취소합니다.

Aurora MySQL version	라이터에 ZDR 적용 여부	리더에 ZDR 적용 여부	ZDR 상시 활성화 여부	참고
3.01~3.03	예	예	예	Aurora는 활성 연결에서 진행 중인 모든 트랜잭션을 롤백합니다. 애플리케이션에서 트랜잭션을 다시 시도해야 합니다. Aurora는 TLS/SSL, 임시 테이블, 테이블 잠금 또는 사용자 잠금을 사용하는 모든 연결을 취소합니다.
3.04 이상	예	예	예	Aurora는 활성 연결에서 진행 중인 모든 트랜잭션을 롤백합니다. 애플리케이션에서 트랜잭션을 다시 시도해야 합니다. Aurora는 임시 테이블, 테이블 잠금 또는 사용자 잠금을 사용하는 모든 연결을 취소합니다.

Aurora MySQL을 사용한 복제 필터 구성

복제 필터를 사용하여 읽기 복제본과 함께 복제할 데이터베이스와 테이블을 지정할 수 있습니다. 복제 필터는 데이터베이스와 테이블을 복제에 포함하거나 복제에서 제외할 수 있습니다.

다음은 복제 필터의 몇 가지 사용 사례입니다.

- 읽기 복제본의 크기를 줄이는 방법. 복제 필터링을 사용하면 읽기 복제본에 필요하지 않은 데이터베이스와 테이블을 제외할 수 있습니다.
- 보안상의 이유로 읽기 복제본에서 데이터베이스와 테이블을 제외하는 방법.
- 다른 읽기 복제본에서 특정 사용 사례에 대해 서로 다른 데이터베이스와 테이블을 복제하는 방법. 예를 들어 분석 또는 샤딩에 특정 읽기 복제본을 사용할 수 있습니다.
- 여러 AWS 리전에 읽기 전용 복제본이 있는 DB 클러스터의 경우 서로 다른 AWS 리전에 있는 다른 데이터베이스 또는 테이블을 복제합니다.
- 인바운드 복제 토폴로지에서 복제본으로 구성된 Aurora MySQL DB 클러스터에서 복제할 데이터베이스와 테이블을 지정합니다. 이 구성에 대한 자세한 정보는 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 섹션을 참조하세요.

주제

Aurora MySQL에 대한 복제 필터링 파라미터 설정

복제 필터를 구성하려면 다음 파라미터를 설정합니다.

- `binlog-do-db` – 지정된 이진 로그에 변경 사항을 복제합니다. binlog 소스 클러스터에 대해 이 파라미터를 설정하면 파라미터에 지정된 바이너리 로그만 복제됩니다.
- `binlog-ignore-db` – 지정된 이진 로그에 변경 사항을 복제하지 않습니다. binlog 소스 클러스터에 대해 `binlog-do-db` 파라미터가 설정되면 이 파라미터는 평가되지 않습니다.
- `replicate-do-db` – 지정된 데이터베이스에 변경 내용을 복제합니다. binlog 복제본 클러스터에 대해 이 파라미터를 설정하면 파라미터에 지정된 데이터베이스만 복제됩니다.
- `replicate-ignore-db` – 지정된 데이터베이스에 변경 내용을 복제하지 마세요. binlog 복제본 클러스터에 대해 `replicate-do-db` 파라미터가 설정되면 이 파라미터는 평가되지 않습니다.
- `replicate-do-table` – 지정된 테이블에 변경 사항을 복제합니다. 읽기 복제본에 대해 이 파라미터를 설정하면 파라미터에 지정된 테이블만 복제됩니다. 또한, `replicate-do-db` 또는 `replicate-ignore-db` 파라미터가 설정되면 복제에 지정된 테이블이 포함된 데이터베이스를 binlog 복제본 클러스터와 함께 포함해야 합니다.
- `replicate-ignore-table` – 지정된 테이블에 변경 내용을 복제하지 마세요. binlog 복제본 클러스터에 대해 `replicate-do-table` 파라미터가 설정되면 이 파라미터는 평가되지 않습니다.
- `replicate-wild-do-table` – 지정된 데이터베이스와 테이블 이름 패턴을 기반으로 테이블을 복제합니다. `%` 및 `_` 와일드카드 문자가 지원됩니다. `replicate-do-db` 또는 `replicate-ignore-db` 파라미터가 설정되면 복제에 지정된 테이블이 포함된 데이터베이스를 binlog 복제본 클러스터와 함께 포함해야 합니다.
- `replicate-wild-ignore-table` – 지정된 데이터베이스와 테이블 이름 패턴을 기반으로 테이블을 복제하지 마세요 `%` 및 `_` 와일드카드 문자가 지원됩니다. binlog 복제본 클러스터에 대해 `replicate-do-table` 또는 `replicate-wild-do-table` 파라미터가 설정되면 이 파라미터는 평가되지 않습니다.

파라미터는 나열된 순서대로 평가됩니다. 이러한 파라미터의 작동 방식에 대한 자세한 내용은 MySQL 설명서를 참조하세요.

- 일반적인 내용은 복제 서버 옵션 및 변수를 참조하세요.
- 데이터베이스 복제 필터링 파라미터를 평가하는 방법에 대한 자세한 내용은 데이터베이스 수준의 복제 및 이진 로깅 옵션 평가를 참조하세요.
- 테이블 복제 필터링 파라미터가 평가되는 방법에 대한 자세한 내용은 테이블 수준의 복제 옵션 평가를 참조하세요.

기본적으로 이러한 각 파라미터에는 빈 값이 있습니다. 각 binlog 클러스터에서 이러한 파라미터를 사용하여 복제 필터를 설정, 변경 및 삭제할 수 있습니다. 이러한 파라미터 중 하나를 설정할 때 각 필터를 쉼표로 구분합니다.

`%` 및 `_` 파라미터에 `replicate-wild-do-table` 및 `replicate-wild-ignore-table` 와일드카드 문자를 사용할 수 있습니다. `%` 와일드카드는 원하는 수의 문자를 찾으며 `_` 와일드카드는 한 문자만 찾습니다.

원본 DB 인스턴스의 이진 로깅 형식은 데이터 변경 기록을 결정하므로 복제에 중요합니다. `binlog_format` 파라미터 설정에 따라 복제가 행 기반인지 또는 문 기반인지가 결정됩니다. 자세한 내용은 단일 AZ 데이터베이스의 Aurora MySQL 이진 로깅 구성 섹션을 참조하세요.

> **Note**
> 원본 DB 인스턴스의 `binlog_format` 설정과 관계없이, 모든 DDL(데이터 정의어) 문은 문으로 복제됩니다.

Aurora MySQL에 대한 복제 필터링 제한 사항

Aurora MySQL에 대한 복제 필터링에 다음과 같은 제한 사항이 적용됩니다.

- 복제 필터는 Aurora MySQL 버전 3에서만 지원됩니다.
- 각 복제 필터링 파라미터에는 2,000자 제한이 있습니다.
- 복제 필터에서는 쉼표가 지원되지 않습니다.
- 복제 필터링은 XA 트랜잭션을 지원하지 않습니다.

  자세한 내용은 MySQL 설명서에서 XA 트랜잭션에 대한 제한 사항을 참조하세요.

Aurora MySQL에 대한 복제 필터링 예제

읽기 복제본에 대한 복제 필터링을 구성하려면 읽기 복제본과 연결된 DB 클러스터 파라미터 그룹에서 복제 필터링 파라미터를 수정합니다.

> **Note**
>
> 기본 DB 클러스터 파라미터 그룹을 수정할 수 없습니다. 읽기 복제본에서 기본 파라미터 그룹을 사용 중인 경우 새 파라미터 그룹을 생성하여 읽기 복제본과 연결합니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 파라미터 그룹에서 DB 클러스터 파라미터를 설정할 수 있습니다. 파라미터 설정에 대한 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정을 참조하세요. DB 클러스터 파라미터 그룹에서 파라미터를 설정하면 파라미터 그룹과 연결된 모든 DB 클러스터가 파라미터 설정을 사용합니다. DB 클러스터 파라미터 그룹에서 복제 필터링 파라미터를 설정하는 경우, 파라미터 그룹이 읽기 복제본 클러스터에만 연결되어 있는지 확인합니다. 원본 DB 인스턴스에 대해 복제 필터링 파라미터를 비워 둡니다.

다음 예제에서는 AWS CLI를 사용하여 파라미터를 설정합니다. 이 예제는 CLI 명령이 완료된 직후에 파라미터가 변경되도록 `ApplyMethod`을(를) `immediate`(으)로 설정합니다. 읽기 복제본이 재부팅된 후 보류 중인 변경 사항을 적용하려면 `ApplyMethod`을(를) `pending-reboot`(으)로 설정합니다.

다음 예제에서는 복제 필터를 설정합니다.

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example 복제에 데이터베이스 포함

다음 예제에서는 복제에 `mydb1` 및 `mydb2` 데이터베이스가 포함되어 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example 복제에 테이블 포함

다음 예제에서는 복제의 `table1` 데이터베이스에 `table2` 및 `mydb1` 테이블이 포함되어 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example 와일드카드 문자를 사용하여 복제에 테이블 포함

다음 예제에서는 복제의 데이터베이스 `order`에 `return` 및 `mydb`(으)로 시작하는 이름을 가진 테이블이 포함되어 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
```

```
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example 복제에서 데이터베이스 제외

다음 예제에서는 mydb5 및 mydb6 데이터베이스를 복제에서 제외합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example 복제에서 테이블 제외

다음 예시에서는 데이터베이스 mydb5의 table1 테이블과 데이터베이스 mydb6의 table2 테이블을 복제에서 제외합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example 와일드카드 문자를 사용하여 복제에서 테이블 제외

다음 예제에서는 데이터베이스 `order`에서 `return` 및 mydb7(으)로 시작하는 이름을 가진 테이블을 복제에서 제외합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

읽기 복제본에 대한 복제 필터 보기

다음과 같은 방법으로 읽기 복제본에 대한 복제 필터를 볼 수 있습니다.

- 읽기 복제본과 연결된 파라미터 그룹에서 복제 필터링 파라미터 설정을 확인합니다.

  지침은 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 값 보기](#) 섹션을 참조하세요.
- MySQL 클라이언트에서 읽기 전용 복제본에 연결하고 `SHOW REPLICA STATUS` 문을 실행합니다.

  출력 시, 다음 필드에서 읽기 복제본에 대한 복제 필터를 보여줍니다.
  - `Binlog_Do_DB`
  - `Binlog_Ignore_DB`
  - `Replicate_Do_DB`

- `Replicate_Ignore_DB`
- `Replicate_Do_Table`
- `Replicate_Ignore_Table`
- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

이러한 필드에 대한 자세한 내용은 MySQL 설명서의 복제 상태 확인을 참조하세요.

Amazon Aurora MySQL 복제 모니터링

읽기 조정과 고가용성은 최소 지연 시간에 따라 달라집니다. Amazon CloudWatch `AuroraReplicaLag` 지표를 모니터링하여 Aurora 복제본이 Aurora MySQL DB 클러스터의 기본 인스턴스보다 얼마나 지연되는지 모니터링할 수 있습니다. `AuroraReplicaLag` 지표는 각 Aurora 복제본에 기록됩니다.

기본 DB 인스턴스는 `AuroraReplicaLagMaximum` 및 `AuroraReplicaLagMinimum` Amazon CloudWatch 지표도 기록합니다. `AuroraReplicaLagMaximum` 메트릭은 기본 DB 인스턴스와 DB 클러스터의 각 Aurora 복제본 간의 최대 지연 정도를 기록합니다. `AuroraReplicaLagMinimum` 메트릭은 기본 DB 인스턴스와 DB 클러스터의 각 Aurora 복제본 간의 최소 지연 정도를 기록합니다.

Aurora 복제본 지연의 최신 값이 필요할 경우 Amazon CloudWatch에서 `AuroraReplicaLag` 지표를 확인할 수 있습니다. Aurora 복제본 지연은 `information_schema.replica_host_status` 테이블에 있는 Aurora MySQL DB 클러스터의 각 Aurora 복제본에도 기록됩니다. 이 테이블에 대한 자세한 내용은 information_schema.replica_host_status 섹션을 참조하세요.

RDS 인스턴스 및 CloudWatch 지표 모니터링에 대한 자세한 내용은 Amazon Aurora 클러스터에서 지표 모니터링 단원을 참조하세요.

여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제

Amazon Aurora MySQL DB 클러스터를 소스 DB 클러스터와 다른 AWS 리전에 읽기 전용 복제본으로 만들 수 있습니다. 이 방식을 택하면 재해 복구 기능을 개선하고, 읽기 작업을 사용자와 더욱 가까운 AWS 리전으로 확장하고, 한 AWS 리전에서 다른 리전으로 손쉽게 마이그레이션할 수 있습니다.

암호화된 DB 클러스터와 암호화되지 않은 DB 클러스터의 읽기 전용 복제본을 모두 만들 수 있습니다. 원본 DB 클러스터가 암호화되어 있으면 읽기 전용 복제본을 암호화해야 합니다.

원본 DB 클러스터당 최대 5개의 리전 간 DB 클러스터(읽기 전용 복제본)를 만들 수 있습니다.

> **Note**
>
> 교차 리전 읽기 전용 복제본의 대안으로, Aurora 전역 데이터베이스를 사용하여 최소한의 지연 시간으로 읽기 작업을 크기 조정할 수 있습니다. Aurora 글로벌 데이터베이스는 하나의 AWS 리전에 프라이머리 Aurora DB 클러스터를 그리고 서로 다른 리전에 최대 10개의 읽기 전용 보조 DB 클러스터를 보유합니다. 각 보조 DB 클러스터에는 최대 16개의 (15보다 나음) Aurora 복제본이 포함될 수 있습니다. 프라이머리 DB 클러스터에서 모든 보조 클러스터로의 복제는 데이터베이스 엔진이 아닌 Aurora 스토리지 계층에서 처리되므로 변경 사항 복제를 위한 지연 시간이 최소화됩니다(일반적으로 1초 미만). 데이터베이스 엔진을 복제 프로세스에서 제외하면 데이터베이스 엔진이 작업 로드 처리 전용으로 사용됩니다. 또한 Aurora MySQL의 binlog(바이너리 로깅) 복제를 구성하거나 관리할 필요가 없습니다. 자세한 내용은 [Amazon Aurora Global Database 사용](#)를 참조하세요.

다른 AWS 리전에 Aurora MySQL DB 클러스터의 읽기 전용 복제본을 만들 때는 다음에 주의해야 합니다.

- 원본 DB 클러스터와 리전 간 읽기 전용 복제본 DB 클러스터 모두 DB 클러스터의 기본 인스턴스 외에 최대 15개의 Aurora 복제본을 가질 수 있습니다. 이 기능을 통해, 소스 AWS 리전과 복제 대상 AWS 리전에 대해 읽기 작업을 확장할 수 있습니다.
- 교차 리전 시나리오에서는 AWS 리전 간 네트워크 채널이 더 길어지기 때문에 소스 DB 클러스터와 읽기 전용 복제본 간의 지연 시간이 증가합니다.
- 리전 간 복제를 위해 데이터를 전송할 때는 Amazon RDS 데이터 전송 요금이 발생합니다. 다음과 같은 교차 리전 복제 작업에서 요금이 발생하는 이유는 소스 AWS 리전을 벗어나 데이터를 전송하기 때문입니다.
  - 읽기 전용 복제본을 생성할 때는 Amazon RDS가 소스 클러스터의 스냅샷을 캡처하여 해당 스냅샷을 읽기 전용 복제본이 있는 AWS 리전으로 전송합니다.

  - 소스 데이터베이스에서 데이터를 변경할 때마다 Amazon RDS가 소스 리전에서 읽기 전용 복제본이 있는 AWS 리전으로 데이터를 전송합니다.

  Amazon RDS 데이터 전송 요금에 대한 자세한 정보는 Amazon Aurora 요금을 참조하십시오.

- 동일한 원본 DB 클러스터를 참조하는 읽기 전용 복제본에 대해 동시 생성 또는 삭제 작업을 여러 개 실행할 수 있습니다. 하지만 각 원본 DB 클러스터에 대한 읽기 전용 복제본 개수를 5개 이하로 유지해야 합니다.
- 효과적인 복제를 위해서는 읽기 전용 복제본도 각각 원본 DB 클러스터와 동일한 양의 컴퓨팅 및 스토리지 리소스를 가져야 합니다. 원본 DB 클러스터를 확장하면 읽기 전용 복제본도 확장해야 합니다.

주제

시작하기 전 준비 사항

리전 간 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들기 전에 먼저 원본 Aurora MySQL DB 클러스터에 대한 이진 로깅을 설정해야 합니다. Aurora MySQL의 리전 간 복제는 리전 간 읽기 전용 복제본 DB 클러스터에서 MySQL 이진 복제를 사용하여 변경 사항을 반복합니다.

Aurora MySQL DB 클러스터에 대한 이진 로깅을 설정하려면 원본 DB 클러스터의 `binlog_format` 파라미터를 업데이트해야 합니다. `binlog_format` 파라미터는 기본 클러스터 파라미터 그룹에 속하는 클러스터 수준 파라미터입니다. DB 클러스터에서 기본 DB 클러스터 파라미터 그룹을 사용하는 경우, `binlog_format` 설정을 수정하려면 새로운 DB 클러스터 파라미터 그룹을 만들어야 합니다. `binlog_format`을 `MIXED`로 설정하는 것이 좋습니다. 그러나 특정한 binlog 형식이 필요하다면 `binlog_format`을 `ROW` 또는 `STATEMENT`로 설정할 수도 있습니다. Aurora DB 클러스터를 재부팅하여 변경 사항을 적용하십시오.

Aurora MySQL을 통한 이진 로깅 사용에 대한 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제을(를) 참조하세요. Aurora MySQL 구성 파라미터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 및 Amazon Aurora의 파라미터 그룹을(를) 참조하세요.

Aurora MySQL의 교차 리전 읽기 전용 복제본 DB 클러스터 생성

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 교차 리전 읽기 전용 복제본인 Aurora DB 클러스터를 만들 수 있습니다. 암호화된 및 암호화되지 않은 DB 클러스터에서 모두 리전 간 읽기 전용 복제본을 만들 수 있습니다.

AWS Management Console을 사용하여 Aurora MySQL에 대한 교차 리전 읽기 전용 복제본을 만드는 경우, Amazon RDS는 대상 AWS 리전에 DB 클러스터를 만든 다음 해당 DB 클러스터의 기본 인스턴스인 DB 인스턴스를 자동으로 만듭니다.

AWS CLI 또는 RDS API를 사용하여 리전 간 읽기 전용 복제본을 만드는 경우, 먼저 대상 AWS 리전에 DB 클러스터를 만들고 활성화될 때까지 기다립니다. 활성화되면 해당 DB 클러스터의 기본 인스턴스인 DB 인스턴스를 만듭니다.

읽기 전용 복제본 DB 클러스터의 기본 인스턴스를 사용할 수 있게 되면 복제가 시작됩니다.

다음 절차를 사용하여 Aurora MySQL DB 클러스터에서 리전 간 읽기 전용 복제본을 만듭니다. 이러한 절차는 암호화되었거나 암호화되지 않은 DB 클러스터에서 읽기 전용 복제본을 만드는 데 사용됩니다.

콘솔

AWS Management Console을 사용하여 교차 리전 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단 모서리에서 소스 DB 클러스터를 호스팅하는 AWS 리전을 선택합니다.
3. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
4. 리전 간 읽기 전용 복제본을 만들 DB 클러스터를 선택합니다.
5. 작업(Actions)에서 리전 간 읽기 전용 복제본 만들기(Create cross-Region read replica)를 선택합니다.
6. 다음 표에 설명되어 있는 대로 리전 간 읽기 전용 복제본 만들기 페이지에서 리전 간 읽기 전용 복제본 DB 클러스터의 옵션 설정을 선택합니다.

옵션	설명
대상 리전	새로운 리전 간 읽기 전용 복제본 DB 클러스터를 호스팅할 AWS 리전을 선택합니다.
[Destination DB subnet group]	리전 간 읽기 전용 복제본 DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다.
공개적으로 액세스할 수 있음(Publicly accessible)	리전 간 읽기 전용 복제본 DB 클러스터에 퍼블릭 IP 주소를 할당하려면 예를 선택하고, 그렇지 않으면 아니요를 선택합니다.
암호화(Encryption)	암호화 활성화(Enable Encryption)를 선택하면 이 DB 클러스터에 대해 미사용 데이터 암호화를 설정할 수 있습니다. 자세한 내용은 Amazon Aurora 리소스 암호화 단원을 참조하십시오.
AWS KMS key	[Encryption]을 [Enable Encryption]으로 설정한 경우에만 사용할 수 있습니다. 현재 DB 클러스터를 암호화하는 데 사용할 AWS KMS key을(를) 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 단원을 참조하십시오.
DB 인스턴스 클래스	DB 클러스터의 기본 인스턴스에 대한 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.
다중 AZ 배포	장애 조치 지원을 위해 대상 AWS 리전의 다른 가용 영역에 새 DB 클러스터의 읽기 전용 복제본을 만들려면 예를 선택합니다. 다중 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 단원을 참조하십시오.
복제본 소스 읽기	리전 간 읽기 전용 복제본을 만들 원본 DB 클러스터를 선택합니다.

옵션	설명
DB 인스턴스 식별자	리전 간 읽기 전용 복제본 DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 새 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다. DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다. • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • 개별 AWS 리전별로 각 AWS 계정의 모든 DB 인스턴스에 대해 고유해야 합니다. 리전 간 읽기 전용 복제본 DB 클러스터는 원본 DB 클러스터의 스냅샷으로 만드는 것이기 때문에 읽기 전용 복제본의 마스터 사용자 이름과 마스터 암호는 원본 DB 클러스터의 마스터 사용자 이름 및 마스터 암호와 동일합니다.

옵션	설명
DB 클러스터 식별자	리전 간 읽기 전용 복제본 DB 클러스터의 이름을 입력합니다. 이 이름은 복제본의 대상 AWS 리전에서 사용자 계정에 고유한 이름이어야 합니다. 이 식별자는 DB 클러스터에 대한 클러스터 엔드포인트 주소로 사용됩니다. 클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora 엔드포인트 연결 단원을 참조하십시오. DB 클러스터 식별자는 다음과 같은 제약 조건이 있습니다. • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • 개별 AWS 계정별로 각 AWS 리전의 모든 DB 클러스터에 대해 고유해야 합니다.
우선 순위	새 DB 클러스터의 기본 인스턴스에 대한 장애 조치의 우선 순위를 선택합니다. 기본 인스턴스의 결함으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.
데이터베이스 포트	애플리케이션과 유틸리티가 데이터베이스에 액세스할 때 사용할 포트를 지정합니다. Aurora DB 클러스터는 기본적으로 MySQL 포트, 3306으로 지정됩니다. 일부 기업에서는 방화벽으로 이 포트에 대한 연결을 차단합니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 새로운 DB 클러스터에 다른 포트를 선택해야 합니다.

옵션	설명
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 설정하려면 향상된 모니터링 사용 설정(Enable enhanced monitoring)을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 단원을 참조하십시오.
역할 모니터링	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. Amazon RDS가 Amazon CloudWatch Logs와 통신할 수 있도록 생성한 IAM 역할을 선택하거나 RDS가 `rds-monitoring-role` 라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.
세부 수준	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
마이너 버전 자동 업그레이드	이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

7. Aurora의 리전 간 읽기 전용 복제본을 만들려면 생성을 선택합니다.

AWS CLI

CLI를 사용하여 리전 간 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들려면

1. 읽기 전용 복제본 DB 클러스터를 만들려는 AWS 리전에서 AWS CLI create-db-cluster 명령을 호출합니다. `--replication-source-identifier` 옵션을 포함시키고, 읽기 전용 복제본을 만들 원본 DB 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다.

   `--replication-source-identifier`로 식별되는 DB 클러스터가 암호화되는 교차 리전 복제의 경우 `--kms-key-id` 옵션 및 `--storage-encrypted` 옵션을 지정합니다.

> **Note**
>
> `--storage-encrypted`를 지정하고 `--kms-key-id`의 값을 제공하여 암호화되지 않은 DB 클러스터에서 암호화된 읽기 전용 복제본으로 리전 간 복제를 설정할 수 있습니다.

`--master-username` 및 `--master-user-password` 파라미터는 지정할 수 없습니다. 이러한 값은 원본 DB 클러스터에서 가져옵니다.

다음 코드 예에서는 us-west-2 리전의 암호화되지 않은 DB 클러스터 스냅샷에서 us-east-1 리전에 읽기 전용 복제본을 만듭니다. 이 명령은 us-east-1 리전에서 호출됩니다. 이 예제에서는 마스터 사용자 암호를 생성하고 이를 Secrets Manager에서 관리하는 `--manage-master-user-password` 옵션을 지정합니다. 자세한 내용은 [Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리](#) 단원을 참조하십시오. 또는 `--master-password` 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Windows의 경우:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

다음 코드 예에서는 us-west-2 리전의 암호화된 DB 클러스터 스냅샷에서 us-east-1 리전에 읽기 전용 복제본을 만듭니다. 이 명령은 us-east-1 리전에서 호출됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

Windows의 경우:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

이 `--source-region` 옵션은 `--replication-source-identifier`에서 식별된 DB 클러스터가 암호화되는 AWS GovCloud(미국 동부)와 AWS GovCloud(미국 서부) 리전 간의 교차 리전 복제에 필요합니다. `--source-region`의 경우 소스 DB 클러스터의 AWS 리전을 지정합니다.

`--source-region`이 지정되지 않은 경우에는 `--pre-signed-url` 값을 지정합니다. 미리 서명된 URL은 소스 AWS 리전에서 호출되는 `create-db-cluster` 명령에 대한 서명 버전 4의 서명된 요청이 포함된 URL입니다. `pre-signed-url` 옵션에 대한 자세한 정보는 AWS CLI 명령 참조에서 create-db-cluster를 참조하세요.

2. 다음 예제와 같이 AWS CLI describe-db-clusters 명령을 사용하여 DB 클러스터를 사용할 수 있는지 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

`describe-db-clusters` 결과에 상태가 `available`로 표시되면 DB 클러스터의 기본 인스턴스를 만들어 복제를 시작합니다. 이렇게 하려면 다음 예제에서와 같이 AWS CLI create-db-instance 명령을 사용하면 됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r5.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora-mysql
```

Windows의 경우:

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r5.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora-mysql
```

DB 인스턴스가 생성되어 사용할 수 있게 되면 복제가 시작됩니다. AWS CLI create-db-instance 명령을 호출하여 DB 인스턴스를 사용할 수 있는지를 파악할 수 있습니다.

RDS API

API를 사용하여 리전 간 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들려면

1. 읽기 전용 복제본 DB 클러스터를 만들려는 AWS 리전에서 RDS API CreateDBCluster 작업을 호출합니다. `ReplicationSourceIdentifier` 파라미터를 포함시키고, 읽기 전용 복제본을 만들 원본 DB 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다.

   `ReplicationSourceIdentifier`로 식별되는 DB 클러스터가 암호화되는 교차 리전 복제의 경우 `KmsKeyId` 파라미터를 지정하고 `StorageEncrypted` 파라미터를 `true`로 설정합니다.

   > **Note**
   >
   > `StorageEncrypted`를 **`true`**로 지정하고 `KmsKeyId`의 값을 제공하여 암호화되지 않은 DB 클러스터에서 암호화된 읽기 전용 복제본으로 리전 간 복제를 설정할 수 있습니다. 이 경우 `PreSignedUrl`을 지정하지 않아도 됩니다.

   `MasterUsername` 및 `MasterUserPassword` 파라미터 값은 원본 DB 클러스터에서 가져오므로 포함시키지 않아도 됩니다.

다음 코드 예에서는 us-west-2 리전의 암호화되지 않은 DB 클러스터 스냅샷에서 us-east-1 리전에 읽기 전용 복제본을 만듭니다. 이 작업은 us-east-1 리전에서 호출됩니다.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

다음 코드 예에서는 us-west-2 리전의 암호화된 DB 클러스터 스냅샷에서 us-east-1 리전에 읽기 전용 복제본을 만듭니다. 이 작업은 us-east-1 리전에서 호출됩니다.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
```

```
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

ReplicationSourceIdentifier로 식별되는 DB 클러스터가 암호화되는 AWS GovCloud(미국 동부)와 AWS GovCloud(미국 서부) 리전 간의 교차 리전 복제의 경우에도 PreSignedUrl 파라미터를 지정합니다. 미리 서명된 URL은 복제할 암호화된 DB 클러스터가 포함된 소스 AWS 리전에서 수행할 수 있는 CreateDBCluster API 작업에 대한 유효한 요청이어야 합니다. KMS 키 식별자는 읽기 전용 복제본을 암호화하는 데 사용되며, 대상 AWS 리전에 대해 유효한 KMS 키여야 합니다. 미리 서명된 URL을 수동이 아닌 자동으로 생성하려면 --source-region 옵션 대신 AWS CLI create-db-cluster 명령을 사용합니다.

2. 다음 예제와 같이 RDS API DescribeDBClusters 작업을 사용하여 DB 클러스터를 사용할 수 있는지 확인합니다.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

DescribeDBClusters 결과에 상태가 available로 표시되면 DB 클러스터의 프라이머리 인스턴스를 만들어 복제를 시작합니다. 그러려면 다음 예제와 같이 RDS API CreateDBInstance 작업을 사용하세요.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r5.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T003808Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

DB 인스턴스가 생성되어 사용할 수 있게 되면 복제가 시작됩니다. AWS CLI DescribeDBInstances 명령을 호출하여 DB 인스턴스를 사용할 수 있는지를 파악할 수 있습니다.

Amazon Aurora MySQL 리전 간 복제본 보기

describe-db-clusters AWS CLI 명령 또는 DescribeDBClusters RDS API 작업을 호출하여 Amazon Aurora MySQL DB 클러스터에 대한 교차 리전 복제 관계를 확인할 수 있습니다. 응답에서 교차 리전 읽기 복제본 DB 클러스터의 DB 클러스터 식별자는 `ReadReplicaIdentifiers` 필드에서 확인하세요. 복제 소스인 소스 DB 클러스터의 ARN은 `ReplicationSourceIdentifier` 요소를 참조하세요.

Aurora MySQL의 읽기 전용 복제본을 DB 클러스터로 승격

Aurora MySQL 읽기 전용 복제본을 독립형 DB 클러스터로 승격할 수 있습니다. Aurora MySQL 읽기 전용 복제본을 승격하면 DB 인스턴스가 재부팅된 후에 사용 가능하게 됩니다.

일반적으로 소스 DB 클러스터가 실패할 경우 데이터 복구 체계로서 Aurora MySQL 읽기 전용 복제본을 독립형 DB 클러스터로 승격합니다.

이렇게 하려면 먼저 읽기 전용 복제본을 생성한 다음 원본 DB 클러스터에 장애가 있는지 모니터링합니다. 그 결과 장애가 발견된 경우에는 다음과 같이 실행합니다.

1. 읽기 전용 복제본을 승격합니다.
2. 데이터베이스 트래픽을 승격된 DB 클러스터로 유도합니다.
3. 승격된 DB 클러스터를 원본으로 하는 교체용 읽기 전용 복제본을 생성합니다.

읽기 전용 복제본을 승격하면 읽기 전용 복제본은 독립형 Aurora DB 클러스터가 됩니다. 승격 프로세스는 읽기 전용 복제본의 크기에 따라 완료하는 데 몇 분 또는 더 오래도 걸릴 수 있습니다. 읽기 전용 복제본이 새 DB 클러스터로 승격된 후에는 다른 DB 인스턴스와 똑같습니다. 예를 들어, 해당 클러스터에서 읽기 전용 복제본을 생성하고 특정 시점으로 복구 작업을 수행할 수 있습니다. 또한 DB 클러스터의 Aurora 복제본을 생성할 수 있습니다.

승격된 DB 클러스터는 더 이상 읽기 전용 복제본이 아니기 때문에 복제 대상으로 사용할 수 없습니다.

다음 단계에서는 읽기 전용 복제본을 DB 클러스터로 승격하기 위한 일반적인 프로세스를 보여줍니다.

1. 트랜잭션이 읽기 전용 복제본 소스 DB 클러스터에 더 이상 기록되지 않도록 한 다음, 읽기 전용 복제본의 업데이트가 모두 끝날 때까지 기다립니다. 읽기 전용 복제본의 데이터베이스 업데이트는 원본 DB 클러스터의 업데이트 후에 수행되며, 이러한 복제 지연은 경우에 따라 크게 다를 수 있습니다. `ReplicaLag` 지표를 사용하여 읽기 전용 복제본의 업데이트가 모두 완료되는 시간을 측정합니다. `ReplicaLag` 지표는 소스 DB 인스턴스를 기준으로 읽기 전용 복제본 DB 인스턴스의 지연 시간을 기록합니다. `ReplicaLag` 지표가 `0`에 도달하면 읽기 전용 복제본이 원본 DB 인스턴스를 따라잡은 것입니다.
2. Amazon RDS 콘솔에서 승격(Promote) 옵션을 사용하거나, AWS CLI 명령 [promote-read-replica-db-cluster]를 사용하거나, [PromoteReadReplicaDBCluster] Amazon RDS API 작업을 사용하여 읽기 전용 복제본을 승격합니다.

   읽기 전용 복제본을 승격할 Aurora MySQL DB 인스턴스를 선택합니다. 읽기 전용 복제본이 승격된 후 Aurora MySQL DB 클러스터가 독립형 DB 클러스터로 승격됩니다. 장애 조치 우선 순위가 가장 높은 DB 인스턴스가 DB 클러스터의 기본 DB 인스턴스로 승격됩니다. 다른 DB 인스턴스는 Aurora 복제본이 됩니다.

> **Note**
>
> 승격 프로세스는 완료할 때까지 몇 분 걸립니다. 읽기 전용 복제본을 승격하면 복제가 중지되고 DB 인스턴스가 재부팅됩니다. 재부팅이 완료되면 읽기 전용 복제본을 새 DB 클러스터로 사용할 수 있습니다.

콘솔

Aurora MySQL 읽기 전용 복제본을 DB 클러스터로 승격하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 콘솔에서 인스턴스를 선택합니다.

   인스턴스 창이 표시됩니다.
3. 인스턴스 창에서 승격하려는 읽기 전용 복제본을 선택합니다.

   읽기 전용 복제본이 Aurora MySQL DB 인스턴스로 나타납니다.
4. 작업에서 읽기 전용 복제본 승격을 선택합니다.
5. 승인 페이지에서 Promote Read Replica를 선택합니다.

AWS CLI

읽기 전용 복제본을 DB 클러스터로 승격하려면 AWS CLI promote-read-replica-db-cluster 명령을 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

Windows의 경우:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS API

읽기 전용 복제본을 DB 클러스터로 승격하려면 PromoteReadReplicaDBCluster를 호출합니다.

Amazon Aurora MySQL의 교차 리전 복제본 문제 해결

Amazon Aurora 리전 간 읽기 전용 복제본을 만들 때 발생할 수 있는 일반적인 오류 메시지와 지정된 오류의 해결 방법이 아래에 목록으로 정리되어 있습니다.

원본 클러스터 [DB 클러스터 ARN]의 binlog가 활성화되어 있지 않음

이 문제를 해결하려면 원본 DB 클러스터에 대한 이진 로깅을 설정합니다. 자세한 내용은 시작하기 전 준비 사항 단원을 참조하십시오.

원본 클러스터 [DB 클러스터 ARN]에 라이터에서 동기화된 클러스터 파라미터 그룹이 없음

`binlog_format` DB 클러스터 파라미터를 업데이트했지만 DB 클러스터의 기본 인스턴스를 재부팅하지 않은 경우 이 오류가 발생합니다. DB 클러스터의 기본 인스턴스(라이터)를 재부팅하고 다시 시도하십시오.

원본 클러스터 [DB 클러스터 ARN]의 읽기 전용 복제본이 이미 이 리전에 있음

모든 AWS 리전에서 소스 DB 클러스터당 최대 5개의 교차 리전 DB 클러스터(읽기 전용 복제본)를 만들 수 있습니다. 특정 AWS 리전에 DB 클러스터에 대한 읽기 전용 복제본의 최대 개수가 이미 있는 경우, 해당 리전에서 교차 리전 DB 클러스터를 새로 생성하기 전에 기존 읽기 전용 복제본을 삭제해야 합니다.

DB 클러스터 [DB 클러스터 ARN]에서 리전 간 복제를 지원하려면 데이터베이스 엔진 업그레이드 필요

이 문제를 해결하려면 원본 DB 클러스터의 모든 인스턴스에 대해 데이터베이스 엔진 버전을 최신 데이터베이스 엔진 버전으로 업그레이드한 다음, 리전 간 읽기 전용 복제본 DB를 다시 만들어 보십시오.

Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제

Amazon Aurora MySQL는 MySQL과 호환되기 때문에 MySQL 데이터베이스와 Amazon Aurora MySQL DB 클러스터 사이에 복제를 설정할 수 있습니다. 이러한 유형의 복제는 MySQL 이진 로그 복제를 사용하며 binlog 복제라고 합니다. Aurora에서 이진 로그 복제를 사용하는 경우 MySQL 데이터베이스에서 MySQL 버전 5.5 이상을 실행하는 것이 좋습니다. Aurora MySQL DB 클러스터가 복제 소스 또는 복제본인 경우 복제를 설정할 수 있습니다. Amazon RDS MySQL DB 인스턴스, Amazon RDS 외부의 MySQL 데이터베이스 또는 다른 Aurora MySQL DB 클러스터를 사용하여 복제할 수 있습니다.

> **Note**
>
> 특정 유형의 Aurora DB 클러스터에서 안팎으로의 binlog 복제를 사용할 수 없습니다. 구체적으로는 Aurora Serverless v1 클러스터에 binlog 복제를 사용할 수 없습니다. `SHOW MASTER STATUS` 및 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 문이 출력을 반환하지 않는 경우 사용 중인 클러스터가 binlog 복제를 지원하는지 확인하세요.

다른 AWS 리전의 RDS for MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터를 사용하여 복제할 수도 있습니다. AWS 리전 간 복제를 수행할 경우 DB 클러스터와 DB 인스턴스에 공개 액세스가 가능

한지 확인합니다. Aurora MySQL DB 클러스터가 VPC의 프라이빗 서브넷에 있는 경우 AWS 리전 간에 VPC 피어링을 사용합니다. 자세한 내용은 VPC에 있는 DB 클러스터에 다른 VPC에 있는 EC2 인스턴스가 액세스 단원을 참조하십시오.

Aurora MySQL DB 클러스터와 다른 AWS 리전의 Aurora MySQL DB 클러스터 간에 복제를 구성하려면 소스 DB 클러스터와 다른 AWS 리전에 Aurora MySQL DB 클러스터를 읽기 전용 복제본으로 생성합니다. 자세한 내용은 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 단원을 참조하십시오.

Aurora MySQL 버전 2 및 3에서는 Aurora MySQL과 외부 원본 또는 복제를 위해 전역 트랜잭션 식별자(GTID)를 사용하는 대상 간에 복제 작업을 할 수 있습니다. Aurora MySQL DB 클러스터에 있는 GTID 관련 파라미터에 외부 데이터베이스의 GTID 상태와 호환되는 설정이 있어야 합니다. 이 작업을 수행하는 방법은 GTID 기반 복제 사용 단원을 참조하세요. Aurora MySQL 버전 3.01 이상에서는 GTID를 사용하지 않는 소스에서 복제되는 트랜잭션에 GTID를 할당하는 방법을 선택할 수 있습니다. 해당 설정을 제어하는 저장 프로시저에 대한 자세한 내용은 mysql.rds_assign_gtids_to_anonymous_transactions(Aurora MySQL 버전 3) 섹션을 참조하세요.

> **Warning**
>
> Aurora MySQL과 MySQL 간에 복제할 경우 InnoDB 테이블만 사용해야 합니다. 복제할 MyISAM 테이블이 있는 경우 다음 명령을 사용하여 복제를 설정하기 전에 해당 테이블을 InnoDB로 변환할 수 있습니다.
>
> ```
> alter table <schema>.<table_name> engine=innodb, algorithm=copy;
> ```

다음 섹션에서는 복제 설정, 복제 중지, 데이터베이스 읽기 규모 조정, binlog 복제 최적화, 향상된 binlog 설정을 설명합니다.

주제

- Aurora MySQL의 이진수 로그 복제 설정
- Aurora MySQL의 이진수 로그 복제 중지
- Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 규모 조정
- Aurora MySQL의 이진수 로그 복제 최적화
- Aurora MySQL에 대해 향상된 binlog 설정

Aurora MySQL의 이진수 로그 복제 설정

Aurora MySQL을 사용하여 MySQL 복제를 설정하는 단계는 다음에서 자세히 설명합니다.

목차

1. 복제 소스에 대한 이진 로깅 설정

다음 데이터베이스 엔진의 복제 소스에 대한 이진 로깅을 설정하는 방법의 지침을 확인합니다.

데이터베이스 엔진	지침
Aurora MySQL	Aurora MySQL DB 클러스터에 대한 이진 로깅을 설정하려면 `binlog_format` DB 클러스터 파라미터를 `ROW`, `STATEMENT` , `MIXED`로 설정합니다. 특정 binlog 형식이 필요하지 않는 한 `MIXED`로 설정하는 것이 좋습니다. (기본값은 `OFF`입니다.) `binlog_format` 파라미터를 변경하려면 사용자 지정 DB 클러스터 파라미터 그룹을 만들고 해당 사용자 지정 파라미터 그룹을 DB 클러스터와 연결합니다. 기본 DB 클러스터 파라미터 그룹에서는 파라미터를 변경할 수 없습니다. `binlog_format` 파라미터를 `OFF`에서 다른 값으로 변경하는 경우, Aurora DB 클러스터를 재부팅해야 변경 사항이 적용됩니다. 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 및 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.

데이터베이스 엔진	지침
RDS for MySQL	Amazon RDS DB 인스턴스에 대한 이진 로깅을 설정하려면 Amazon RDS DB 인스턴스에 대한 이진 로깅을 직접 설정할 수 없지만, 다음 중 하나를 수행하여 설정할 수 있습니다. • DB 인스턴스의 자동 백업을 설정합니다. DB 인스턴스를 생성할 때 자동 백업을 설정하거나 기존 DB 인스턴스를 수정하여 백업을 설정할 수 있습니다. 자세한 내용은 Amazon RDS 사용 설명서의 DB 인스턴스 생성 단원을 참조하세요. • DB 인스턴스의 읽기 전용 복제본을 생성합니다. 자세한 내용은 Amazon RDS 사용 설명서의 읽기 전용 복제본 작업을 참조하십시오.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>MySQL(외부)</td><td>
암호화된 복제 설정
Aurora MySQL 버전 2를 사용하여 데이터를 안전하게 복제하려면 암호화된 복제를 사용하세요.
<blockquote> Note 암호화된 복제를 사용할 필요가 없다면 이 단계를 건너 뛸 수 있습니다. </blockquote>
다음은 암호화된 복제를 사용하기 위한 사전 조건입니다.

외부 MySQL 소스 데이터베이스에서 SSL(Secure Socket Layer)를 활성화해야 합니다.
Aurora MySQL DB 클러스터에 대해 클라이언트 키와 클라이언트 인증서를 준비해야 합니다.

암호화 복제 중, Aurora MySQL DB 클러스터는 MySQL 데이터베이스 서버의 클라이언트 역할을 합니다. Aurora MySQL 클라이언트 인증서와 키는 .pem 형식의 파일입니다.

암호화 복제를 준비해야 합니다.

외부 MySQL 소스 데이터베이스에서 SSL을 설정하지 않고 클라이언트 키와 클라이언트 인증서가 준비되지 않은 경우 MySQL 데이터베이스 서버의 SSL을 설정하고 필요한 클라이언트 키와 클라이언트 인증서를 생성합니다.
외부 소스에 SSL이 활성화되어 있으면 Aurora MySQL DB 클러스터에 클라이언트 키와 인증서를 제공합니다. 인증서와 키가 없다면, Aurora MySQL DB 클러스터에 대해 새 키와 인증서를 생성합니다. 클라이언트 인증서에 서명하려면, 외부 MySQL 소스 데이터베이스의 SSL을 구성할 때 사용하는 인증 기관(CA) 키가 있어야 합니다.

자세한 내용은 MySQL 설명서의 openssl을 사용하여 SSL 인증서 및 키 생성을 참조하십시오.

</td></tr>
</table>

데이터베이스 엔진	지침
	인증 기관(CA) 인증서, 클라이언트 키, 클라이언트 인증서가 필요합니다. 2. SSL을 사용하여 마스터 사용자로 Aurora MySQL DB 클러스터에 연결하십시오. SSL을 이용한 Aurora MySQL DB 클러스터 연결에 대한 자세한 정보는 [Aurora MySQL DB 클러스터에 대한 TLS 연결]을 참조하십시오. 3. `mysql.rds_import_binlog_ssl_material` 저장 프로시저를 실행하여 Aurora MySQL DB 클러스터로 SSL 정보를 가져오십시오. `ssl_material_value` 파라미터에서, Aurora MySQL DB 클러스터의 정보를 올바른 JSON 페이로드에 삽입하십시오. 다음은 Aurora MySQL DB 클러스터에 SSL 정보를 가져오는 예제입니다. .pem 형식 파일은 본문 코드의 길이가 일반적으로 예제의 본문 코드 길이보다 깁니다.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
```

데이터베이스 엔진	지침

```
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS706V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

자세한 내용은 mysql.rds_import_binlog_ssl_material 및 Aurora MySQL DB 클러스터에 대한 TLS 연결 단원을 참조하세요.

> **Note**
>
> 프로시저를 실행한 후 파일에 암호를 저장합니다. 이 파일을 나중에 삭제하려면 mysql.rds_remove_binlog_ssl_material 저장 프로시저를 실행하면 됩니다.

외부 MySQL 데이터베이스에 대한 이진 로깅을 설정하려면

1. 명령 셸에서 mysql 서비스를 중지합니다.

```
sudo service mysqld stop
```

2. `my.cnf` 파일을 편집합니다(이 파일은 보통 `/etc`에 있음).

```
sudo vi /etc/my.cnf
```

`log_bin` 및 `server_id` 옵션을 `[mysqld]` 섹션에 추가합니다. `log_bin` 옵션은 이진 로그 파일에 대한 파일 이름 식별자를 제공합니다. `server_id` 옵션은 소스-복제본 관계에서 서버의 고유 식별자를 제공합니다.

암호화된 복제가 필요 없는 경우 binlog를 활성화하고 SSL을 설정하여 외부 MySQL 데이터베이스를 시작합니다.

데이터베이스 엔진	지침
	다음은 암호화된 데이터와 관련된 `/etc/my.cnf` 파일 항목들입니다.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

암호화된 복제가 필요하다면, SSL과 binlogs를 활성화시켜 외부 MySQL 데이터베이스를 시작합니다.

`/etc/my.cnf` 파일 항목에는 MySQL 데이터베이스 서버의 .pem 파일 위치가 포함되어 있습니다.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

또한 MySQL DB 인스턴스의 `sql_mode` 옵션을 0으로 설정하거나, my.cnf 파일에 이 옵션이 포함되어서는 안 됩니다.

외부 MySQL 데이터베이스 레코드에 연결되어 있는 동안 외부 MySQL 데이터베이스의 이진수 로그 위치를 기록합니다.

```
mysql> SHOW MASTER STATUS;
```

다음과 유사하게 출력되어야 합니다:

```
+------------------+----------+--------------+------------------
+-------------------+
```

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td></td><td>

```
| File              | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+-------------------+----------+--------------+------------------
+--------------------+
| mysql-bin.000031  |      107 |              |                  |
              |
+-------------------+----------+--------------+------------------
+--------------------+
1 row in set (0.00 sec)
```

자세한 내용은 MySQL 설명서에서 Setting the replication source configuration 섹션을 참조하세요.

3. mysql 서비스를 시작합니다.

```
sudo service mysqld start
```

</td></tr>
</table>

2. 더 이상 필요 없을 때까지 이진 로그를 복제 소스에 보관

MySQL 이진 로그 복제를 사용할 경우 Amazon RDS에서 복제 프로세스를 관리하지 않습니다. 따라서 변경 사항이 복제본에 적용된 이후까지 복제 소스의 binlog 파일이 보관되는지 확인해야 합니다. 이 유지 관리는 오류 발생 시 원본 데이터베이스를 복원하는 데 도움이 됩니다.

데이터베이스 엔진에 대한 바이너리 로그를 유지하려면 다음 지침을 따르세요.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>Aurora MySQL</td><td>

Aurora MySQL DB 클러스터에 이진 로그를 보관하려면

Aurora MySQL DB 클러스터의 binlog 파일에 대한 액세스 권한이 없습니다. 따라서 Amazon RDS에서 binlog 파일을 삭제하기 이전에 변경 사항이 복제본에 적용되도록 복제 소스에 binlog 파일을 보관할 기간을 충분히 길게 선택해야 합니다. Aurora MySQL DB 클러스터에 binlog 파일을 최대 90일 동안 보관할 수 있습니다.

MySQL 데이터베이스 또는 RDS for MySQL DB 인스턴스를 복제본으로 사용하여 복제를 설정하고 복제본을 생성할 데이터베이스가 매우 큰 경우, 복제본에 대한 데이터베

</td></tr>
</table>

데이터베이스 엔진	지침
	이스의 초기 복사가 완료되고 복제 지연이 0에 도달할 때까지 binlog 파일을 보관하도록 기간을 길게 선택합니다. 바이너리 로그 보관 기간을 설정하려면 mysql.rds_set_configuration 프로시저를 사용하여 `'binlog retention hours'` 구성 파라미터와 함께 DB 클러스터에 binlog 파일을 보관할 시간을 지정하세요. Aurora MySQL 버전 2.11.0 이상 및 버전 3의 최대 값은 2160(90일)입니다. 다음 예에서는 binlog 파일의 보관 기간을 6일로 설정합니다. `CALL mysql.rds_set_configuration('binlog retention hours', 144);` 복제를 시작한 후 복제본에 대해 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL version 3) 명령을 실행하고 `Seconds behind master` 필드를 확인하여 변경 사항이 복제본에 적용되었는지 확인할 수 있습니다. `Seconds behind master` 필드가 0이면 복제본 지연이 없습니다. 복제 지연이 없는 경우 `binlog retention hours` 구성 파라미터를 더 짧은 기간으로 설정하여 binlog 파일을 보관할 기간을 줄입니다. 이 설정을 지정하지 않으면 Aurora MySQL의 기본값은 24(1일)입니다. `'binlog retention hours'` 값을 최대값보다 높게 지정하면 Aurora MySQL은 해당 최대값을 사용합니다.
RDS for MySQL	Amazon RDS DB 인스턴스에 이진 로그를 보관하려면 이전 행에서 설명한 Aurora MySQL DB 클러스터와 마찬가지로 binlog 보관 시간을 설정하여 Amazon RDS DB 인스턴스에 바이너리 로그 파일을 보관할 수 있습니다. DB 인스턴스에 대한 읽기 전용 복제본을 생성하여 Amazon RDS DB 인스턴스에 binlog 파일을 보관할 수도 있습니다. 이 읽기 전용 복제본은 임시 복제본이며 binlog 파일 보관의 목적으로만 사용됩니다. 읽기 전용 복제본이 생성된 후 읽기 전용 복제본에서 mysql.rds_stop_replication 프로시저를 호출합니다. 복제가 중지된 동안 Amazon RDS에서는 복제 소스에서 binlog 파일을 삭제하지 않습니다. 영구 복제본을 사용하여 복제를 설정한 후 복제 소스와 영구 복제본 사이의 복제 지연(`Seconds behind master` 필드)이 0에 도달한 경우 읽기 전용 복제본을 삭제할 수 있습니다.

데이터베이스 엔진	지침
MySQL(외부)	외부 MySQL 데이터베이스에 이진 로그를 보관하려면 외부 MySQL 데이터베이스의 binlog 파일은 Amazon RDS에서 관리하지 않으므로 삭제할 때까지 보관됩니다. 복제를 시작한 후 복제본에 대해 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL version 3) 명령을 실행하고 `Seconds behind master` 필드를 확인하여 변경 사항이 복제본에 적용되었는지 확인할 수 있습니다. `Seconds behind master` 필드가 0이면 복제본 지연이 없습니다. 복제 지연이 없는 경우 이전 binlog 파일을 삭제할 수 있습니다.

3. 복제 소스의 사본 또는 덤프 만들기

복제 소스의 스냅샷, 클론 또는 덤프는 데이터의 기본 사본을 복제본으로 로드하는 데 사용됩니다. 그런 다음 그 시점부터 복제를 시작합니다.

다음 지침에 따라 데이터베이스 엔진에 대한 복제 소스의 사본 또는 덤프를 생성하세요.

데이터베이스 엔진	지침
Aurora MySQL	Aurora MySQL DB 클러스터의 사본을 만들려면 다음 방법 중 한 가지를 선택하세요. • 스냅샷에서 DB 클러스터를 복원합니다. 1. Amazon Aurora DB 클러스터의 DB 클러스터 스냅샷을 생성합니다. 자세한 내용은 DB 클러스터 스냅샷 생성 섹션을 참조하세요. 2. 방금 만든 DB 클러스터 스냅샷에서 복원하여 새 Aurora DB 클러스터를 생성합니다. 복원된 DB 클러스터의 DB 파라미터 그룹을 원래 DB 클러스터와 동일하게 유지해야 합니다. 이렇게 해야 DB 클러스터 사본에 이진수 로깅이 활성화됩니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 단원을 참조하십시오.

데이터베이스 엔진	지침

	• DB 클러스터를 복제합니다. 자세한 내용은 Aurora DB 클러스터에 대한 볼륨 복제 단원을 참조하십시오.
	
	binlog 파일 이름과 위치를 확인하려면
	
	다음 방법 중 한 가지를 선택하세요.
	
	• AWS Management Console:
	1. 데이터베이스를 선택하고 새로운 Aurora DB 클러스터의 기본 인스턴스(라이터)를 선택하여 세부 정보를 표시합니다.
	2. [Recent Events]로 스크롤합니다. binlog 파일 이름과 위치를 포함한 이벤트 메시지가 표시됩니다. 이벤트 메시지의 형식은 다음과 같습니다.
	`Binlog position from crash recovery is binlog-file-name  binlog-position`
	3. 복제 시작 위치에 해당하는 binlog 파일 이름 및 위치 값을 저장합니다.
	• 다음 예제와 같이 describe-events AWS CLI 명령을 직접적으로 호출합니다.
	`aws rds describe-events`
	`{ "Events": [{ "EventCategories": [], "SourceType": "db-instance", "SourceArn": "arn:aws:rds:us-west-2:123456789012:db:sample-restored-instance", "Date": "2016-10-28T19:43:46.862Z", "Message": "Binlog position from crash recovery is mysql-bin-changelog.000003 4278", "SourceIdentifier": "sample-restored-instance" }] }`
	• MySQL 오류 로그에서 마지막 MySQL binlog 파일 위치를 확인합니다.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td></td><td>
Aurora MySQL DB 클러스터의 덤프를 만들려면
복제본 대상이 외부 MySQL 데이터베이스 또는 RDS for MySQL DB 인스턴스인 경우, Aurora DB 클러스터에서 덤프 파일을 만들어야 합니다.
생성한 소스 DB 클러스터 사본에 대해 mysqldump 명령을 실행해야 합니다. 이는 덤프 시 잠금 고려 사항을 피하기 위한 것입니다. 원본 DB 클러스터에서 직접 덤프를 수행한 경우 덤프가 진행되는 동안 원본 테이블에 동시 쓰기가 발생하지 않도록 원본 테이블을 잠가야 합니다.

MySQL 클라이언트를 사용하여 DB 클러스터에 연결
mysqldump 명령을 실행합니다. 예시:
<pre>PROMPT> mysqldump --databases database_name --single-transaction
--order-by-primary -r backup.sql -u local_user s -p</pre>
덤프 파일을 만든 후 DB 클러스터 사본을 삭제할 수 있습니다.

</td></tr>
<tr><td>RDS for MySQL</td><td>
Amazon RDS DB 인스턴스의 스냅샷을 만들려면
Amazon RDS DB 인스턴스의 읽기 전용 복제본을 생성합니다. 자세한 내용은 Amazon Relational Database Service 사용 설명서의 읽기 전용 복제본 생성 단원을 참조하십시오.

읽기 전용 복제본에 연결하고 mysql.rds_stop_replication 프로시저를 실행하여 복제를 중지합니다.
읽기 전용 복제본이 중지된 동안 읽기 전용 복제본에 연결하고 SHOW SLAVE STATUS(Aurora MySQL 버전 2) 또는 SHOW REPLICA STATUS(Aurora MySQL 버전 3) 명령을 실행합니다. Relay_Master_Log_File 필드에서 현재 이진 로그 파일 이름을 검색하고 Exec_Master_Log_Pos 필드에서 로그 파일 위치를 검색합니다. 복제를 시작할 때에 대비하여 해당 값을 저장합니다.
읽기 전용 복제본을 중지됨 상태로 유지하면서 읽기 전용 복제본의 DB 스냅샷을 생성합니다. 자세한 내용은 Amazon Relational Database Service 사용 설명서의 DB 스냅샷 생성 단원을 참조하십시오.
읽기 전용 복제본을 삭제합니다.

</td></tr>
</table>

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>MySQL(외부)</td><td>

외부 MySQL 데이터베이스의 스냅샷 생성

1. 덤프를 생성하기 전에 덤프의 binlog 위치가 소스 인스턴스의 최신 데이터와 일치하는지 확인해야 합니다. 이렇게 하려면 먼저 다음 명령을 사용하여 인스턴스에 대한 쓰기 작업을 중지해야 합니다.

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. 다음과 같이 `mysqldump` 명령을 사용하여 MySQL 데이터베이스의 덤프를 생성합니다.

```
PROMPT> sudo mysqldump --databases database_name --master-data=2 --
single-transaction \
--order-by-primary -r backup.sql -u local_user -p
```

3. 덤프를 생성한 후 다음 명령을 사용하여 MySQL 데이터베이스에서 테이블의 잠금을 해제합니다.

```
mysql> UNLOCK TABLES;
```

</td></tr>
</table>

4. 복제본 대상으로 덤프를 로드합니다(필요한 경우).

Amazon RDS 외부에 있는 MySQL 데이터베이스 덤프에서 데이터를 로드할 계획이라면 덤프 파일을 복사할 EC2 인스턴스를 만드는 것이 좋습니다. 그런 다음 해당 EC2 인스턴스에서 DB 클러스터 또는 DB 인스턴스로 데이터를 로드할 수 있습니다. 이 방법을 사용하면 EC2 인스턴스에 덤프 파일을 복사하기 전에 압축하여 Amazon RDS에 데이터를 복사하는 것과 관련한 네트워크 비용을 줄일 수 있습니다. 또한 덤프 파일 또는 파일을 암호화하여 네트워크 간에 전송되는 데이터를 보호할 수 있습니다.

Note

복제본 대상으로 새로운 Aurora MySQL DB 클러스터를 만드는 경우 덤프 파일을 로드할 필요가 없습니다.

- DB 클러스터 스냅샷에서 복원하여 새로운 DB 클러스터를 만들 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 단원을 참조하십시오.

- 소스 DB 클러스터를 복제하여 새 DB 클러스터를 생성할 수 있습니다. 자세한 내용은 Aurora DB 클러스터에 대한 볼륨 복제 단원을 참조하십시오.
- DB 인스턴스의 데이터를 새 DB 클러스터로 마이그레이션할 수 있습니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 단원을 참조하십시오.

데이터베이스 엔진의 복제본 대상에 복제 소스의 덤프를 로드하는 방법에 대한 다음 지침을 확인하세요.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>Aurora MySQL</td><td>Aurora MySQL DB 클러스터로 덤프 로드

1. mysqldump 명령의 출력을 복제 소스에서 Aurora MySQL DB 클러스터에도 연결 가능한 위치로 복사합니다.
2. mysql 명령을 사용하여 Aurora MySQL DB 클러스터에 연결합니다. 다음은 예제입니다.
<pre>PROMPT> mysql -h host_name -port=3306 -u db_master_user -p</pre>3. mysql 프롬프트에서 source 명령을 실행하고 데이터베이스 덤프 파일의 이름을 전달하여 Aurora MySQL DB 클러스터로 데이터를 로드합니다. 예를 들면 다음과 같습니다.
<pre>mysql> source backup.sql;</pre></td></tr>
<tr><td>RDS for MySQL</td><td>Amazon RDS DB 인스턴스로 덤프 로드

1. mysqldump 명령의 출력을 복제 소스에서 MySQL DB 인스턴스에도 연결 가능한 위치로 복사합니다.
2. mysql 명령을 사용하여 MySQL DB 인스턴스에 연결합니다. 다음은 예제입니다.
<pre>PROMPT> mysql -h host_name -port=3306 -u db_master_user -p</pre>3. mysql 프롬프트에서 source 명령을 실행하고 데이터베이스 덤프 파일의 이름을 전달하여 MySQL DB 인스턴스로 데이터를 로드합니다. 예를 들면 다음과 같습니다.</td></tr>
</table>

데이터베이스 엔진	지침
	`mysql> source backup.sql;`

MySQL(외부)	외부 MySQL 데이터베이스로 덤프 로드
	
	DB 스냅샷 또는 DB 클러스터 스냅샷을 외부 MySQL 데이터베이스로 로드할 수 없습니다. 대신 `mysqldump` 명령의 출력을 사용해야 합니다.
	
	1. `mysqldump` 명령의 출력을 복제 소스에서 MySQL 데이터베이스에도 연결 가능한 위치로 복사합니다.
	2. `mysql` 명령을 사용하여 MySQL 데이터베이스에 연결합니다. 다음은 예제입니다.
	`PROMPT> mysql -h host_name  -port=3306 -u  db_master_user  -p`
	3. `mysql` 프롬프트에서 `source` 명령을 실행하고 데이터베이스 덤프 파일의 이름을 전달하여 MySQL 데이터베이스로 데이터를 로드합니다. 다음은 예제입니다.
	`mysql> source backup.sql;`

5. 복제 소스에 복제 사용자 생성

복제에 사용되는 사용자 ID를 생성할 수도 있습니다. 다음은 RDS for MySQL 또는 외부 MySQL 소스 데이터베이스의 예제입니다.

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

Aurora MySQL 소스 데이터베이스의 경우, `skip_name_resolve` DB 클러스터 파라미터는 1(ON)로 설정되며 수정할 수 없으므로 도메인 이름 대신 호스트의 IP 주소를 사용해야 합니다. 자세한 내용은 MySQL 설명서의 skip_name_resolve를 참조하세요.

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한이 필요합니다. 해당 사용자에게 이 권한을 부여합니다.

암호화된 복제를 사용해야 한다면, 복제 사용자에게 SSL 연결이 반드시 필요합니다. 예를 들어, 다음 문 중 하나를 사용하여 사용자 계정 `repl_user`에 대한 SSL 연결을 요구할 수 있습니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> `REQUIRE SSL`이 포함되어 있지 않다면, 복제 연결이 자동으로 암호화되지 않은 연결로 돌아갈 수 있습니다.

6. 복제본 대상에서 복제 설정

복제를 설정하기 전에 Aurora MySQL DB 클러스터 또는 RDS for MySQL DB 인스턴스 복제본 대상의 스냅샷을 수동으로 생성하는 것이 좋습니다. 문제가 발생하여 DB 클러스터 또는 DB 인스턴스 복제본 대상을 통해 복제를 다시 설정해야 하는 경우, 데이터를 복제본 대상으로 다시 가져오는 대신 이 스냅샷에서 DB 클러스터 또는 DB 인스턴스를 복원할 수 있습니다.

데이터베이스 엔진에 대한 복제를 켜려면 다음 지침을 확인하세요.

데이터베이스 엔진	지침
Aurora MySQL	Aurora MySQL DB 클러스터에서 복제를 설정하려면 1. 복제를 수행할 시작 위치를 찾습니다. binlog 파일 이름과 binlog 위치가 필요합니다. DB 클러스터 복제본 대상이 다음에서 생성된 경우 • DB 클러스터 스냅샷 또는 클론 - 3. 복제 소스의 사본 또는 덤프 만들기에 나와 있듯이, 새로운 DB 클러스터의 최신 이벤트에서 binlog 파일 이름과 위치를 검색합니다. • DB 스냅샷 - 복제 소스의 스냅샷을 생성했을 때 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 명령에서 binlog 파일 이름과 위치를 검색했습니다. 2. DB 클러스터에 연결하고 다음 절차를 호출하여 이전 단계의 이진 로그 파일 이름 및 위치를 사용하여 복제 소스로 복제를 시작합니다.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td></td><td>

- mysql.rds_set_external_source(Aurora MySQL 버전 3)
- mysql.rds_set_external_master(Aurora MySQL 버전 2)
- mysql.rds_start_replication(모든 버전)

다음은 Aurora MySQL 버전 3에 대한 예입니다.

```
CALL mysql.rds_set_external_source ('mydbinstance.123456789012
.us-east-1.rds.amazonaws.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107,
 0);
CALL mysql.rds_start_replication;
```

SSL 암호화를 사용하려면 최종 값을 `0` 대신 `1`로 설정합니다.

</td></tr>
<tr><td>RDS for MySQL</td><td>

Amazon RDS DB 인스턴스에서 복제를 설정하려면

1. DB 스냅샷에서 DB 인스턴스 복제본 대상을 생성한 경우 복제 시작 위치에 해당하는 binlog 파일 및 binlog 위치가 필요합니다. 복제 소스의 스냅샷을 생성했을 때 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 명령에서 이러한 값을 검색했습니다.
2. 복제 소스를 사용하여 복제를 시작하려면 DB 인스턴스에 연결하고 mysql.rds_set_external_master(Aurora MySQL 버전 2) 또는 mysql.rds_set_external_source(Aurora MySQL 버전 3) 및 mysql.rds_start_replication 프로시저를 호출합니다. 이전 단계의 바이너리 로그 파일 이름 및 위치를 사용합니다. 다음은 예입니다.

```
CALL mysql.rds_set_external_master ('mydbcluster.cluster-12345
6789012.us-east-1.rds.amazonaws.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107,
 0);
CALL mysql.rds_start_replication;
```

SSL 암호화를 사용하려면 최종 값을 `0` 대신 `1`로 설정합니다.

</td></tr>
</table>

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>MySQL(외부)</td><td>

외부 MySQL 데이터베이스에서 복제를 설정하려면

1. 복제 시작 위치에 해당하는 binlog 파일 및 binlog 위치를 검색합니다. 복제 소스의 스냅샷을 생성했을 때 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 명령에서 이러한 값을 검색했습니다. `mysqldump` 명령을 `--master-data=2` 옵션과 함께 실행하여 그 출력으로 외부 MySQL 복제본 대상을 채운 경우 binlog 파일 및 binlog 위치는 출력에 포함되어 있습니다. 다음은 예입니다.

```
--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031',
 MASTER_LOG_POS=107;
```

2. 외부 MySQL 복제본 대상에 연결하고 `CHANGE MASTER TO` 및 `START SLAVE`(Aurora MySQL 버전 2) 또는 `START REPLICA`(Aurora MySQL 버전 3)를 실행하여 이전 단계의 바이너리 로그 파일 이름과 위치를 사용하여 복제 소스로 복제를 시작합니다.

```
CHANGE MASTER TO
  MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.r
ds.amazonaws.com',
  MASTER_PORT = 3306,
  MASTER_USER = 'repl_user',
  MASTER_PASSWORD = 'password',
  MASTER_LOG_FILE = 'mysql-bin-changelog.000031',
  MASTER_LOG_POS = 107;
-- And one of these statements depending on your engine version:
START SLAVE; -- Aurora MySQL version 2
START REPLICA; -- Aurora MySQL version 3
```

</td></tr>
</table>

복제가 실패하면 복제본에서 의도하지 않은 I/O가 크게 증가하여 성능이 저하될 수 있습니다. 복제가 실패하거나 더 이상 필요하지 않은 경우 mysql.rds_reset_external_master(Aurora MySQL 버전 2) 또는

mysql.rds_reset_external_source(Aurora MySQL 버전 3) 저장 프로시저를 실행하여 복제 구성을 제거할 수 있습니다.

읽기 전용 복제본에 대한 복제를 중지할 위치 설정

Aurora MySQL 버전 3.04 이상에서는 mysql.rds_start_replication_until(Aurora MySQL 버전 3) 저장 프로시저를 사용하여 복제를 시작한 다음 지정된 이진 로그 파일 위치에서 복제를 중지할 수 있습니다.

읽기 전용 복제본에 대한 복제를 시작하고 특정 위치에서 복제를 중지하려면

1. MySQL 클라이언트를 사용하여 Aurora MySQL DB 클러스터 복제본에 마스터 사용자로 연결합니다.
2. mysql.rds_start_replication_until(Aurora MySQL 버전 3) 저장 프로시저를 실행합니다.

   다음 예제에서는 복제를 시작하고 `120` 바이너리 로그 파일의 `mysql-bin-changelog.000777` 위치에 도달할 때까지 변경 사항을 복제합니다. 재해 복구 시나리오에서 `120`이 재해 직전 위치라고 가정합니다.

   ```
   call mysql.rds_start_replication_until(
     'mysql-bin-changelog.000777',
     120);
   ```

중지 지점에 도달하면 복제가 자동으로 중지됩니다. `Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure` RDS 이벤트가 생성됩니다.

GTID 기반 복제를 사용하는 경우 mysql.rds_start_replication_until_gtid(Aurora MySQL 버전 3) 저장 프로시저 대신 mysql.rds_start_replication_until(Aurora MySQL 버전 3) 저장 프로시저를 사용하십시오. GTID 기반 복제에 대한 자세한 내용은 GTID 기반 복제 사용 단원을 참조하십시오.

7. 복제본 모니터링

Aurora MySQL DB 클러스터를 사용하여 MySQL 복제를 설정한 경우 복제본 대상인 Aurora MySQL DB 클러스터에 대한 장애 조치 이벤트를 모니터링해야 합니다. 장애 조치가 발생할 경우에는 복제본 대상인 DB 클러스터가 다른 네트워크 주소를 가진 새 호스트에서 다시 생성될 수도 있습니다. 장애 조치 이벤트를 모니터링하는 자세한 방법은 Amazon RDS 이벤트 알림 작업 단원을 참조하세요.

또한 복제본 대상에 연결하고 `SHOW SLAVE STATUS`(Aurora MySQL 버전 2) 또는 `SHOW REPLICA STATUS`(Aurora MySQL 버전 3) 명령을 실행하여 복제본 대상이 복제 소스보다 얼마나 지연되는지 모

니터링할 수 있습니다. 명령 출력의 `Seconds Behind Master` 필드는 복제본 대상이 소스보다 얼마나 지연되었는지 알려줍니다.

> **Important**
>
> DB 클러스터를 업그레이드하고 사용자 지정 파라미터 그룹을 지정하는 경우 업그레이드가 완료된 후 클러스터를 수동으로 재부팅해야 합니다. 이렇게 하면 클러스터가 새 사용자 지정 파라미터 설정을 사용하고 binlog 복제를 다시 시작합니다.

복제 소스와 타겟 간 비밀번호 동기화

SQL 문을 사용하여 복제 소스에서 사용자 계정 및 암호를 변경하면 해당 변경 사항이 복제 대상에 자동으로 복제됩니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 복제 소스의 마스터 암호를 변경하는 경우 이러한 변경 사항은 복제 대상에 자동으로 복제되지 않습니다. 소스 시스템과 타겟 시스템 간에 마스터 사용자 및 마스터 비밀번호를 동기화하려는 경우 복제 타겟을 직접 동일하게 변경해야 합니다.

Aurora MySQL의 이진수 로그 복제 중지

MySQL DB 인스턴스, 외부 MySQL 데이터베이스 또는 다른 Aurora DB 클러스터와의 이진 로그 복제 중지 단계는 다음과 같으며, 이 주제의 뒷부분에 자세히 설명되어 있습니다.

1. 복제본 대상의 이진 로그 복제 중단

2. 복제 소스에 대한 이진 로깅 해제

1. 복제본 대상의 이진 로그 복제 중단

데이터베이스 엔진의 바이너리 로그 복제를 중지하려면 다음 지침을 따르세요.

데이터베이스 엔진	지침
Aurora MySQL	Aurora MySQL DB 클러스터 복제본 대상에 대한 이진 로그 복제를 중지하려면 복제본 대상인 Aurora DB 클러스터에 연결하고 mysql.rds_stop_replication 프로시저를 호출합니다.

데이터베이스 엔진	지침
RDS for MySQL	Amazon RDS DB 인스턴스에 대한 이진 로그 복제를 중지하려면 복제본 대상인 RDS DB 클러스터에 연결하고 mysql.rds_stop_replication 프로시저를 호출합니다.
MySQL(외부)	외부 MySQL 데이터베이스에서 이진 로그 복제를 중지하려면 MySQL 데이터베이스에 연결하고 `STOP SLAVE`(버전 5.7) 또는 `STOP REPLICA`(버전 8.0) 명령을 실행합니다.

2. 복제 소스에 대한 이진 로깅 해제

데이터베이스 엔진의 복제 소스에서 바이너리 로깅을 비활성화하려면 다음 테이블의 지침을 따르세요.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>Aurora MySQL</td><td>
Amazon Aurora DB 클러스터에 대한 이진 로깅을 해제하려면

복제 소스인 Aurora DB 클러스터에 연결합니다.
다음 예제에 표시된 것처럼 mysql.rds_set_configuration 프로시저를 사용하여 구성 파라미터 <code>binlog retention hours</code>를 NULL 값과 함께 지정합니다.
<pre>CALL mysql.rds_set_configuration('binlog retention hours', NULL);</pre>
Note
<code>binlog retention hours</code>에는 <code>0</code> 값을 사용할 수 없습니다.

복제 소스에서 <code>binlog_format</code> 파라미터를 <code>OFF</code>로 설정합니다. <code>binlog_format</code> 파라미터는 DB 클러스터와 연결된 사용자 지정 DB 클러스터 파라미터 그룹에 있습니다.

</td></tr>
</table>

데이터베이스 엔진	지침
	`binlog_format` 파라미터 값을 변경한 후에는 DB 클러스터를 재부팅해야만 변경 사항이 적용됩니다. 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 및 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하세요.
RDS for MySQL	Amazon RDS DB 인스턴스에 대한 이진 로깅을 해제하려면 Amazon RDS DB 인스턴스에 대한 이진 로깅을 직접 해제할 수 없지만, 다음 중 하나를 수행하여 해제할 수 있습니다. 1. DB 인스턴스의 자동 백업을 해제합니다. 기존 DB 인스턴스를 수정하고 백업 보관 기간을 0으로 설정하여 자동 백업을 해제할 수 있습니다. 자세한 내용은 Amazon Relational Database Service 사용 설명서의 Amazon RDS DB 인스턴스 수정 및 백업 작업을 참조하세요. 2. DB 인스턴스의 읽기 전용 복제본을 모두 삭제합니다. 자세한 정보는 Amazon Relational Database Service 사용 설명서에서 MariaDB, MySQL, PostgreSQL DB 인스턴스의 읽기 전용 복제본 사용을 참조하십시오.

<table>
<tr><th>데이터베이스 엔진</th><th>지침</th></tr>
<tr><td>MySQL(외부)</td><td>
외부 MySQL 데이터베이스에 대한 이진 로깅을 해제하려면
MySQL 데이터베이스에 연결하고 <code>STOP REPLICATION</code> 명령을 호출합니다.

명령 셸에서 <code>mysqld</code> 서비스를 중지합니다.
<pre><code>sudo service mysqld stop</code></pre>
<code>my.cnf</code> 파일을 편집합니다(이 파일은 보통 <code>/etc</code>에 있음).
<pre><code>sudo vi /etc/my.cnf</code></pre>
<code>log_bin</code> 섹션에서 <code>server_id</code> 및 <code>[mysqld]</code>옵션을 삭제합니다.
자세한 내용은 MySQL 설명서에서 Setting the replication source configuration 섹션을 참조하세요.
mysql 서비스를 시작합니다.
<pre><code>sudo service mysqld start</code></pre>

</td></tr>
</table>

Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 규모 조정

MySQL DB 인스턴스에서 Amazon Aurora를 사용하여 Amazon Aurora의 읽기 조정 기능을 활용하고 MySQL DB 인스턴스에 대한 읽기 작업을 확장할 수 있습니다. Aurora을 사용하여 MySQL DB 인스턴스에 대한 읽기 조정을 수행하려면 Amazon Aurora MySQL DB 클러스터를 생성한 후 이 클러스터를 MySQL DB 인스턴스의 읽기 전용 복제본으로 설정합니다. 이러한 설정은 RDS for MySQL DB 인스턴스 또는 Amazon RDS 외부에서 실행 중인 MySQL 데이터베이스에 적용됩니다.

Amazon Aurora DB 클러스터 생성에 대한 자세한 정보는 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

MySQL DB 인스턴스와 Amazon Aurora DB 클러스터 간의 복제를 설정할 때는 다음 지침을 따라야 합니다.

- Amazon Aurora DB 클러스터를 참조할 때 Amazon Aurora MySQL DB 클러스터 엔드포인트 주소를 사용합니다. 장애 조치가 발생하는 경우 Aurora DB 클러스터용 기본 인스턴스로 승격되는 Aurora MySQL 복제본에서 DB 클러스터 엔드포인트 주소가 계속 사용됩니다.
- Binlog가 Aurora 복제본에 적용되었음을 확인할 때까지는 라이터 인스턴스에서 binlog를 유지 관리합니다. 이렇게 유지 관리해야 오류 발생 시 라이터 인스턴스를 복원할 수 있습니다.

> **Important**
>
> 자체 관리형 복제를 사용하는 경우, 발생할 수 있는 복제 문제를 직접 모니터링하고 해결해야 합니다. 자세한 내용은 읽기 전용 복제본 간 지연 문제 진단 및 해결 단원을 참조하십시오.

> **Note**
>
> Aurora MySQL DB 클러스터에서 복제를 시작하는 데 필요한 권한이 제한되고 Amazon RDS 마스터 사용자를 사용할 수 없습니다. 그러므로 mysql.rds_set_external_master(Aurora MySQL 버전 2) 또는 mysql.rds_set_external_source(Aurora MySQL 버전 3) 및 mysql.rds_start_replication 프로시저를 사용하여 Aurora MySQL DB 클러스터와 MySQL DB 인스턴스 간 복제를 설정해야 합니다.

외부 소스 인스턴스와 Aurora MySQL DB 인스턴스 간의 복제 시작

1. 원본 MySQL DB 인스턴스를 읽기 전용으로 설정합니다.

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. 원본 MySQL DB 인스턴스에서 `SHOW MASTER STATUS` 명령을 실행하여 binlog 위치를 확인합니다. 다음 예제와 비슷한 출력 결과를 얻습니다.

```
File                        Position
------------------------------------
 mysql-bin-changelog.000031      107
------------------------------------
```

3. `mysqldump`를 사용하여 외부 MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터베이스를 복사합니다. 초대형 데이터베이스의 경우 Amazon Relational Database Service

사용 설명서의 [가동 중지 시간을 단축하여 MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기] 절차를 사용하려고 할 수도 있습니다.

대상 LinuxmacOS, 또는Unix:

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

Windows의 경우:

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

Note

-p 옵션과 입력한 암호 사이에 공백이 없어야 합니다.

`--host` 명령의 `--user (-u)`, `--port`, `-p` 및 `mysql` 옵션을 사용하여 Aurora DB 클러스터에 연결할 호스트 이름, 사용자 이름, 포트 및 비밀번호를 지정하십시오. 호스트 이름은 Amazon Aurora DB 클러스터 엔드포인트의 DNS 이름으로, 예를 들면 `mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com`입니다.

Amazon RDS Management Console의 클러스터 세부 정보에서 엔드포인트 값을 찾을 수 있습니다.

4. 다시 원본 MySQL DB 인스턴스를 쓰기 가능한 상태로 설정합니다.

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

복제에 사용할 백업을 만드는 방법에 대한 자세한 내용은 MySQL 설명서에서 [Backing up a source or replica by making it read only](#) 섹션을 참조하세요.

5. Amazon RDS Management Console에서 원본 MySQL 데이터베이스를 호스팅하는 서버의 IP 주소를 Amazon Aurora DB 클러스터용 VPC 보안 그룹에 추가합니다. VPC 보안 그룹 수정에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [VPC용 보안 그룹](#)을 참조하십시오.

Amazon Aurora DB 클러스터의 IP 주소에서의 연결을 허용하도록 로컬 네트워크를 구성하여 원본 MySQL 인스턴스와 통신할 수 있도록 해야 할 수도 있습니다. Amazon Aurora DB 클러스터의 IP 주소를 검색하려면 `host` 명령을 사용하십시오.

```
host aurora_endpoint_address
```

호스트 이름은 Amazon Aurora DB 클러스터 엔드포인트의 DNS 이름입니다.

6. 선택한 클라이언트를 사용하여 외부 MySQL 인스턴스에 연결하고 복제에 사용될 MySQL 사용자를 만듭니다. 이 계정은 오직 복제용으로만 사용되며 보안 향상을 위해 사용자의 도메인으로 제한되어야 합니다. 다음은 예제입니다.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

7. 외부 MySQL 인스턴스의 경우 복제 사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다. 예를 들어 도메인의 '`REPLICATION CLIENT`' 사용자를 위해 모든 데이터베이스에서 `REPLICATION SLAVE` 및 `repl_user` 권한을 부여하려면 다음 명령을 실행합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com'
    IDENTIFIED BY 'password';
```

8. 복제를 설정하기 전에 Aurora MySQL DB 클러스터의 수동 스냅샷을 읽기 전용 복제본으로 생성합니다. DB 클러스터와 복제를 읽기 복제본으로 다시 설정해야 할 경우, MySQL DB 인스턴스를 새로운 Aurora MySQL DB 클러스터로 가져오는 대신 이 스냅샷에서 Aurora MySQL DB 클러스터를 복원할 수 있습니다.

9. Amazon Aurora DB 클러스터를 복제본으로 만듭니다. Amazon Aurora DB 클러스터에 마스터 사용자로 연결하고 mysql.rds_set_external_master(Aurora MySQL 버전 2) 또는 mysql.rds_set_external_source(Aurora MySQL 버전 3) 및 mysql.rds_start_replication 프로시저를 사용하여 소스 MySQL 데이터베이스를 복제 소스로 식별합니다.

   2단계에서 결정한 binlog 파일 이름과 위치를 사용합니다. 다음은 예입니다.

```
For Aurora MySQL version 2:
CALL mysql.rds_set_external_master ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

For Aurora MySQL version 3:
CALL mysql.rds_set_external_source ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
```

10. Amazon Aurora DB 클러스터에서 mysql.rds_start_replication 프로시저를 호출하여 복제를 시작합니다.

```
CALL mysql.rds_start_replication;
```

원본 MySQL DB 인스턴스와 Amazon Aurora DB 클러스터 간의 복제를 설정한 후에는 Aurora 복제본을 Amazon Aurora DB 클러스터에 추가할 수 있습니다. 그러고 나면 Aurora 복제본에 연결하여 데이터에 대한 읽기 조정을 수행할 수 있습니다. Aurora 복제본 생성에 대한 자세한 정보는 DB 클러스터에 Aurora 복제본 추가 단원을 참조하십시오.

Aurora MySQL의 이진수 로그 복제 최적화

다음으로 Aurora MySQL에서 이진 로그 복제 성능을 최적화하고 관련 문제를 해결하는 방법을 배울 수 있습니다.

> **Tip**
>
> 이 논의에서는 MySQL 이진 로그 복제 메커니즘과 작동 방식에 대해 잘 알고 있다고 가정합니다. 배경 정보는 MySQL 설명서의 복제 구현 단원을 참조하세요.

다중 스레드 이진 로그 복제

다중 스레드 이진 로그 복제를 사용하면 SQL 스레드가 릴레이 로그에서 이벤트를 읽고 SQL 작업자 스레드에서 적용하도록 이벤트를 대기열에 넣습니다. SQL 작업자 스레드는 코디네이터 스레드에서 관리합니다. 가능한 경우 이진 로그 이벤트가 병렬로 적용됩니다.

다중 스레드 이진 로그 복제는 Aurora MySQL 버전 3, Aurora MySQL 버전 2.12.1 이상에서 지원됩니다.

Aurora MySQL DB 인스턴스가 binlog 복제를 사용하도록 구성된 경우 복제본 인스턴스는 Aurora MySQL 3.04 미만 버전에 기본적으로 단일 스레드 복제를 사용합니다. 다중 스레드 복제를 사용 설정하려면 `replica_parallel_workers` 파라미터를 사용자 정의 파라미터 그룹에서 0보다 큰 값으로 업데이트합니다.

Aurora MySQL 버전 3.04 이상에서는 복제가 기본적으로 다중 스레드로 설정되며 `replica_parallel_workers`가 4로 설정됩니다. 사용자 지정 파라미터 그룹에서 이 파라미터를 수정할 수 있습니다.

다음 구성 옵션을 사용하면 다중 스레드 복제를 세부 조정할 수 있습니다. 사용 정보는 MySQL 참조 설명서의 [복제, 이진 로깅 옵션 및 변수](#)를 참조하세요.

최적의 구성은 여러 요인에 따라 달라집니다. 예를 들어 이진 로그 복제의 성능은 데이터베이스 워크로드 특성과 복제본이 실행 중인 DB 인스턴스 클래스의 영향을 받습니다. 따라서 프로덕션 인스턴스에 새 파라미터 설정을 적용하기 전에 이러한 구성 파라미터에 대한 모든 변경 사항을 철저히 테스트하는 것이 좋습니다.

- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`
- `binlog_transaction_dependency_history_size`
- `binlog_transaction_dependency_tracking`
- `replica_preserve_commit_order`
- `replica_parallel_type`
- `replica_parallel_workers`

Aurora MySQL 버전 3.06 이상에서는 보조 인덱스가 두 개 이상인 대규모 테이블의 트랜잭션을 복제할 때 binlog 복제본의 성능을 개선할 수 있습니다. 이 기능은 binlog 복제본에서 보조 인덱스 변경 사항을 병렬로 적용하는 스레드 풀을 도입합니다. 이 기능은 보조 인덱스 변경 사항을 적용하는 데 사용할 수 있는 총 병렬 스레드 수를 제어하는

`aurora_binlog_replication_sec_index_parallel_workers` DB 클러스터 파라미터에 의해 제어됩니다. 기본적으로 파라미터는 `0`(비활성화됨)로 설정됩니다. 이 기능을 활성화해도 인스턴스를 다시 시작할 필요가 없습니다. 이 기능을 활성화하려면 진행 중인 복제를 중지하고 원하는 수의 병렬 작업자 스레드를 설정한 다음 복제를 다시 시작하세요.

이 파라미터를 글로벌 변수로 사용할 수도 있습니다. 여기서 *n*은 병렬 작업자 스레드 수입니다.

```
SET global aurora_binlog_replication_sec_index_parallel_workers=n;
```

binlog 복제 최적화(Aurora MySQL 2.10 이상)

Aurora MySQL 2.10 이상에서 Aurora는 binlog I/O 캐시라는 최적화를 바이너리 로그 복제에 자동으로 적용합니다. 이 최적화는 가장 최근에 커밋된 binlog 이벤트를 캐싱하여 binlog 덤프 스레드 성능을 개선하고 binlog 소스 인스턴스에 대한 포그라운드 트랜잭션에 미치는 영향을 제한하도록 설계되었습니다.

> **Note**
> 이 기능에 사용되는 메모리는 MySQL `binlog_cache` 설정과 독립적입니다.
> 이 기능은 `db.t2` 및 `db.t3` 인스턴스 클래스를 사용하는 Aurora DB 인스턴스에는 적용되지 않습니다.

이 최적화를 켜기 위해 구성 파라미터를 조정할 필요가 없습니다. 특히 이전 Aurora MySQL 버전에서 구성 파라미터 `aurora_binlog_replication_max_yield_seconds`를 0이 아닌 값으로 조정하는 경우 Aurora MySQL 2.10 이상에서는 0으로 다시 설정해야 합니다.

상태 변수 `aurora_binlog_io_cache_reads` 및 `aurora_binlog_io_cache_read_requests`는 Aurora MySQL 2.10 이상에서 사용할 수 있습니다. 이러한 상태 변수는 binlog I/O 캐시의 데이터를 읽는 빈도를 모니터링하는 데 도움이 됩니다.

- `aurora_binlog_io_cache_read_requests`는 캐시의 binlog I/O 읽기 요청 수를 보여줍니다.
- `aurora_binlog_io_cache_reads`는 캐시의 정보를 검색하는 binlog I/O 읽기 수를 보여줍니다.

다음 SQL 쿼리는 캐시된 정보를 활용하는 binlog 읽기 요청의 백분율을 계산합니다. 이 경우 비율이 100에 가까울수록 더 좋습니다.

```
mysql> SELECT
```

```
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

binlog I/O 캐시 기능에는 binlog 덤프 스레드와 관련된 새로운 지표도 포함됩니다. 덤프 스레드는 새로운 binlog 복제본이 binlog 소스 인스턴스에 연결될 때 생성되는 스레드입니다.

덤프 스레드 지표는 60초 간격으로 접두사 [Dump thread metrics]를 사용하여 데이터베이스 로그에 인쇄됩니다. 지표에는 Secondary_id, Secondary_uuid, binlog 파일 이름 및 각 복제본에서 읽는 중인 위치와 같은 각 binlog 복제본에 대한 정보가 포함됩니다. 지표에는 복제 소스와 복제본 간의 거리(바이트)를 나타내는 Bytes_behind_primary도 포함됩니다. 이 지표는 복제본 I/O 스레드의 지연을 측정합니다. 이 수치는 binlog 복제본에서 seconds_behind_master 지표로 표시되는 복제본 SQL 적용자 스레드의 지연과 다릅니다. 이 거리의 감소 또는 증가를 확인하여 binlog 복제본이 소스를 따라 잡고 있는지, 뒤쳐지지 않는지 확인할 수 있습니다.

binlog 복제 최적화(Aurora MySQL 2~2.09)

Aurora MySQL에 대한 이진 로그 복제를 최적화하려면 다음 클러스터 수준 최적화 매개 변수를 조정합니다. 이러한 파라미터는 binlog 원본 인스턴스의 지연 시간과 복제 지연 사이의 적절한 균형을 지정하는 데 도움이 됩니다.

- aurora_binlog_use_large_read_buffer
- aurora_binlog_read_buffer_size
- aurora_binlog_replication_max_yield_seconds

> **Note**
>
> MySQL 5.7 호환 클러스터의 경우 Aurora MySQL 버전 2부터 2.09.*까지 이러한 파라미터를 사용할 수 있습니다. Aurora MySQL 2.10.0 이상에서는 이러한 파라미터가 binlog I/O 캐시 최적화로 대체되므로 사용할 필요가 없습니다.

주제

- 대용량 읽기 버퍼 및 최대 수율 최적화 개요
- 관련 파라미터
- 이진 로그 복제 최대 수율 메커니즘 사용 설정
- 이진 로그 복제 최대 수율 최적화 해제
- 대량 읽기 버퍼 해제

대용량 읽기 버퍼 및 최대 수율 최적화 개요

클러스터가 많은 수의 트랜잭션을 처리하는 동안 이진 로그 덤프 스레드가 Aurora 클러스터 볼륨에 액세스할 때 이진 로그 복제 성능이 저하될 수 있습니다. 파라미터 `aurora_binlog_use_large_read_buffer`, `aurora_binlog_replication_max_yield_seconds`, 및 `aurora_binlog_read_buffer_size`를 사용하여 이러한 유형의 문제를 최소화할 수 있습니다.

`aurora_binlog_replication_max_yield_seconds`가 0보다 큰 값으로 설정되고 덤프 스레드의 현재 binlog 파일이 활성 상태인 상황을 가정해 보겠습니다. 이 경우 기존 binlog 파일을 트랜잭션으로 채울 때까지 이진 로그 덤프 스레드가 지정한 시간(초)까지 대기합니다. 이 대기 기간은 각 binlog 이벤트를 개별적으로 복제하여 발생할 수 있는 문제를 방지합니다. 그러나 이러면 이진 로그 복제본에 대한 복제 지연이 증가합니다. 이러한 복제본은 `aurora_binlog_replication_max_yield_seconds` 설정과 동일한 시간(초)만큼 소스의 뒤떨어질 수 있습니다.

현재 binlog 파일은 덤프 스레드가 복제를 수행하기 위해 현재 읽고 있는 binlog 파일을 의미합니다. binlog 파일이 업데이트되거나 들어오는 트랜잭션에 의해 업데이트되도록 열려 있을 때 binlog 파일이 활성 상태인 것으로 간주됩니다. Aurora MySQL이 활성 binlog 파일을 채운 후 MySQL은 새 binlog 파일을 생성하고 전환합니다. 이전 binlog 파일이 비활성 상태가 됩니다. 더 이상 들어오는 트랜잭션에 의해 업데이트되지 않습니다.

> **Note**
>
> 이러한 파라미터를 조정하기 전에 시간에 따른 트랜잭션 지연 시간 및 처리량을 측정하십시오. 이진 로그 복제 성능이 안정적이며 가끔 경합이 발생하더라도 대기 시간이 짧다는 것을 알 수 있을 것입니다.

`aurora_binlog_use_large_read_buffer`

이 매개 변수를 1로 설정하면 Aurora MySQL는 파라미터 `aurora_binlog_read_buffer_size` 및 `aurora_binlog_replication_max_yield_seconds`의 설정에 따라 이진 로그 복제를 최적화합니다. `aurora_binlog_use_large_read_buffer`가 0이면 Aurora MySQL은 `aurora_binlog_read_buffer_size` 및 `aurora_binlog_replication_max_yield_seconds` 파라미터의 값을 무시합니다.

`aurora_binlog_read_buffer_size`

더 큰 읽기 버퍼가 있는 이진 로그 덤프 스레드는 각 I/O에 대해 더 많은 이벤트를 읽음으로써 읽기 I/O 작업 수를 최소화합니다. 파라미터 `aurora_binlog_read_buffer_size`는 읽기 버퍼 크기를 설정합니다. 대용량 읽기 버퍼는 대량의 binlog 데이터를 생성하는 워크로드에 대한 이진 로그 경합을 줄여줄 수 있습니다.

> **Note**
>
> 이 파라미터는 클러스터에 설정 `aurora_binlog_use_large_read_buffer=1`이 있는 경우에만 영향을 줍니다.
> 읽기 버퍼의 크기를 늘려도 이진 로그 복제 성능에 영향을 주지 않습니다. 이진 로그 덤프 스레드는 읽기 버퍼를 채우기 위해 트랜잭션을 업데이트할 때까지 기다리지 않습니다.

`aurora_binlog_replication_max_yield_seconds`

워크로드에 낮은 트랜잭션 대기 시간이 필요하고 일부 복제 지연을 허용할 수 있는 경우 `aurora_binlog_replication_max_yield_seconds` 파라미터를 늘릴 수 있습니다. 이 매개 변수는 클러스터에서 이진 로그 복제의 최대 수율 속성을 제어합니다.

> **Note**
>
> 이 파라미터는 클러스터에 설정 `aurora_binlog_use_large_read_buffer=1`이 있는 경우에만 영향을 줍니다.

Aurora MySQL 는 `aurora_binlog_replication_max_yield_seconds` 파라미터 값에 대한 모든 변경 사항을 즉시 인식합니다. DB 인스턴스를 다시 시작할 필요가 없습니다. 그러나 이 설정을 설정하면 현재 binlog 파일이 최대 크기인 128MB에 도달하고 새 파일로 회전될 때에 한하여 덤프 스레드가 생성되기 시작합니다.

관련 파라미터

binlog 최적화를 활성화하려면 다음 DB 클러스터 파라미터를 사용합니다.

파라미터	기본값	유효한 값	설명
aurora_binlog_use_large_read_buffer	1	0, 1	복제 개선 기능을 켜기 위한 스위치. 값이 1이면 바이너리 로그 덤프 스레드는 바이너리 로그 복제에 aurora_binlog_read_buffer_size 를 사용합니다. 그렇지 않으면 기본 버퍼 크기(8K)가 사용됩니다. Aurora MySQL 버전 3에서 사용되지 않습니다.
aurora_binlog_read_buffer_size	5242880	8192-536870912	파라미터 aurora_binlog_use_large_read_buffer 가 1로 설정된 경우 이진 로그 덤프 스레드가 사용하는 버퍼 크기를 읽습니다. Aurora MySQL 버전 3에서 사용되지 않습니다.
aurora_binlog_replication_max_yield_seconds	0	0-36000	Aurora MySQL 버전 2.07*에서 허용되는 최댓값은 45입니다. 2.09 및 그 이후 버전에서 더 높은 값으로 조정할 수 있습니다.

파라미터	기본값	유효한 값	설명
			버전 2의 경우 이 파라미터는 파라미터 aurora_binlog_use_large_read_buffer 가 1로 설정된 경우에만 작동합니다.

이진 로그 복제 최대 수율 메커니즘 사용 설정

다음과 같이 이진 로그 복제 최대 수율 최적화를 시작할 수 있습니다. 이렇게 하면 binlog 원본 인스턴스에서 트랜잭션에 대한 지연 시간이 최소화됩니다. 그러나 더 높은 복제 지연이 발생할 수 있습니다.

Aurora MySQL 클러스터에 대해 최대 수율 binlog 최적화를 설정하려면

1. 다음 파라미터 설정을 사용해 DB 클러스터 파라미터 그룹을 생성 또는 편집합니다.
   - aurora_binlog_use_large_read_buffer: ON 또는 1의 값으로 켭니다.
   - aurora_binlog_replication_max_yield_seconds: 0보다 큰 값을 입력합니다.
2. DB 클러스터 파라미터 그룹을 binlog 소스로 작동하는 Aurora MySQL 클러스터와 연결합니다. 이 작업을 수행하려면 Amazon Aurora의 파라미터 그룹의 절차를 따르십시오.
3. 파라미터 변경 사항이 적용되는지 확인합니다. 이렇게 하기 위해 binlog 원본 인스턴스에서 다음 쿼리를 실행하십시오.

```
SELECT @@aurora_binlog_use_large_read_buffer,
 @@aurora_binlog_replication_max_yield_seconds;
```

다음과 유사하게 출력되어야 합니다.

```
+---------------------------------------
+-----------------------------------------------+
| @@aurora_binlog_use_large_read_buffer |
 @@aurora_binlog_replication_max_yield_seconds |
+---------------------------------------
+-----------------------------------------------+
```

```
|                                      1 |
  45 |
+----------------------------------------
+------------------------------------------------+
```

이진 로그 복제 최대 수율 최적화 해제

다음과 같이 이진 로그 복제 최대 수율 최적화를 해제할 수 있습니다. 이렇게 하면 복제 지연이 최소화됩니다. 그러나 binlog 원본 인스턴스의 트랜잭션에 대한 지연 시간이 더 길어질 수 있습니다.

Aurora MySQL 클러스터에 대한 최대 수율 최적화를 해제하는 방법

1. Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 `aurora_binlog_replication_max_yield_seconds`가 0으로 설정되어 있는지 확인합니다. 파라미터 그룹을 사용한 구성 파라미터 설정에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.
2. 파라미터 변경 사항이 적용되는지 확인합니다. 이렇게 하기 위해 binlog 원본 인스턴스에서 다음 쿼리를 실행하십시오.

```
SELECT @@aurora_binlog_replication_max_yield_seconds;
```

다음과 유사하게 출력되어야 합니다.

```
+------------------------------------------------+
| @@aurora_binlog_replication_max_yield_seconds |
+------------------------------------------------+
|                                              0 |
+------------------------------------------------+
```

대량 읽기 버퍼 해제

다음과 같이 전체 대량 읽기 버퍼 기능을 해제할 수 있습니다.

Aurora MySQL 클러스터에 대한 큰 이진 로그 읽기 버퍼를 해제하려면

1. `aurora_binlog_use_large_read_buffer`를 OFF 또는 0으로 재설정합니다.

   Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 `aurora_binlog_use_large_read_buffer`가 0으로 설정되어 있는지 확인합니다. 파라미터

그룹을 사용한 구성 파라미터 설정에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹] 단원을 참조하십시오.

2. binlog 원본 인스턴스에서 다음 쿼리를 실행합니다.

```
SELECT @@ aurora_binlog_use_large_read_buffer;
```

다음과 유사하게 출력되어야 합니다.

```
+----------------------------------------+
| @@aurora_binlog_use_large_read_buffer |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
```

Aurora MySQL에 대해 향상된 binlog 설정

향상된 binlog는 binlog를 켜서 발생하는 컴퓨팅 성능 오버헤드를 줄여 주는데, 이 오버헤드는 경우에 따라 최대 50% 까지 증가할 수 있습니다. 향상된 binlog를 사용하면 이러한 오버헤드를 약 13%로 줄일 수 있습니다. 오버헤드를 줄이기 위해 향상된 binlog는 바이너리 로그와 트랜잭션 로그를 스토리지에 병렬식으로 기록하여 트랜잭션 커밋 시 기록되는 데이터를 최소화합니다.

또한 향상된 binlog를 사용하면 커뮤니티 MySQL binlog에 비해 다시 시작 및 장애 조치 후 데이터베이스 복구 시간이 최대 99% 향상됩니다. 향상된 binlog는 기존 binlog 기반 워크로드와 호환되며 커뮤니티 MySQL binlog와 상호 작용하는 것과 동일한 방식으로 상호 작용합니다.

향상된 binlog는 Aurora MySQL 버전 3.03.1 이상에서 사용할 수 있습니다.

주제

- 향상된 binlog 파라미터 구성
- 기타 관련 파라미터
- 향상된 binlog와 커뮤니티 MySQL binlog의 차이점
- 향상된 빈로그를 위한 Amazon CloudWatch 지표
- 향상된 빈로그의 제한 사항

향상된 binlog 파라미터 구성

향상된 binlog 파라미터를 켜거나 끄면 커뮤니티 MySQL binlog와 향상된 binlog 간에 전환할 수 있습니다. 기존 binlog 소비자는 binlog 파일 시퀀스의 공백 없이 binlog 파일을 계속 읽고 사용할 수 있습니다.

향상된 binlog를 켜려면 다음 파라미터를 설정합니다.

파라미터	Default	설명
`binlog_format`	–	`binlog_format` 파라미터를 원하는 바이너리 로깅 형식으로 설정하여 향상된 binlog를 켭니다. `binlog_format parameter` 가 OFF로 설정되어 있지 않은지 확인하세요. 자세한 내용은 [Aurora MySQL 이진 로깅 구성](#)을 참조하세요.
`aurora_enhanced_binlog`	`0`	Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 이 파라미터를 `1`로 설정합니다. 이 파라미터의 값을 변경할 때 `DBClusterParameterGroupStatus` 값이 `pending-reboot` 로 표시되면 라이터 인스턴스를 재부팅해야 합니다.
`binlog_backup`	`1`	향상된 binlog를 켜려면 이 파라미터를 끄세요. 이렇게 하려면 이 파라미터의 값을 `0`으로 설정합니다.
`binlog_replication_globaldb`	`1`	향상된 binlog를 켜려면 이 파라미터를 끄세요. 이렇게 하려면 이 파라미터의 값을 `0`으로 설정합니다.

> **⚠ Important**
>
> 향상된 binlog를 사용하는 경우에만 `binlog_backup` 및 `binlog_replication_globaldb` 파라미터를 끌 수 있습니다.

향상된 binlog를 끄려면 다음 파라미터를 설정합니다.

파라미터	설명
`aurora_enhanced_binlog`	Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 이 파라미터를 `0`로 설정합니다. 이 파라미터의 값을 변경할 때 `DBClusterParameterGroupStatus` 값이 `pending-reboot` 로 표시되면 라이터 인스턴스를 재부팅해야 합니다.
`binlog_backup`	향상된 binlog를 끄려면 이 파라미터를 켜세요. 이렇게 하려면 이 파라미터의 값을 `1`으로 설정합니다.
`binlog_replication_globaldb`	향상된 binlog를 끄려면 이 파라미터를 켜세요. 이렇게 하려면 이 파라미터의 값을 `1`으로 설정합니다.

향상된 binlog가 켜져 있는지 확인하려면 MySQL 클라이언트에서 다음 명령을 사용하세요.

```
mysql>show status like 'aurora_enhanced_binlog';

+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| aurora_enhanced_binlog  | ACTIVE |
+-------------------------+--------+
1 row in set (0.00 sec)
```

향상된 binlog가 켜져 있을 경우 `aurora_enhanced_binlog`에 대한 출력이 `ACTIVE`로 표시됩니다.

기타 관련 파라미터

향상된 binlog를 켜면 다음 파라미터가 영향을 받습니다.

- `max_binlog_size` 파라미터는 표시되지만 수정할 수는 없습니다. 향상된 binlog가 켜지면 기본값 `134217728`이 `268435456`으로 자동 조정됩니다.
- 커뮤니티 MySQL binlog와 달리 향상된 binlog가 켜져 있을 때는 `binlog_checksum`이 동적 파라미터로 작동하지 않습니다. 이 파라미터에 대한 변경 사항을 적용하려면 `ApplyMethod`가 `immediate`인 경우에도 DB 클러스터를 수동으로 재부팅해야 합니다.
- 향상된 binlog가 켜져 있을 때는 `binlog_order_commits` 파라미터에 설정한 값이 커밋 순서에 영향을 주지 않습니다. 커밋은 성능에 더 이상 영향을 주지 않고 항상 순서가 지정됩니다.

향상된 binlog와 커뮤니티 MySQL binlog의 차이점

향상된 binlog는 커뮤니티 MySQL binlog와 비교할 때 복제, 백업 및 Aurora Global Database와 다르게 상호 작용합니다. 향상된 binlog를 사용하기 전에 다음과 같은 차이점을 이해하는 것이 좋습니다.

- 소스 DB 클러스터의 향상된 binlog 파일은 복제된 DB 클러스터에서 사용할 수 없습니다.
- 향상된 binlog 파일은 Aurora 백업에 포함되지 않습니다. 따라서 DB 클러스터를 복원한 후에는 보존 기간이 설정되어 있더라도 소스 DB 클러스터의 향상된 binlog 파일을 사용할 수 없습니다.
- Aurora 글로벌 데이터베이스와 함께 사용하면 프라이머리 DB 클러스터의 향상된 binlog 파일이 보조 리전의 DB 클러스터에 복제되지 않습니다.

예시

다음 예는 향상된 binlog와 커뮤니티 MySQL binlog 간의 차이점을 설명합니다.

복원되거나 복제된 DB 클러스터에서

향상된 binlog가 켜져 있으면 복원되거나 복제된 DB 클러스터에서 이전 binlog 파일을 사용할 수 없습니다. 복원 또는 복제 작업 후 binlog가 켜져 있으면 새 DB 클러스터는 1(mysql-bin-changelog.000001)부터 고유한 binlog 파일 시퀀스를 쓰기 시작합니다.

복원 또는 복제 작업 후에 향상된 binlog를 켜려면 복원되거나 복제된 DB 클러스터에서 필요한 DB 클러스터 파라미터를 설정하세요. 자세한 내용은 [향상된 binlog 파라미터 구성](#) 단원을 참조하십시오.

Example 예제: 향상된 binlog가 켜져 있을 때 수행되는 복제 또는 복원 작업

소스 DB 클러스터:

```
mysql> show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

복원되거나 복제된 DB 클러스터에서는 향상된 binlog가 켜져 있을 때 binlog 파일이 백업되지 않습니다. binlog 데이터의 불연속성을 방지하기 위해 향상된 binlog를 켜기 전에 작성한 binlog 파일도 사용할 수 없습니다.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 예제: 향상된 binlog가 꺼져 있을 때 수행되는 복제 또는 복원 작업

소스 DB 클러스터:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
```

```
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

향상된 binlog는 `mysql-bin-changelog.000003` 이후에 비활성화됩니다. 복원되거나 복제된 DB 클러스터에서는 향상된 binlog를 끈 후에 작성한 binlog 파일을 사용할 수 있습니다.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Amazon Aurora Global Database에서

Amazon Aurora Global Database에서는 프라이머리 DB 클러스터의 binlog 데이터가 보조 DB 클러스터에 복제되지 않습니다. 크로스 리전 장애 조치 프로세스 후에는 새로 승격된 프라이머리 DB 클러스터에서 binlog 데이터를 사용할 수 없습니다. binlog가 켜져 있으면 새로 승격된 DB 클러스터는 1(mysql-bin-changelog.000001)부터 고유한 binlog 파일 시퀀스를 시작합니다.

장애 조치 후 향상된 binlog를 켜려면 보조 DB 클러스터에서 필수 DB 클러스터 파라미터를 설정해야 합니다. 자세한 내용은 [향상된 binlog 파라미터 구성](#) 단원을 참조하십시오.

Example 예제: 향상된 binlog가 켜져 있을 때 글로벌 데이터베이스 장애 조치 작업이 수행됩니다.

이전 프라이머리 DB 클러스터(장애 조치 전):

```
mysql>show binary logs;
```

```
+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

새 프라이머리 DB 클러스터(장애 조치 후):

향상된 binlog가 켜져 있을 때 binlog 파일은 보조 리전에 복제되지 않습니다. binlog 데이터의 불연속성을 방지하기 위해 향상된 binlog를 켜기 전에 작성한 binlog 파일은 사용할 수 없습니다.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 예제: 향상된 binlog가 꺼져 있을 때 글로벌 데이터베이스 장애 조치 작업이 수행됩니다.

소스 DB 클러스터:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
```

```
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

복원되거나 복제된 DB 클러스터:

향상된 binlog는 `mysql-bin-changelog.000003` 이후에 비활성화됩니다. 향상된 binlog를 끈 후 작성된 binlog 파일은 복제되며 새로 승격된 DB 클러스터에서 사용할 수 있습니다.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

향상된 빈로그를 위한 Amazon CloudWatch 지표

다음 Amazon CloudWatch 지표는 향상된 binlog가 켜진 경우에만 게시됩니다.

CloudWatch 지표	설명	단위
ChangeLogBytesUsed	향상된 binlog가 사용하는 스토리지 양입니다.	바이트
ChangeLogReadIOPs	5분 간격 내에 향상된 binlog에서 수행된 읽기 I/O 작업의 수입니다.	5분당 개수
ChangeLogWriteIOPs	5분 간격 내에 향상된 binlog에서 수행된 쓰기 디스크 I/O 작업의 수입니다.	5분당 개수

향상된 빈로그의 제한 사항

향상된 binlog가 켜져 있는 경우 Amazon Aurora DB 클러스터에 다음과 같은 제한 사항이 적용됩니다.

- 향상된 binlog는 Aurora MySQL 버전 3.03.1 이상에서만 지원됩니다.
- 프라이머리 DB 클러스터에 작성된 향상된 binlog 파일은 복제되거나 복원된 DB 클러스터에 복사되지 않습니다.
- Amazon Aurora Global Database와 함께 사용하면 프라이머리 DB 클러스터의 향상된 binlog 파일이 보조 DB 클러스터에 복제되지 않습니다. 따라서 장애 조치 프로세스 후에는 이전 binlog 데이터를 새 프라이머리 DB 클러스터에서 사용할 수 없습니다.
- 다음 binlog 구성 파라미터는 무시됩니다.
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- 데이터베이스에서 손상된 테이블을 삭제하거나 이름을 바꿀 수 없습니다. 이 테이블을 삭제하려면 지원에 문의하세요.
- 향상된 binlog가 켜지면 binlog I/O 캐시가 비활성화됩니다. 자세한 내용은 Aurora MySQL의 이진수 로그 복제 최적화 단원을 참조하십시오.

> **Note**
>
> 향상된 binlog는 binlog I/O 캐시와 유사한 읽기 성능 향상 및 더 나은 쓰기 성능 개선을 제공합니다.

- 역추적 기능은 지원되지 않습니다. 다음과 같은 조건에서는 향상된 binlog를 DB 클러스터에서 켤 수 없습니다.
  - 현재 역추적 기능이 활성화된 DB 클러스터.
  - 이전에 역추적 기능이 활성화되었지만 비활성화되지 않은 DB 클러스터
  - 역추적 기능이 활성화된 소스 DB 클러스터 또는 스냅샷에서 복원된 DB 클러스터.

GTID 기반 복제 사용

아래 내용에서는 Aurora MySQL 클러스터와 외부 소스 간 이진 로그(binlog) 복제를 통해 전역 트랜잭션 식별자(GTID)를 사용하는 방법이 나와 있습니다.

> **Note**
>
> Aurora의 경우 외부 MySQL 데이터베이스로 또는 외부 MySQL 데이터베이스로부터 binlog 복제를 사용하는 Aurora MySQL 클러스터에서만 이 기능을 사용할 수 있습니다. 다른 데이터베이스는 Amazon RDS MySQL 인스턴스, 온프레미스 MySQL 데이터베이스 또는 다른 AWS 리전의 Aurora DB 클러스터일 수 있습니다. 이러한 종류의 복제를 구성하는 방법에 대해 알아보려면 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하세요.

binlog 복제를 사용하고 있지만 MySQL을 사용한 GTID 기반 복제에 대해 잘 알지 못하는 경우 MySQL 설명서의 전역 트랜잭션 식별자를 사용한 복제를 참조하세요.

Aurora MySQL 버전 2 및 3에서 GTID 기반 복제가 지원됩니다.

주제

- 전역 트랜잭션 식별자(GTID) 개요
- GTID 기반 복제 파라미터
- Aurora MySQL 클러스터에 대한 GTID 기반 복제 활성화
- Aurora MySQL DB 클러스터에 대해 GTID 기반 복제 비활성화

전역 트랜잭션 식별자(GTID) 개요

전역 트랜잭션 ID(GTIDs) are unique identifiers generated for committed MySQL transactions. GTID를 사용해 binlog 복제 관련 문제를 더 간편하게 해결할 수 있습니다.

> **Note**
>
> Aurora가 클러스터 내 DB 인스턴스 간에 데이터를 동기화하는 경우 이 복제 메커니즘은 이진 로그(binlog)를 수반하지 않습니다. Aurora MySQL의 경우 GTID 기반 복제는 binlog 복제를 사용해 외부 MySQL 호환 데이터베이스로부터 Aurora MySQL DB 클러스터 안 또는 밖으로 복제할 때만 적용됩니다.

MySQL은 binlog 복제에 다음 두 가지 유형의 트랜잭션을 사용합니다.

- GTID 트랜잭션 – GTID로 식별되는 트랜잭션.

- 익명 트랜잭션 – GTID가 할당되지 않은 트랜잭션.

복제 구성의 GTID는 모든 DB 인스턴스에서 고유합니다. GTID를 사용하면 로그 파일 위치를 참조할 필요가 없기 때문에 복제 구성이 간편해집니다. 또한 GTID를 사용하면 복제된 트랜잭션을 추적하고 소스 인스턴스 및 복제본이 일치하는지를 쉽게 확인할 수 있습니다.

외부 MySQL 호환 데이터베이스에서 Aurora 클러스터로 복제할 때 일반적으로 Aurora에서 GTID 기반 복제를 사용합니다. 온프레미스 또는 Amazon RDS 데이터베이스를 Aurora MySQL로 마이그레이션하는 절차의 일부로 이 복제 구성을 설정할 수 있습니다. 외부 데이터베이스에서 이미 GTID를 사용하는 경우 Aurora 클러스터에 대해 GTID 기반 복제를 활성화하면 복제 프로세스가 간소화됩니다.

먼저 DB 클러스터 파라미터 그룹에 있는 해당 구성 파라미터를 설정하여 Aurora MySQL 클러스터에 대해 GTID 기반 복제를 구성합니다. 그런 다음 이 파라미터 그룹을 클러스터와 연결합니다.

GTID 기반 복제 파라미터

다음 파라미터를 사용하여 GTID 기반 복제를 구성할 수 있습니다.

파라미터	유효한 값	설명
`gtid_mode`	`OFF`, `OFF_PERMISSIVE` , `ON_PERMISSIVE` , `ON`	`OFF`는 새 트랜잭션을 익명 트랜잭션(GTID가 없음)으로 지정하며, 트랜잭션을 복제하려면 익명이어야 합니다. `OFF_PERMISSIVE` 는 새 트랜잭션을 익명 트랜잭션(GTID가 없음)으로 지정하지만, 모든 트랜잭션을 복제할 수 있습니다. `ON_PERMISSIVE` 는 새 트랜잭션을 GTID 트랜잭션으로 지정하지만, 모든 트랜잭션을 복제할 수 있습니다. `ON`은 새 트랜잭션을 GTID 트랜잭션으로 지정하고, 트랜잭션을 복제하려면 GTID 트랜잭션이어야 합니다.
`enforce_gtid_consistency`	`OFF`, `ON`, `WARN`	`OFF`는 트랜잭션이 GTID 일관성을 위반하는 것을 허용합니다.

파라미터	유효한 값	설명
		ON은 트랜잭션이 GTID 일관성을 위반하지 않도록 합니다. WARN은 트랜잭션이 GTID 일관성을 위반하는 것을 허용하지만, 위반이 발생할 경우 경고를 생성합니다.

> **Note**
>
> AWS Management Console 콘솔에서 `gtid_mode` 파라미터는 `gtid-mode`로 표시됩니다.

GTID 기반 복제의 경우 Aurora MySQL DB 클러스터의 DB 클러스터 파라미터 그룹에 대해 이 설정을 사용하십시오.

- `ON` 및 `ON_PERMISSIVE`는 Aurora MySQL 클러스터에서 밖으로 복제하는 경우에만 적용됩니다. 이 두 값으로 인해 외부 데이터베이스로 복제되는 트랜잭션에 대해 Aurora DB 클러스터가 GTID를 사용하게 됩니다. `ON`은 외부 데이터베이스에서도 GTID 기반 복제를 사용할 것을 요구합니다. `ON_PERMISSIVE`로 인해 외부 데이터베이스에서 GTID 기반 복제는 선택 사항이 됩니다.
- `OFF_PERMISSIVE`가 설정된 경우 이는 Aurora DB 클러스터가 외부 데이터베이스에서 안으로 복제하는 것을 수락할 수 있음을 뜻합니다. 외부 데이터베이스가 GTID 기반 복제를 사용하든 사용하지 않든 이렇게 할 수 있습니다.
- `OFF`가 설정된 경우 이는 Aurora DB 클러스터가 GTID 기반 복제를 사용하지 않는 외부 데이터베이스에서 안으로 복제하는 것만을 수락할 수 있음을 뜻합니다.

> **Tip**
>
> 안으로 복제하는 것은 Aurora MySQL 클러스터에 가장 흔한 binlog 복제 시나리오입니다. 안으로 복제하는 경우 GTID 모드를 `OFF_PERMISSIVE`로 설정하는 것이 좋습니다. 이 설정을 통해 복제 원본의 GTID 설정에 관계없이 외부 데이터베이스에서 안으로 복제하는 것이 가능해집니다.

파라미터 그룹에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 단원을 참조하십시오.

Aurora MySQL 클러스터에 대한 GTID 기반 복제 활성화

GTID 기반 복제가 Aurora MySQL DB 클러스터에 대해 활성화되면 GTID 설정은 인바운드 및 아웃바운드 binlog 복제본 모두에 적용됩니다.

Aurora MySQL 클러스터에 대한 GTID 기반 복제를 활성화하려면

1. 다음 파라미터 설정을 사용해 DB 클러스터 파라미터 그룹을 생성 또는 편집하십시오.
   - `gtid_mode` – `ON` 또는 `ON_PERMISSIVE`
   - `enforce_gtid_consistency` – `ON`
2. DB 클러스터 파라미터 그룹을 Aurora MySQL 클러스터와 연결합니다. 이 작업을 수행하려면 Amazon Aurora의 파라미터 그룹의 절차를 따르세요.
3. (선택 사항) GTID를 포함하지 않는 트랜잭션에 GTID를 할당하는 방법을 지정합니다. 이렇게 하려면 mysql.rds_assign_gtids_to_anonymous_transactions(Aurora MySQL 버전 3)에서 저장 프로시저를 호출하면 됩니다.

Aurora MySQL DB 클러스터에 대해 GTID 기반 복제 비활성화

Aurora MySQL DB 클러스터에 대한 GTID 기반 복제를 비활성화합니다. 이렇게 하면 Aurora 클러스터가 GTID 기반 복제를 사용하는 외부 데이터베이스에 대해 인바운드 또는 아웃바운드 binlog 복제를 수행할 수 없습니다.

> **Note**
>
> 다음 절차에서 읽기 전용 복제본은 외부 데이터베이스로의 또는 외부 데이터베이스로부터의 binlog 복제를 포함한 Aurora 구성의 복제 대상을 의미합니다. 읽기 전용 Aurora 복제본 DB 인스턴스를 뜻하는 것은 아닙니다. 예를 들어 Aurora 클러스터가 외부 원본에서 안으로의 복제를 수락하는 경우 Aurora 기본 인스턴스는 binlog 복제에 대해 읽기 전용 복제본의 역할을 합니다.

이 단원에 언급된 저장 프로시저에 대한 자세한 내용은 Aurora MySQL 저장 프로시저 참조 단원을 참조하세요.

Aurora MySQL DB 클러스터에 대해 GTID 기반 복제 사용 중지

1. Aurora 복제본에서 다음 프로시저를 실행합니다.

   버전 3의 경우

```
CALL mysql.rds_set_source_auto_position(0);
```

버전 2의 경우

```
CALL mysql.rds_set_master_auto_position(0);
```

2. gtid_mode를 ON_PERMISSIVE로 재설정합니다.

   a. Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 gtid_mode가 ON_PERMISSIVE로 설정되어 있는지 확인합니다.

      파라미터 그룹을 사용한 구성 파라미터 설정에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.

   b. Aurora MySQL 클러스터를 다시 시작합니다

3. gtid_mode를 OFF_PERMISSIVE로 재설정합니다.

   a. Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 gtid_mode가 OFF_PERMISSIVE로 설정되어 있는지 확인합니다.

   b. Aurora MySQL 클러스터를 다시 시작합니다

4. Aurora 기본 인스턴스에서 모든 GTID 트랜잭션이 적용될 때까지 기다립니다. 이러한 사항이 적용되었는지 확인하려면 다음 단계를 수행합니다.

   a. Aurora 기본 인스턴스에서 SHOW MASTER STATUS 명령을 실행합니다.

      출력이 다음 출력과 유사해야 합니다.

```
File                          Position
------------------------------------
mysql-bin-changelog.000031      107
------------------------------------
```

      출력에서 파일 및 위치를 메모합니다.

   b. 각 읽기 전용 복제본에서 이전 단계의 소스 인스턴스의 파일 및 위치 정보를 사용하여 다음 쿼리를 실행합니다.

      버전 3의 경우

```
SELECT SOURCE_POS_WAIT('file', position);
```

버전 2의 경우

```
SELECT MASTER_POS_WAIT('file', position);
```

예를 들어 파일 이름이 `mysql-bin-changelog.000031`이고 위치가 `107`일 경우 다음 문을 실행합니다.

버전 3의 경우

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

버전 2의 경우

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. GTID 기반 복제를 비활성화하도록 GTID 파라미터를 재설정합니다.

   a. Aurora MySQL 클러스터와 연결된 DB 클러스터 파라미터 그룹에서 다음과 같이 파라미터가 설정되어 있는지 확인합니다.

      - `gtid_mode` – `OFF`
      - `enforce_gtid_consistency` – `OFF`

   b. Aurora MySQL 클러스터를 다시 시작합니다

Amazon Aurora MySQL DB 클러스터에서 로컬 쓰기 전달 사용

로컬(클러스터 내) 쓰기 전달을 사용하면 애플리케이션이 Aurora 복제본에서 직접 읽기/쓰기 트랜잭션을 발행할 수 있도록 합니다. 그러면 이러한 트랜잭션이 라이터 DB 인스턴스로 전달되어 커밋됩니다. 한 트랜잭션에서 최신 쓰기를 읽는 기능인 쓰기 후 읽기 읽관성이 애플리케이션에 필요한 경우 로컬 쓰기 전달을 사용할 수 있습니다.

읽기 전용 복제본은 라이터로부터 비동기식으로 업데이트를 받습니다. 쓰기 전달을 사용하지 않으면 쓰기 후 읽기 일관성이 필요한 모든 읽기를 라이터 DB 인스턴스에서 처리해야 합니다. 또는 확장성을 목적으로 여러 읽기 전용 복제본을 활용하기 위해 복잡한 사용자 지정 애플리케이션 로직을 개발해야 합니다. 애플리케이션이 트래픽을 올바른 엔드포인트로 전송하기 위해 두 세트의 데이터베이스 연결을 유지하면서 모든 읽기 및 쓰기 트래픽을 완전히 분할해야 합니다. 쿼리가 애플리케이션 내의 단일 논리적 세션 또는 트랜잭션의 일부인 경우 이러한 개발 오버헤드로 인해 애플리케이션 설계가 복잡해집니다. 게다가 읽기 전용 복제본마다 복제 지연이 다를 수 있기 때문에 데이터베이스의 모든 인스턴스에서 전역 읽기 일관성을 유지하기가 어렵습니다.

쓰기 전달을 사용하면 이러한 트랜잭션을 분할하거나 라이터에게만 전송할 필요가 없으므로 애플리케이션 개발이 간소화됩니다. 이 새로운 기능을 사용하면 트랜잭션에서 최신 쓰기를 읽어야 하고 쓰기 지연 시간에 민감하지 않은 워크로드에 대한 읽기 규모를 쉽게 달성할 수 있습니다.

로컬 쓰기 전달은 보조 DB 클러스터의 쓰기를 Aurora 글로벌 데이터베이스의 기본 DB 클러스터로 전달하는 전역 쓰기 전달과는 다릅니다. Aurora 글로벌 데이터베이스의 일부인 DB 클러스터에서 로컬 쓰기 전달을 사용할 수 있습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 단원을 참조하십시오.

쓰기 전달에는 Aurora MySQL 버전 3.04 이상이 필요합니다.

주제

- [전달된 트랜잭션 및 쿼리 식별]

로컬 쓰기 전달 활성화

기본적으로 Aurora MySQL DB 클러스터에는 로컬 쓰기 전달이 활성화되어 있지 않습니다. 인스턴스 수준이 아닌 클러스터 수준에서 직접 로컬 쓰기 전달을 활성화합니다.

> **Important**
>
> 또한 이진 로깅을 사용하지만 쓰기 작업이 소스 AWS 리전에 전달되지 않는 리전 간 읽기 전용 복제본에 대해 로컬 쓰기 전달을 활성화할 수 있습니다. 이는 binlog 읽기 전용 복제본 클러스터의 라이터 DB 인스턴스로 전달됩니다.
> 이 방법은 보조 AWS 리전의 binlog 읽기 전용 복제본에 쓰기 위한 사용 사례가 있는 경우에만 사용하세요. 그렇지 않으면 복제된 데이터 세트가 서로 일치하지 않는 '스플릿 브레인(split-brain)' 시나리오가 발생할 수 있습니다.
> 꼭 필요한 경우가 아니면 리전 간 읽기 전용 복제본에 로컬 쓰기 전달을 사용하기보다는 글로벌 데이터베이스에 전역 쓰기 전달을 사용하는 것이 좋습니다. 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용] 단원을 참조하십시오.

콘솔

DB 클러스터를 만들거나 수정할 때 AWS Management Console을 사용하여 읽기 전용 복제본 쓰기 전달 아래의 로컬 쓰기 전달 켜기 확인란을 선택합니다.

AWS CLI

AWS CLI를 사용하여 쓰기 전달을 활성화하려면 `--enable-local-write-forwarding` 옵션을 사용합니다. 이 옵션은 `create-db-cluster` 명령을 사용하여 새 DB 클러스터를 생성할 때 작동합니다. `modify-db-cluster` 명령을 사용하여 기존 DB 클러스터를 수정할 때도 작동합니다. 이러한 동일한 CLI 명령과 함께 `--no-enable-local-write-forwarding` 옵션을 사용하여 쓰기 전달을 비활성화할 수 있습니다.

다음 예시에서는 쓰기 전달이 활성화된 Aurora MySQL DB 클러스터를 생성합니다.

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
```

```
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

그런 다음 쓰기 전달을 사용할 수 있도록 라이터 및 리더 DB 인스턴스를 생성합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

RDS API

Amazon RDS API를 사용하여 쓰기 전달을 활성화하려면 EnableLocalWriteForwarding 파라미터를 true로 설정합니다. 이 파라미터는 CreateDBCluster 작업을 사용하여 새 DB 클러스터를 생성할 때 작동합니다. ModifyDBCluster 작업을 사용하여 기존 DB 클러스터를 수정할 때도 작동합니다. EnableLocalWriteForwarding 파라미터를 false로 설정하여 쓰기 전달을 비활성화할 수 있습니다.

데이터베이스 세션에 대한 쓰기 전달 활성화

aurora_replica_read_consistency 파라미터는 쓰기 전달을 가능하게 하는 DB 파라미터 및 DB 클러스터 파라미터입니다. 읽기 일관성 수준에 대해 EVENTUAL, SESSION 또는 GLOBAL을 지정할 수 있습니다. 일관성 수준에 대한 자세한 내용은 쓰기 전달을 위한 읽기 일관성 단원을 참조하세요.

이 파라미터에는 다음과 같은 규칙이 적용됩니다.

- 기본값은 ''(Null)입니다.
- aurora_replica_read_consistency를 EVENTUAL, SESSION 또는 GLOBAL로 설정하는 경우에만 쓰기 전달을 사용할 수 있습니다. 이 파라미터는 쓰기 전달이 활성화되어 있는 DB 클러스터의 리더 인스턴스에만 적용됩니다.
- 다중 명령문 트랜잭션 내에서 이 변수가 비어 있는 경우 설정하거나 이미 설정된 경우 설정을 취소할 수 없습니다. 이러한 트랜잭션 중에 하나의 유효한 값에서 다른 유효한 값으로 변경할 수는 있지만 이 작업은 권장되지 않습니다.

DB 클러스터에 쓰기 전달이 활성화되어 있는지 확인

DB 클러스터에서 쓰기 전달을 사용할 수 있는지 확인하려면 클러스터에 LocalWriteForwardingStatus 속성이 enabled로 되어 있는지 확인합니다.

AWS Management Console에서 클러스터의 세부 정보 페이지 구성 탭에서 로컬 읽기 복제본 쓰기 전달이 활성화됨 상태인 것을 볼 수 있습니다.

모든 클러스터에 대한 쓰기 전달 설정의 상태를 보려면 다음 AWS CLI 명령을 실행합니다.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

DB 클러스터는 `LocalWriteForwardingStatus`에 대해 다음과 같은 값을 가질 수 있습니다.

- `disabled` - 쓰기 전달이 비활성화되었습니다.
- `disabling` - 쓰기 전달을 비활성화하는 중입니다.
- `enabled` - 쓰기 전달이 활성화되었습니다.
- `enabling` - 쓰기 전달을 활성화하는 중입니다.
- `null` - 이 DB 클러스터에서는 쓰기 전달을 사용할 수 없습니다.
- `requested` - 쓰기 전달이 요청되었지만 아직 활성 상태가 아닙니다.

쓰기 전달과 애플리케이션 및 SQL 호환성

다음과 같은 종류의 SQL 문을 쓰기 전달과 함께 사용할 수 있습니다.

- `INSERT`, `DELETE` 및 `UPDATE`와 같은 데이터 조작 언어(DML) 문입니다. 다음 설명과 같이, 쓰기 전달과 함께 사용할 수 있는 이러한 명령문의 속성에는 몇 가지 제한 사항이 있습니다.

- `SELECT ... LOCK IN SHARE MODE` 및 `SELECT FOR UPDATE` 문입니다.
- `PREPARE` 및 `EXECUTE` 문입니다.

특정 명령문은 쓰기 전달과 함께 DB 클러스터에서 사용할 때 허용되지 않거나 부실 결과를 생성할 수 있습니다. 또한 사용자 정의 함수 및 사용자 정의 프로시저는 지원되지 않습니다. 따라서 DB 클러스터에 대해 기본적으로 `EnableLocalWriteForwarding` 설정이 비활성화됩니다. 활성화하기 전에 애플리케이션 코드가 이러한 제한 사항의 영향을 받지 않는지 확인하세요.

쓰기 전달에 사용하는 SQL 문에는 다음과 같은 제한 사항이 적용됩니다. 경우에 따라서는 쓰기 전달이 활성화된 DB 클러스터에서 명령문을 사용할 수 있습니다. 이 접근 방식은 `aurora_replica_read_consistency` 구성 파라미터를 통해 세션 내에서 쓰기 전달을 활성화하지 않은 경우에 작동합니다. 쓰기 전달로 인해 허용되지 않는 명령문을 사용하려고 하면 다음과 비슷한 오류 메시지가 표시됩니다.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

데이터 정의 언어(DDL)

라이터 DB 인스턴스에 연결하여 DDL 명령문을 실행합니다. 리더 DB 인스턴스에서 실행할 수 없습니다.

임시 테이블의 데이터를 사용하여 영구 테이블 업데이트

쓰기 전달이 활성화된 DB 클러스터에서 임시 테이블을 사용할 수 있습니다. 하지만 명령문이 임시 테이블을 참조하는 경우 DML 문을 사용하여 영구 테이블을 수정할 수 없습니다. 예를 들어 임시 테이블에서 데이터를 가져오는 `INSERT ... SELECT` 문을 사용할 수 없습니다.

XA 트랜잭션

세션 내에서 쓰기 전달이 활성화되어 있으면 DB 클러스터에서 다음 명령문을 사용할 수 없습니다. 쓰기 전달이 활성화되지 않은 DB 클러스터나 `aurora_replica_read_consistency` 설정이 비어 있는 세션 내에서 이러한 명령문을 사용할 수 있습니다. 세션 내에서 쓰기 전달을 활성화하기 전에 코드에서 이러한 명령문을 사용하는지 확인하세요.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
```

```
XA RECOVER [CONVERT XID]
```

영구 테이블에 대한 LOAD 문

쓰기 전달이 활성화된 DB 클러스터에서는 다음 명령문을 사용할 수 없습니다.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

플러그인 문

쓰기 전달이 활성화된 DB 클러스터에서는 다음 명령문을 사용할 수 없습니다.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT 문

세션 내에서 쓰기 전달이 활성화되어 있으면 DB 클러스터에서 다음 명령문을 사용할 수 없습니다. 쓰기 전달이 설정되어 있지 않은 DB 클러스터나 `aurora_replica_read_consistency` 설정이 비어 있는 세션 내에서 이러한 명령문을 사용할 수 있습니다. 세션 내에서 쓰기 전달을 활성화하기 전에 코드에서 이러한 명령문을 사용하는지 확인하세요.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

쓰기 전달을 위한 격리 수준

쓰기 전달을 사용하는 세션에서는 `REPEATABLE READ` 격리 수준만 사용할 수 있습니다. Aurora 복제본에서도 `READ COMMITTED` 격리 수준을 사용할 수 있지만, 쓰기 전달에는 해당 격리 수준이 작동하지 않습니다. `REPEATABLE READ` 및 `READ COMMITTED` 격리 수준에 대한 자세한 내용은 [Aurora MySQL 격리 수준](#) 단원을 참조하십시오.

쓰기 전달을 위한 읽기 일관성

DB 클러스터에서 읽기 일관성 정도를 제어할 수 있습니다. 읽기 일관성 수준은 라이터에서 일부 또는 모든 변경 사항이 복제되도록 각 읽기 작업 전에 DB 클러스터가 대기하는 기간을 결정합니다. 읽기 일관성 수준을 조정하여 세션에서 전달된 모든 쓰기 작업이 후속 쿼리 전에 DB 클러스터에 표시되도록

할 수 있습니다. 또한 이 설정을 사용하여 DB 클러스터의 쿼리에 항상 라이터의 최신 업데이트가 표시되게 할 수 있습니다. 이 설정은 다른 세션이나 다른 클러스터에서 제출한 쿼리에도 적용됩니다. 애플리케이션에 이러한 유형의 동작을 지정하려면 aurora_replica_read_consistency DB 파라미터 또는 DB 클러스터 파라미터의 값을 선택합니다.

> **Important**
>
> 쓰기를 전달하려는 경우 항상 aurora_replica_read_consistency DB 파라미터 또는 DB 클러스터 파라미터를 설정하세요. 설정하지 않으면 Aurora가 쓰기를 전달하지 않습니다. 이 파라미터는 기본적으로 비어 있으므로, 이 파라미터를 사용할 때 특정 값을 선택하세요. aurora_replica_read_consistency 파라미터는 쓰기 전달이 활성화된 DB 클러스터 또는 인스턴스에만 영향을 미칩니다.

일관성 수준을 높이면 애플리케이션은 변경 사항이 DB 인스턴스 간에 전파될 때까지 더 오랜 시간 동안 대기합니다. 빠른 응답 시간과 쿼리가 실행되기 전에 다른 DB 인스턴스에서 변경한 내용을 완전히 사용할 수 있도록 보장하는 것 사이에서 균형을 선택할 수 있습니다.

aurora_replica_read_consistency 파라미터에 대해 다음 값을 지정할 수 있습니다.

- EVENTUAL - 쓰기 작업이 라이터 DB 인스턴스에서 수행될 때까지 동일한 세션에서의 쓰기 작업 결과가 표시되지 않습니다. 쿼리는 업데이트된 결과를 사용할 수 있을 때까지 대기하지 않습니다. 따라서 명령문의 타이밍 및 복제 지연의 양에 따라 이전 데이터 또는 업데이트된 데이터를 검색하게 될 수 있습니다. 이는 쓰기 전달을 사용하지 않는 Aurora MySQL DB 클러스터와 동일한 일관성입니다.
- SESSION - 쓰기 전달을 사용하는 모든 쿼리에 해당 세션에서 이루어진 모든 변경 결과가 표시합니다. 변경 내용은 트랜잭션이 커밋되었는지 여부와 상관없이 표시됩니다. 필요한 경우 전달된 쓰기 작업의 결과가 복제될 때까지 쿼리가 대기합니다.
- GLOBAL - DB 클러스터의 모든 세션 및 인스턴스에서 커밋된 모든 변경 사항이 세션에 표시됩니다. 각 쿼리는 세션 지연 양에 따라 다른 기간 동안 대기할 수 있습니다. 쿼리가 시작된 시간을 기준으로, DB 클러스터가 라이터에서 커밋된 모든 데이터로 최신 상태가 되면 쿼리가 진행됩니다.

쓰기 전달과 관련된 파라미터에 대한 자세한 내용은 쓰기 전달에 대한 구성 파라미터 섹션을 참조하세요.

> **Note**
>
> aurora_replica_read_consistency를 세션 변수로 사용할 수도 있습니다. 예를 들면 다음과 같습니다.

```
mysql> set aurora_replica_read_consistency = 'session';
```

쓰기 전달 사용의 예제

다음 예시에서는 INSERT 명령문 다음에 SELECT 명령문을 실행하는 aurora_replica_read_consistency 파라미터의 효과를 보여줍니다. aurora_replica_read_consistency의 값과 명령문의 타이밍에 따라 결과가 다를 수 있습니다.

일관성을 높이기 위해 SELECT 문을 실행하기 전에 잠시 기다립니다. 또는 Aurora은(는) SELECT을(를) 계속하기 전에 결과 복제가 완료될 때까지 자동으로 대기할 수 있습니다.

DB 파라미터 설정에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 섹션을 참조하세요.

Example **aurora_replica_read_consistency**를 **EVENTUAL**로 설정

INSERT 명령문을 실행한 직후에 SELECT 명령문을 실행하면 새 행이 삽입되기 전의 행 수로 COUNT(*) 값이 반환됩니다. 잠시 후에 SELECT를 다시 실행하면 업데이트된 행 수가 반환됩니다. SELECT 명령문은 대기하지 않습니다.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
```

```
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency**를 **SESSION**으로 설정

SELECT 명령문 직후 INSERT를 실행하면 INSERT 명령문으로 인한 변경 사항이 표시될 때까지 대기합니다. 후속 SELECT 명령문은 대기하지 않습니다.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

읽기 일관성 설정이 여전히 SESSION인 경우, INSERT 문을 수행한 후 짧은 대기 시간을 도입하면 다음 번 SELECT 문이 실행되는 시간까지 업데이트된 행 수를 사용할 수 있습니다.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
```

```
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency**를 **GLOBAL**로 설정

각 SELECT 명령문은 쿼리를 수행하기 전에 명령문의 시작 시간을 기준으로 모든 데이터 변경 사항이 표시되도록 대기합니다. 각 SELECT 명령문의 대기 시간은 복제 지연의 양에 따라 달라집니다.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

쓰기 전달을 사용하여 멀티파트 문 실행

DML 명령문은 INSERT ... SELECT 명령문 또는 DELETE ... WHERE 명령문과 같이 여러 부분으로 구성될 수 있습니다. 이 경우 전체 명령문이 라이터 DB 인스턴스로 전달되어 그 위치에서 실행됩니다.

쓰기 전달을 사용한 트랜잭션

트랜잭션 액세스 모드가 읽기 전용으로 설정된 경우 쓰기 전달이 사용되지 않습니다. SET TRANSACTION 문 또는 START TRANSACTION 문을 사용하여 트랜잭션에 대한 액세스 모드를 지정할 수 있습니다. [transaction_read_only](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_transaction_read_only) 세션 변수의 값을 변경하여 트랜잭션 액세스 모드를 지정할 수도 있습니다. 쓰기 전달이 활성화된 DB 클러스터에 연결되어 있는 동안에만 이 세션 값을 변경할 수 있습니다.

장기 실행 트랜잭션이 상당한 기간 동안 명령문을 실행하지 않으면 유휴 제한 시간을 초과할 수 있습니다. 이 기간의 기본값은 1분입니다. 최대 1일로 늘리도록 aurora_fwd_writer_idle_timeout 파라미터를 설정할 수 있습니다. 유휴 시간 제한을 초과하는 트랜잭션은 라이터 인스턴스에서 취소됩니다. 다음 번 후속 명령문을 제출하면 시간 제한 오류가 수신됩니다. 그런 다음 Aurora는 트랜잭션을 롤백합니다.

이러한 유형의 오류는 쓰기 전달을 사용할 수 없는 다른 경우에 발생할 수 있습니다. 예를 들어 DB 클러스터를 다시 시작하거나 쓰기 전달을 비활성화하면 Aurora가 쓰기 전달을 사용하는 트랜잭션을 취소합니다.

로컬 쓰기 전달을 사용하는 클러스터의 라이터 인스턴스가 다시 시작되면 로컬 쓰기 전달을 사용하는 리더 인스턴스의 활성 상태이며 전달된 트랜잭션 및 쿼리가 모두 자동으로 닫힙니다. 라이터 인스턴스를 다시 사용할 수 있게 되면 이러한 트랜잭션을 다시 시도할 수 있습니다.

쓰기 전달에 대한 구성 파라미터

Aurora DB 파라미터 그룹에는 쓰기 전달 기능에 대한 설정이 포함되어 있습니다. 이러한 파라미터에 대한 자세한 내용은 다음 표에 요약되어 있으며, 표 뒤에 사용 참고 사항이 나와 있습니다.

파라미터	범위	유형	기본값	유효값
aurora_fwd_writer_idle_timeout	클러스터	부호 없는 정수	60	1–86,400
aurora_fwd_writer_max_connections_pct	클러스터	부호 없는 긴 정수	10	0–90
aurora_replica_read_consistency	클러스터 또는 인스턴스	Enum	"(Null)	EVENTUAL, SESSION, GLOBAL

수신되는 쓰기 요청을 제어하려면 다음 설정을 사용하세요.

- `aurora_fwd_writer_idle_timeout` - 라이터 DB 인스턴스가 연결을 닫기 전에 리더 인스턴스로에서 전달된 연결의 활동을 기다리는 시간(초)입니다. 세션이 이 기간 동안 유휴 상태로 유지되면 Aurora에서 세션이 취소됩니다.
- `aurora_fwd_writer_max_connections_pct` - 리더 인스턴스에서 전달된 쿼리를 처리하기 위해 라이터 DB 인스턴스에서 사용할 수 있는 데이터베이스 연결의 상한입니다. 이 값은 라이터에 대한 `max_connections` 설정의 백분율로 표시됩니다. 예를 들어 `max_connections`가 800이고 `aurora_fwd_master_max_connections_pct` 또는`aurora_fwd_writer_max_connections_pct`가 10이면 라이터는 최대 80개의 동시 전달 세션을 허용합니다. 이러한 연결은 `max_connections` 설정을 통해 관리되는 동일한 연결 풀에서 발생합니다.

  이 설정은 쓰기 전달이 활성화된 경우 라이터에만 적용됩니다. 값을 줄이면 기존 연결은 영향을 받지 않습니다. Aurora는 DB 클러스터에서 새 연결을 생성하려고 할 때 설정의 새 값을 고려합니다. 기본값은 `max_connections` 값의 10% 를 나타내는 10입니다.

> **Note**
>
> `aurora_fwd_writer_idle_timeout` 및 `aurora_fwd_writer_max_connections_pct`는 DB 클러스터 파라미터이므로 각 클러스터의 모든 DB 인스턴스에서 이러한 파라미터의 값이 동일해야 합니다.

`aurora_replica_read_consistency`에 대한 자세한 정보는 [쓰기 전달을 위한 읽기 일관성](#) 섹션을 참조하십시오.

DB 파라미터 그룹에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 단원을 참조하십시오.

쓰기 전달을 위한 Amazon CloudWatch 지표 및 Aurora MySQL 상태 변수

다음 Amazon CloudWatch 지표와 Aurora MySQL 상태 변수는 Aurora MySQL에 대한 쓰기 전달을 사용할 때 적용됩니다. Aurora MySQL 라이터 및 리더 DB 인스턴스의 지표를 자세히 알아보려면 다음 주제를 참조하세요.

주제

- [Aurora MySQL 라이터 DB 인스턴스의 쓰기 전달에 대한 지표](#)
- [Aurora MySQL 리더 DB 인스턴스의 쓰기 전달에 대한 지표](#)

Aurora MySQL 라이터 DB 인스턴스의 쓰기 전달에 대한 지표

다음 Amazon CloudWatch 지표는 하나 이상의 DB 클러스터에서 쓰기 전달을 사용할 때 적용됩니다. 이러한 지표는 모두 라이터 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위	설명
`ForwardingWriterDMLLatency`	밀리초	라이터 DB 인스턴스에서 전달된 각 DML 문을 처리하는 평균 시간입니다. DB 클러스터가 쓰기 요청을 전달하는 시간이나 변경 사항을 라이터에 다시 복제하는 시간은 포함되지 않습니다.
`ForwardingWriterDMLThroughput`	초당 개수	이 라이터 DB 인스턴스에서 초당 처리되는 전달된 DML 문 수입니다.
`ForwardingWriterOpenSessions`	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다.

다음 Aurora MySQL 상태 변수는 하나 이상의 DB 클러스터에서 쓰기 전달을 사용할 때 적용됩니다. 이러한 상태 변수는 모두 라이터 DB 인스턴스에서 측정됩니다.

Aurora MySQL 상태 변수	단위	설명
`Aurora_fwd_writer_dml_stmt_count`	개수	이 라이터 DB 인스턴스로 전달된 총 DML 문 수입니다.
`Aurora_fwd_writer_dml_stmt_duration`	마이크로초	이 라이터 DB 인스턴스로 전달된 DML 문의 총 기간입니다.
`Aurora_fwd_writer_open_sessions`	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_writer_select_stmt_count	개수	이 라이터 DB 인스턴스로 전달된 총 SELECT 문 수입니다.
Aurora_fwd_writer_select_stmt_duration	마이크로초	이 라이터 DB 인스턴스로 전달된 SELECT 문의 총 기간입니다.

Aurora MySQL 리더 DB 인스턴스의 쓰기 전달에 대한 지표

다음 CloudWatch 지표는 쓰기 전달이 활성화된 DB 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위	설명
ForwardingReplicaDMLLatency	밀리초	복제본에 있는 전달된 DML의 평균 응답 시간입니다.
ForwardingReplicaDMLThroughput	초당 개수	초당 처리되는 전달된 DML 문 수입니다.
ForwardingReplicaOpenSessions	개수	리더 DB 인스턴스에서 쓰기 전달을 사용하는 세션 수입니다.
ForwardingReplicaReadWaitLatency	밀리초	리더 DB 인스턴스의 SELECT 명령문이 라이터를 따라잡기 위해 대기하는 평균 대기 시간입니다. 리더 DB 인스턴스가 쿼리를 처리하기 전에 대기하는 정도는 aurora_replica_read_consistency 설정에 따라 다릅니다.
ForwardingReplicaReadWaitThroughput	초당 개수	쓰기를 전달하는 모든 세션에서 초당 처리되는 SELECT 문의 총 수입니다.

CloudWatch 지표	단위	설명
ForwardingReplicaSelectLatency	밀리초	전달된 SELECT 지연 시간으로, 모니터링 기간 내에 전달된 모든 SELECT 명령문의 평균입니다.
ForwardingReplicaSelectThroughput	초당 개수	모니터링 기간 내 전달된 SELECT 초당 평균 처리량입니다.

다음 Aurora MySQL 상태 변수는 쓰기 전달이 활성화된 DB 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_replica_dml_stmt_count	개수	이 리더 DB 인스턴스에서 전달된 총 DML 문 수입니다.
Aurora_fwd_replica_dml_stmt_duration	마이크로초	이 리더 DB 인스턴스에서 전달된 모든 DML 문의 총 기간입니다.
Aurora_fwd_replica_errors_session_limit	개수	다음 오류 조건 중 하나로 인해 기본 클러스터에서 거부한 세션 수입니다. • 라이터 가득 참 • 진행 중인 전달된 명령문이 너무 많음
Aurora_fwd_replica_open_sessions	개수	리더 DB 인스턴스에서 쓰기 전달을 사용하는 세션 수입니다.
Aurora_fwd_replica_read_wait_count	개수	이 리더 DB 인스턴스에서 총 쓰기 후 읽기 대기 횟수입니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_replica_read_wait_duration	마이크로초	이 리더 DB 인스턴스의 읽기 일관성 설정으로 인한 총 대기 기간입니다.
Aurora_fwd_replica_select_stmt_count	개수	이 리더 DB 인스턴스에서 전달된 총 SELECT 문 수입니다.
Aurora_fwd_replica_select_stmt_duration	마이크로초	이 리더 DB 인스턴스에서 전달된 SELECT 문의 총 기간입니다.

전달된 트랜잭션 및 쿼리 식별

`information_schema.aurora_forwarding_processlist` 테이블을 사용하여 전달된 트랜잭션 및 쿼리를 식별할 수 있습니다. 이 테이블에 대한 자세한 내용은 [information_schema.aurora_forwarding_processlist](#) 섹션을 참조하세요.

다음 예시는 라이터 DB 인스턴스에 있는 전달된 연결을 모두 보여줍니다.

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+--------------------------------+----------------------+-----------------+
| ID  | USER     | HOST                | DB       | COMMAND | TIME | STATE          |
 INFO                                         | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER      | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+--------------------------------+--------------------------------------+
| 648 | myuser   | IP_address:port1    | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2           | my-db-cluster         | us-west-2      |
| 650 | myuser   | IP_address:port2    | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |                639 | my-
db-cluster-instance-2           | my-db-cluster         | us-west-2      |
```

```
+-----+----------+--------------------+----------+---------+------+--------------
+------------------------------------------+--------------+--------------------
+--------------------------------+---------------------+----------------+
```

전달하는 리더 DB 인스턴스에서 `SHOW PROCESSLIST`를 실행하여 이러한 라이터 DB 연결과 관련된 스레드를 볼 수 있습니다. 라이터의 `REPLICA_SESSION_ID` 값, 637, 639는 리더의 `Id` 값과 같습니다.

```
mysql> select @@aurora_server_id;

+--------------------------------+
| @@aurora_server_id             |
+--------------------------------+
| my-db-cluster-instance-2       |
+--------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State         | Info
                                          |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+
12 rows in set (0.00 sec)
```

Amazon Aurora MySQL을 다른 AWS 서비스와 통합

Aurora MySQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora MySQL이 다른 AWS 서비스와 통합되었습니다. Aurora MySQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- AWS Lambda 또는 `lambda_sync` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 `lambda_async` 함수를 호출하십시오. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 섹션을 참조하세요.
- `LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 명령을 사용하여 Amazon Simple Storage Service(Amazon S3) 버킷에 저장된 텍스트 또는 XML 파일에서 DB 클러스터로 데이터를 로드하십시오. 자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 섹션을 참조하세요.
- `SELECT INTO OUTFILE S3` 명령을 사용하여 DB 클러스터의 데이터를 Amazon S3 버킷에 저장된 텍스트 파일에 저장하십시오. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 섹션을 참조하세요.
- Application Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거하십시오. 자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 단원을 참조하십시오.
- Amazon Comprehend로 감성 분석을 수행하거나 SageMaker AI로 광범위한 기계 학습 알고리즘을 수행하세요. 자세한 내용은 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.

Aurora에는 AWS Identity and Access Management(IAM)를 사용하여 다른 AWS 서비스에 액세스할 수 있는 기능이 있습니다. 필요한 권한이 있는 IAM 역할을 만든 다음 해당 역할을 DB 클러스터와 연결하여 다른 AWS 서비스에 대한 액세스 권한을 부여합니다. 사용자를 대신하여 Aurora MySQL DB 클러스터에서 다른 AWS 서비스에 액세스하도록 허용하는 방법에 대한 세부 정보 및 지침은 Amazon Aurora MySQL이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 섹션을 참조하세요.

Amazon Aurora MySQL이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여

Aurora MySQL DB 클러스터가 사용자를 대신하여 다른 서비스에 액세스하려면 AWS Identity and Access Management(IAM) 역할을 생성하고 구성하세요. 이 역할은 다른 AWS 서비스에 액세스할 수 있도록 DB 클러스터의 데이터베이스 사용자에 권한을 부여합니다. 자세한 내용은 AWS 서비스에 액세스할 수 있는 IAM 역할 설정 섹션을 참조하세요.

또한 대상 AWS 서비스로의 아웃바운드 연결을 허용하도록 Aurora DB 클러스터를 구성해야 합니다. 자세한 내용은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

이렇게 하면 데이터베이스 사용자가 다른 AWS 서비스를 사용하여 다음 작업을 수행할 수 있습니다.

- AWS Lambda 또는 `lambda_sync` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 `lambda_async` 함수를 호출하십시오. 또는 AWS Lambda 프로시저를 사용하여 비동기적으로 `mysql.lambda_async` 함수를 호출하십시오. 자세한 내용은 Aurora MySQL 네이티브 함수로 Lambda 함수 호출 섹션을 참조하세요.
- `LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 문을 사용하여 Amazon S3 버킷에 저장된 텍스트 또는 XML 파일에서 DB 클러스터로 데이터를 로드할 수 있습니다. 자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 섹션을 참조하세요.
- `SELECT INTO OUTFILE S3` 문을 사용하여 DB 클러스터의 데이터를 Amazon S3 버킷에 저장된 텍스트 파일에 저장합니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 섹션을 참조하세요.
- 로그 데이터를 Amazon CloudWatch Logs MySQL로 내보냅니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 섹션을 참조하세요.
- Application Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거하십시오. 자세한 내용은 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling 단원을 참조하십시오.

AWS 서비스에 액세스할 수 있는 IAM 역할 설정

Aurora DB 클러스터가 다른 AWS 서비스에 액세스하도록 허용하려면 다음을 수행합니다.

1. AWS 서비스에 권한을 부여하는 IAM 정책을 만듭니다. 자세한 내용은 다음 항목을 참조하십시오.
   - Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성
   - AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성
   - CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성
   - AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성
2. IAM 역할을 만들고 만든 정책을 연결합니다. 자세한 내용은 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 섹션을 참조하세요.
3. IAM 역할을 Aurora DB 클러스터와 연결합니다. 자세한 내용은 IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 단원을 참조하십시오.

Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora는 Aurora DB 클러스터로 데이터를 로드하거나 Aurora DB 클러스터로부터 데이터를 저장하기 위해 Amazon S3 리소스에 액세스할 수 있습니다. 하지만 Aurora가 Amazon S3에 액세스하도록 허용하는 버킷 및 객체 권한을 제공하는 IAM 정책을 먼저 생성해야 합니다.

다음 표에는 사용자 대신 Amazon S3 버킷에 액세스할 수 있는 Aurora 기능과 각 기능에 요구되는 최소 필요 버킷 및 객체 권한이 나와 있습니다.

기능	버킷 권한	객체 권한
`LOAD DATA FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`
`LOAD XML FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`

`SELECT INTO OUTFILE S3`	`ListBucket`	`AbortMultipartUpload`
		`DeleteObject`
		`GetObject`
		`ListMultipartUploadParts`
		`PutObject`

다음 정책은 Aurora에서 사용자를 대신하여 Amazon S3 버킷에 액세스하는 데 필요할 수 있는 권한을 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
```

```
                "Effect": "Allow",
                "Action": [
                    "s3:PutObject",
                    "s3:GetObject",
                    "s3:AbortMultipartUpload",
                    "s3:ListBucket",
                    "s3:DeleteObject",
                    "s3:GetObjectVersion",
                    "s3:ListMultipartUploadParts"
                ],
                "Resource": [
                    "arn:aws:s3:::amzn-s3-demo-bucket/*",
                    "arn:aws:s3:::amzn-s3-demo-bucket"
                ]
            }
        ]
}
```

> **Note**
>
> `Resource` 값에 두 항목을 모두 포함해야 합니다. Aurora에는 버킷 자체와 버킷 내부의 모든 객체에 대한 권한이 필요합니다.
> 사용 사례에 따라 샘플 정책에 모든 권한을 추가하지 않아도 됩니다. 또한 다른 권한이 필요할 수 있습니다. 예를 들어 Amazon S3 버킷이 암호화된 경우에는 `kms:Decrypt` 권한을 추가해야 합니다.

다음 단계를 사용하여 사용자를 대신하여 Amazon S3 버킷에 액세스하는 데 필요한 최소 권한을 Aurora에 제공하는 IAM 정책을 생성할 수 있습니다. Aurora이 모든 Amazon S3 버킷에 액세스할 수 있도록 허용하려면 이러한 단계를 건너뛰고 직접 생성하지 않고 `AmazonS3ReadOnlyAccess` 또는 `AmazonS3FullAccess` 미리 정의된 IAM 정책을 사용하십시오.

Amazon S3 리소스에 대한 액세스 권한을 부여하는 IAM 정책 생성 방법

1. IAM 관리 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. [Create policy]를 선택합니다.
4. [Visual editor] 탭에서 [Choose a service]를 선택한 다음 [S3]을 선택합니다.

5. 작업에서 Expand all(모두 확장)을 선택한 다음 IAM 정책에 필요한 버킷 권한 및 객체 권한을 선택합니다.

   객체 권한은 Amazon S3의 객체 작업에 대한 권한으로, 버킷 자체가 아닌 버킷의 객체에 대해 부여해야 합니다. Amazon S3의 객체 작업의 권한에 대한 자세한 내용은 [객체 작업에 대한 권한](#)을 참조하십시오.

6. 리소스를 선택하고 bucket(버킷)에 대해 Add ARN(ARN 추가)을 선택합니다.
7. [Add ARN(s)] 대화 상자에서 리소스에 대한 세부 정보를 제공하고 [Add]를 선택합니다.

   액세스를 허용할 Amazon S3 버킷을 지정하십시오. 예를 들어, Aurora가 *amzn-s3-demo-bucket*이라는 Amazon S3 버킷에 액세스할 수 있게 허용하려면 Amazon 리소스 이름(ARN) 값을 `arn:aws:s3:::amzn-s3-demo-bucket`으로 설정하세요.

8. [object] 리소스가 나열되면 객체에 대해 [Add ARN]을 선택합니다.
9. [Add ARN(s)] 대화 상자에서 리소스에 대한 세부 정보를 제공합니다.

   Amazon S3 버킷의 경우 액세스를 허용할 Amazon S3 버킷을 지정하십시오. 객체의 경우 [Any]를 선택하여 버킷의 모든 객체에 권한을 허용할 수 있습니다.

   > **Note**
   >
   > Amazon Resource Name(ARN)(Amazon 리소스 이름(ARN))을 더 구체적인 ARN 값으로 설정하여 Aurora에서 Amazon S3 버킷의 특정 파일 또는 폴더에만 액세스하도록 허용할 수 있습니다. Amazon S3에 대한 액세스 정책을 정의하는 방법에 대한 자세한 내용은 [Amazon S3 리소스에 대한 액세스 권한 관리](#) 단원을 참조하십시오.

10. (선택 사항) 버킷에 대해 Add ARN(ARN 추가)를 선택하여 다른 Amazon S3 버킷을 정책에 추가하고 버킷에 대해 이전 단계를 반복합니다.

    > **Note**
    >
    > 이 단계를 반복하여 Aurora에서 액세스할 각 Amazon S3 버킷의 정책에 해당 버킷 권한 설명문을 추가할 수 있습니다. 또는 Amazon S3의 모든 버킷 및 객체에 대한 액세스 권한을 부여할 수도 있습니다.

11. 정책 검토를 선택합니다.

12. 이름에서 IAM 정책의 이름을 입력합니다(예: `AllowAuroraToExampleBucket`). IAM 역할을 만들어 Aurora DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
13. [Create policy]를 선택합니다.
14. Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성의 단계를 수행합니다.

AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora에서 사용자를 대신하여 AWS Lambda 함수를 호출하는 데 필요한 최소 권한을 제공하는 IAM 정책을 만들 수 있습니다.

아래 정책은 Aurora에서 사용자를 대신하여 AWS Lambda 함수를 호출하는 데 필요한 권한을 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleFunction",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource":
 "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"
        }
    ]
}
```

다음 단계를 사용하여 Aurora에서 사용자를 대신하여 AWS Lambda 함수를 호출하는 데 필요한 최소 권한을 제공하는 IAM 정책을 생성할 수 있습니다. Aurora에서 모든 AWS Lambda 함수를 호출하도록 허용하려면 이러한 단계를 건너뛰고 정책을 직접 만드는 대신에 사전 정의된 `AWSLambdaRole` 정책을 사용할 수 있습니다.

AWS Lambda 함수에 대한 호출 권한을 부여하는 IAM 정책을 생성하려면

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.

3. [Create policy]를 선택합니다.
4. [Visual editor] 탭에서 [Choose a service]를 선택한 다음 [Lambda]를 선택합니다.
5. 작업에서 Expand all(모두 확장)을 선택한 후 IAM 정책에 필요한 AWS Lambda 권한을 선택합니다.

   `InvokeFunction`이 선택되었는지 확인합니다. 이는 Amazon Aurora를 활성화하여 AWS Lambda 함수를 호출하는 데 필요한 최소 권한입니다.
6. [Resources]를 선택하고 함수에 대해 [Add ARN]을 선택합니다.
7. [Add ARN(s)] 대화 상자에서 리소스에 대한 세부 정보를 제공합니다.

   액세스를 허용할 Lambda 함수를 지정하십시오. 예를 들어 Aurora이 `example_function`라는 Lambda 함수에 액세스하도록 허용하려고 하는 경우 ARN 값을 `arn:aws:lambda:::function:example_function`으로 설정하십시오.

   AWS Lambda에 대한 액세스 정책을 정의하는 방법에 대한 자세한 내용은 AWS Lambda의 인증 및 액세스 제어를 참조하십시오.
8. 선택적으로 권한 추가를 선택하여 다른 AWS Lambda 함수를 정책에 추가하고 함수에 대해 이전 단계를 반복하십시오.

   > **Note**
   > 이 단계를 반복하여 Aurora에서 액세스할 각 AWS Lambda 함수의 정책에 해당 함수 권한 설명문을 추가할 수 있습니다.
9. 정책 검토를 선택합니다.
10. [Name]을 IAM 정책의 이름으로 설정합니다(예: `AllowAuroraToExampleFunction`). IAM 역할을 만들어 Aurora DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
11. [Create policy]를 선택합니다.
12. Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성의 단계를 수행합니다.

CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora은 CloudWatch Logs에 액세스하여 Aurora DB 클러스터에서 감사 로그 데이터를 내보낼 수 있습니다. 하지만 Aurora에서 CloudWatch Logs에 액세스하도록 허용하는 로그 그룹 및 로그 스트림 권한을 제공하는 IAM 정책을 먼저 생성해야 합니다.

다음 정책은 사용자 대신 Aurora Amazon CloudWatch Logs에 액세스하는 데 필요한 권한과 로그 그룹을 생성하고 데이터를 내보내는 데 필요한 최소 권한을 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid":
 "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

정책의 ARN을 수정하여 특정 AWS 리전 및 계정에 대한 액세스를 제한할 수 있습니다.

다음 단계를 사용하여 사용자 대신 Aurora에서 CloudWatch Logs에 액세스하는 데 필요한 최소 권한을 제공하는 IAM 정책을 생성할 수 있습니다. Aurora에서 CloudWatch Logs에 대한 모든 액세스를 허용하려면 이러한 단계를 건너뛰고 정책을 직접 생성하는 대신 `CloudWatchLogsFullAccess` 미리 정의된 IAM 정책을 사용할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 [CloudWatch Logs에 대한 자격 증명 기반 정책(IAM 정책) 사용]을 참조하십시오.

CloudWatch Logs 리소스에 대한 액세스 권한을 부여하는 IAM 정책 생성 방법

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. [Create policy]를 선택합니다.
4. [Visual editor] 탭에서 [Choose a service]를 선택한 다음 [CloudWatch Logs]를 선택합니다.
5. 작업에서 Expand all(모두 확장)(오른쪽에 있음)을 선택한 후 IAM 정책에 필요한 Amazon CloudWatch Logs 권한을 선택합니다.

   다음 권한이 선택되었는지 확인합니다.

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. [Resources]를 선택하고 log-group에 대해 [Add ARN]을 선택합니다.
7. [Add ARN(s)] 대화 상자에서 다음 값을 입력합니다.
   - 리전 - AWS 리전 또는 *
   - 계정 – 계정 번호 또는 *
   - 로그 그룹 이름 – `/aws/rds/*`
8. [Add ARN(s)] 대화 상자에서 [Add]를 선택합니다.
9. [log-stream]에 대해 [Add ARN]을 선택합니다.
10. [Add ARN(s)] 대화 상자에서 다음 값을 입력합니다.
    - 리전 - AWS 리전 또는 *
    - 계정 – 계정 번호 또는 *
    - 로그 그룹 이름 – `/aws/rds/*`
    - 로그 Stream Name – *
11. [Add ARN(s)] 대화 상자에서 [Add]를 선택합니다.
12. 정책 검토를 선택합니다.

13. [Name]을 IAM 정책의 이름으로 설정합니다(예: `AmazonRDSCloudWatchLogs`). IAM 역할을 만들어 Aurora DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
14. [Create policy]를 선택합니다.
15. Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성의 단계를 수행합니다.

AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora는 데이터베이스 백업 암호화에 사용되는 AWS KMS keys에 액세스할 수 있습니다. 하지만 Aurora에서 KMS 키에 액세스하도록 허용하는 IAM 정책을 먼저 생성해야 합니다.

아래 정책은 이 사용자를 대신하여 Aurora에서 KMS 키에 액세스하는 데 필요한 권한을 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"
        }
    ]
}
```

다음 단계를 사용하여 사용자 대신 Aurora에서 KMS 키에 액세스하는 데 필요한 최소 권한을 제공하는 IAM 정책을 생성할 수 있습니다.

KMS 키에 대한 액세스 권한을 부여하는 IAM 정책 생성

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. [Create policy]를 선택합니다.

4. Visual editor(시각적 편집기) 탭에서 Choose a service(서비스 선택)를 선택한 다음 KMS를 선택합니다.
5. 작업에서 Write(쓰기)를 선택한 후 Decrypt(암호 해독)를 선택합니다.
6. [Resources]를 선택하고 [Add ARN]를 선택합니다.
7. [Add ARN(s)] 대화 상자에서 다음 값을 입력합니다.
   - 리전 - `us-west-2`와 같은 AWS 리전을 입력합니다.
   - 계정 – 사용자 계정 번호를 입력하십시오.
   - 로그 스트림 이름 – KMS 키 식별자를 입력합니다.
8. [Add ARN(s)] 대화 상자에서 [Add]를 선택합니다.
9. 정책 검토를 선택합니다.
10. [Name]을 IAM 정책의 이름으로 설정합니다(예: `AmazonRDSKMSKey`). IAM 역할을 만들어 Aurora DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
11. [Create policy]를 선택합니다.
12. Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성의 단계를 수행합니다.

Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성

Aurora에서 AWS 리소스에 액세스하도록 허용하는 IAM 정책을 생성한 다음에는 IAM 역할을 생성하여 IAM 정책을 새 IAM 역할에 연결해야 합니다.

Amazon RDS 클러스터에서 사용자를 대신하여 다른 AWS 서비스와 통신하도록 허용하는 IAM 역할을 만들려면 다음 단계를 수행합니다.

Amazon RDS에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할을 만들려면

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. [AWS service]에서 [RDS]를 선택합니다.
5. Select your use case(사용 사례 선택)에서 RDS – Add Role to Database(데이터베이스에 역할 추가)를 선택하십시오.
6. Next(다음)를 선택합니다.

7. 권한 정책(Permissions policies) 페이지에서 검색(Search) 필드에 정책 이름을 입력합니다.
8. 리스트에 표시되면, 다음 섹션 중 하나의 지침을 사용하여 앞서 정의한 정책을 선택합니다.

   - Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성
   - AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성
   - CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성
   - AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성
9. Next(다음)를 선택합니다.
10. 역할 이름에 IAM 역할의 이름을 입력합니다(예: `RDSLoadFromS3`). 설명 값(선택 사항)을 추가할 수도 있습니다.
11. Create Role(역할 생성)을 선택합니다.
12. IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결의 단계를 수행합니다.

IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결

Amazon Aurora DB 클러스터의 데이터베이스 사용자가 다른 AWS 서비스에 대한 액세스를 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 IAM 역할을 해당 DB 클러스터와 연결합니다. 서비스를 직접 연결하여 AWS가 새 IAM 역할을 생성하도록 할 수도 있습니다.

> **Note**
>
> IAM 역할을 Aurora Serverless v1 DB 클러스터와 연결할 수 없습니다. 자세한 내용은 Amazon Aurora Serverless v1 사용 단원을 참조하십시오.
> IAM 역할을 Aurora Serverless v2 DB 클러스터와 연결할 수 있습니다.

IAM 역할을 DB 클러스터와 연결하려면 다음 두 가지 작업을 수행합니다.

1. RDS 콘솔, add-role-to-db-cluster AWS CLI 명령 또는 AddRoleToDBCluster RDS API 작업을 사용하여 DB 클러스터의 연결된 역할 목록에 역할을 추가하십시오.

   각 Aurora DB 클러스터에 대해 최대 다섯 개의 IAM 역할을 추가할 수 있습니다.
2. 관련 AWS 서비스에 대한 클러스터 수준 파라미터를 연결된 IAM 역할의 ARN으로 설정합니다.

   다음 표에는 다른 AWS 서비스에 액세스하는 데 사용되는 IAM 역할의 클러스터 수준 파라미터 이름이 나와 있습니다.

클러스터 수준 파라미터	설명
aws_default_lambda_role	DB 클러스터에서 Lambda 함수를 호출하는 데 사용됩니다.
aws_default_logs_role	이 파라미터는 DB 클러스터의 로그 데이터를 Amazon CloudWatch Logs Logs로 내보낼 때 더 이상 필요하지 않습니다. 이제 Aurora MySQL은 필요한 권한에 서비스 연결 역할을 사용합니다. 서비스 연결 역할에 대한 자세한 내용은 Amazon Aurora에 서비스 연결 역할 사용 단원을 참조하십시오.
aws_default_s3_role	DB 클러스터에서 `LOAD DATA FROM S3`, `LOAD XML FROM S3` 또는 `SELECT INTO OUTFILE S3` 문을 호출하는 데 사용됩니다. Aurora MySQL 버전 2에서 이 파라미터에 지정된 IAM 역할은 적절한 문에 대해 `aurora_load_from_s3_role` 또는 `aurora_select_into_s3_role` 에 IAM 역할이 지정되지 않은 경우에 사용됩니다. Aurora MySQL 버전 3에서는 이 파라미터에 지정된 IAM 역할이 항상 사용됩니다.
aurora_load_from_s3_role	DB 클러스터에서 `LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 문을 호출하는 데 사용됨. 이 파라미터에 대해 지정된 IAM 역할이 없는 경우 `aws_default_s3_role` 에 지정된 IAM 역할이 사용됩니다. Aurora MySQL 버전 3에서는 이 파라미터를 사용할 수 없습니다.

클러스터 수준 파라미터	설명
aurora_select_into_s3_role	DB 클러스터에서 SELECT INTO OUTFILE S3 문을 호출하는 데 사용됨. 이 파라미터에 대해 지정된 IAM 역할이 없는 경우 aws_default_s3_role 에 지정된 IAM 역할이 사용됩니다. Aurora MySQL 버전 3에서는 이 파라미터를 사용할 수 없습니다.

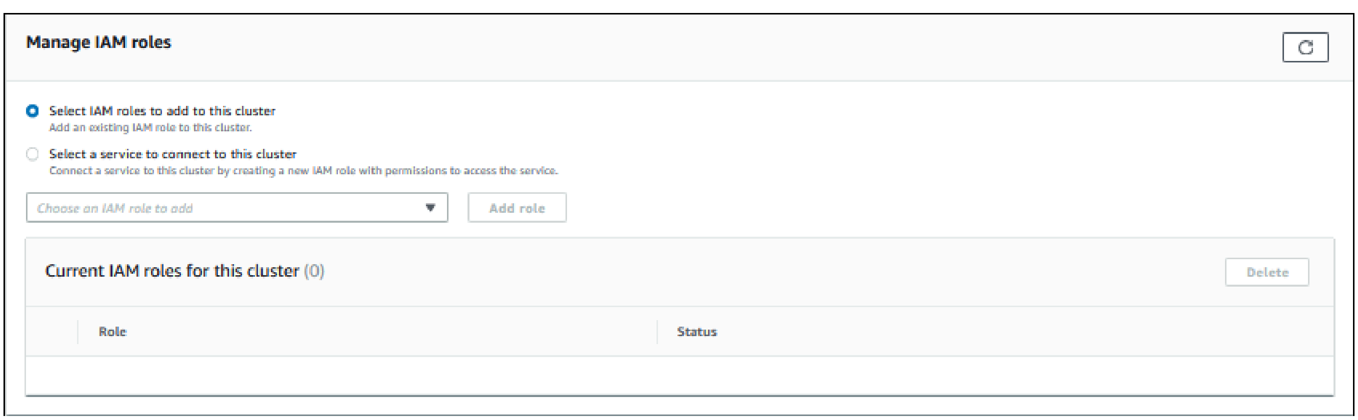
Amazon RDS 클러스터에서 사용자 대신 다른 AWS 서비스와 통신하도록 허용하는 IAM 역할을 연결하려면 다음 단계를 수행하세요.

콘솔

콘솔을 사용하여 IAM 역할을 Aurora DB 클러스터와 연결하려면

1. https://console.aws.amazon.com/rds/에서 RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. IAM 역할과 연결할 Aurora DB 클러스터의 이름을 선택하여 세부 정보를 표시합니다.
4. 연결 및 보안 탭의 IAM 역할 관리 섹션에서 다음 중 하나를 수행합니다.

   - 이 클러스터에 추가할 IAM 역할 선택(기본값)
   - 이 클러스터에 연결할 서비스 선택



5. 기존 IAM 역할을 사용하려면 메뉴에서 해당 역할을 선택한 다음 역할 추가를 선택합니다.

   역할 추가가 성공하면 상태가 `Pending` 다음에 `Available`로 표시됩니다.

6. 서비스를 직접 연결하려면 다음을 수행합니다.

   a. 이 클러스터에 연결할 서비스 선택을 선택합니다.

   b. 메뉴에서 서비스를 선택하고 서비스 연결을 선택합니다.

   c. *### ##*에 클러스터 연결에서 서비스에 연결하는 데 사용할 Amazon 리소스 이름(ARN)을 입력한 다음 서비스 연결을 선택합니다.

   AWS가 서비스 연결을 위한 새 IAM 역할을 생성합니다. 상태는 `Pending` 다음 `Available`로 표시됩니다.

7. (선택 사항) IAM 역할과 DB 클러스터의 연결을 중지하고 관련 권한을 제거하려면 해당 역할을 선택한 다음 삭제를 선택합니다.

연결된 IAM 역할에 클러스터 수준 파라미터를 설정하는 방법

1. RDS 콘솔의 탐색 창에서 [Parameter groups]를 선택합니다.

2. 이미 사용자 지정 DB 파라미터 그룹을 사용 중인 경우, 새 DB 클러스터 파라미터 그룹을 만드는 대신에 해당 그룹을 선택할 수 있습니다. 기본 DB 클러스터 파라미터 그룹을 사용 중인 경우, 다음 단계의 설명에 따라 새 DB 클러스터 파라미터 그룹을 만듭니다.

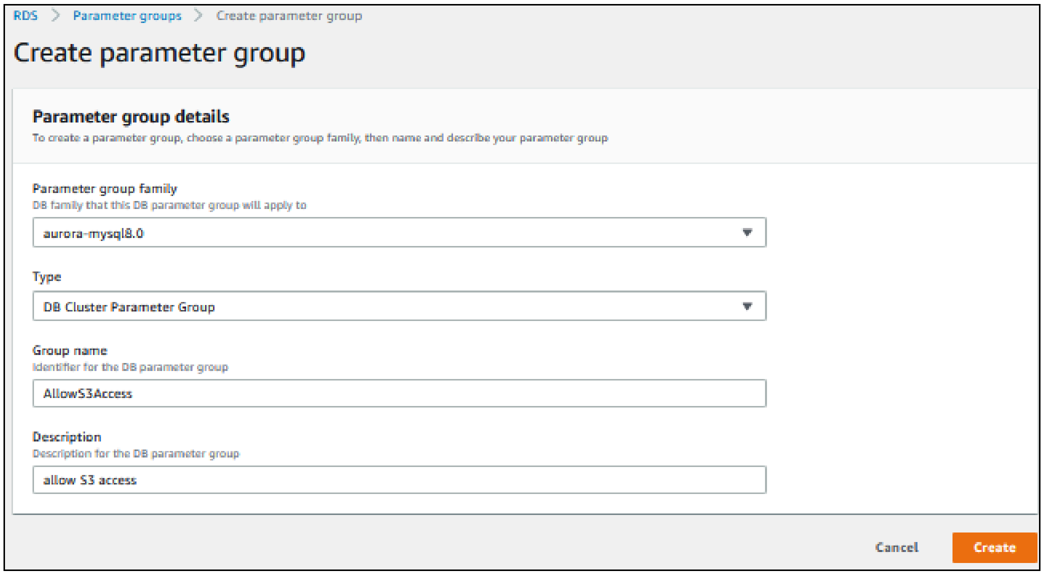
   a. [Create parameter group]을 선택합니다.

   b. 파라미터 그룹 패밀리의 경우 Aurora MySQL 8.0 호환 DB 클러스터에는 `aurora-mysql8.0`을 선택하고, Aurora MySQL 5.7 호환 DB 클러스터에는 `aurora-mysql5.7`을 선택합니다.

   c. [Type]에서 [DB Cluster Parameter Group]을 선택합니다.

   d. [Group name]에 새 DB 클러스터 파라미터 그룹의 이름을 입력합니다.

   e. [Description]에 새 DB 클러스터 파라미터 그룹에 대한 설명을 입력합니다.



   f. 생성(Create)을 선택합니다.

3. 파라미터 그룹 페이지에서 DB 클러스터 파라미터 그룹을 선택하고 Parameter group actions(파라미터 그룹 작업)에서 편집을 선택합니다.

4. 적절한 클러스터 수준 파라미터를 관련 IAM 역할 ARN 값으로 설정합니다.

   예를 들어, `aws_default_s3_role` 파라미터는 `arn:aws:iam::123456789012:role/AllowS3Access`로 설정할 수 있습니다.

5. Save changes(변경 사항 저장)를 선택합니다.

6. DB 클러스터의에서 DB 클러스터 파라미터 그룹을 변경하려면 다음 단계를 완료하십시오.

   a. 데이터베이스와 Aurora DB 클러스터를 차례대로 선택합니다.

   b. 수정을 선택합니다.

   c. 데이터베이스 옵션으로 스크롤하여 DB 클러스터 파라미터 그룹을 DB 클러스터 파라미터 그룹으로 설정합니다.

   d. [Continue]를 선택합니다.

   e. 변경 사항을 확인한 다음 [Apply immediately]를 선택합니다.

   f. Modify cluster(클러스터 수정)를 선택합니다.

   g. 데이터베이스를 선택한 다음 DB 클러스터에서 사용할 기본 인스턴스를 선택합니다.

   h. 작업에서 재부팅을 선택합니다.

      인스턴스가 재부팅되면 IAM 역할이 DB 클러스터와 연결됩니다.

클러스터 파라미터 그룹에 대한 자세한 내용은 Aurora MySQL 구성 파라미터 단원을 참조하십시오.

CLI

AWS CLI를 사용하여 IAM 역할을 DB 클러스터와 연결하려면

1. 다음과 같이 `add-role-to-db-cluster`에서 AWS CLI 명령을 호출하여 DB 클러스터에 IAM 역할의 ARN을 추가하십시오.

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. 기본 DB 클러스터 파라미터 그룹을 사용 중인 경우, 새 DB 클러스터 파라미터 그룹을 만듭니다. 이미 사용자 지정 DB 파라미터 그룹을 사용 중인 경우, 새 DB 클러스터 파라미터 그룹을 만드는 대신에 해당 그룹을 사용할 수 있습니다.

   새 DB 클러스터 파라미터 그룹을 만들려면 다음과 같이 `create-db-cluster-parameter-group`에서 AWS CLI 명령을 호출하십시오.

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
     --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

   Aurora MySQL 5.7 호환 DB 클러스터의 경우 `aurora-mysql5.7`에 `--db-parameter-group-family`을 지정하십시오. Aurora MySQL 8.0 호환 DB 클러스터의 경우 `--db-parameter-group-family`에 `aurora-mysql8.0`을 지정하세요.

3. 다음과 같이 DB 클러스터 파라미터 그룹에서 적절한 클러스터 수준 파라미터 및 관련 IAM 역할 ARN 값을 설정합니다.

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
```

```
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. 다음과 같이 새 DB 클러스터 파라미터 그룹을 사용하도록 DB 클러스터를 수정한 다음 클러스터를 재부팅합니다.

   ```
   PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
   parameter-group-name AllowAWSAccess
   PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
   ```

   인스턴스가 재부팅되면 IAM 역할이 DB 클러스터와 연결됩니다.

   클러스터 파라미터 그룹에 대한 자세한 내용은 Aurora MySQL 구성 파라미터 단원을 참조하십시오.

Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화

Amazon Aurora와 함께 다른 특정 AWS 서비스를 사용하려면 Aurora DB 클러스터의 네트워크 구성에서 이러한 서비스 엔드포인트로의 아웃바운드 연결을 허용해야 합니다. 다음 작업을 수행하려면 이 네트워크 구성이 필요합니다.

- AWS Lambda 함수 호출. 이 기능에 대한 자세한 내용은 Aurora MySQL 네이티브 함수로 Lambda 함수 호출을 참조하십시오.
- Amazon S3에서 파일 액세스. 이 기능에 대한 자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 및 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장을 참조하십시오.
- AWS KMS 엔드포인트 액세스. Aurora MySQL과 함께 데이터베이스 활동 스트림을 사용하려면 AWS KMS 액세스 권한이 필요합니다. 이 기능에 대한 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링을 참조하십시오.
- SageMaker AI 엔드포인트에 액세스 Aurora MySQL과 함께 SageMaker AI 기계 학습을 사용하려면 SageMaker AI 액세스가 필요합니다. 이 기능에 대한 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용을 참조하십시오.

서비스 엔드포인트에 연결할 수 없다면 Aurora은 다음과 같은 오류 메시지를 반환합니다.

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

Aurora MySQL을 사용하는 데이터베이스 활동 스트림의 경우 DB 클러스터에서 AWS KMS 엔드포인트에 액세스할 수 없으면 활동 스트림의 작동이 중지됩니다. Aurora에서는 RDS 이벤트를 사용하여 이 문제에 대해 알립니다.

해당 AWS 서비스를 사용하는 동안 이러한 메시지가 표시되면 Aurora DB 클러스터가 퍼블릭인지 프라이빗인지 확인하세요. Aurora DB 클러스터가 프라이빗이라면, 연결을 허용하도록 구성해야 합니다.

Aurora DB 클러스터를 퍼블릭으로 설정하려면, 퍼블릭 액세스 가능으로 표시해야 합니다. 이러한 경우 AWS Management Console의 DB 클러스터 세부 정보를 살펴보면 퍼블릭 액세스 가능이 예입니다. 또한 DB 클러스터는 Amazon VPC의 퍼블릭 서브넷에 있어야 합니다. 퍼블릭 액세스 DB 인스턴스에 대한 자세한 내용은 VPC에서 DB 클러스터를 사용한 작업 단원을 참조하십시오. 퍼블릭 Amazon VPC 서브넷에 대한 자세한 내용은 VPC 및 서브넷을 참조하십시오.

Aurora DB 클러스터가 퍼블릭 액세스 상태가 아니며 VPC 퍼블릭 서브넷에 있다면, 프라이빗 상태입니다. 프라이빗 DB 클러스터가 있고 이 네트워크 구성이 필요한 기능 중 하나를 사용하려고 할 수 있습니다. 그런 경우 NAT(Network Address Translation)를 통해 인터넷 주소에 연결할 수 있도록 클러스터를 구성합니다. Amazon S3, Amazon SageMaker AI 및 AWS Lambda의 대안으로 DB 클러스터의 라우팅 테이블과 관련된 다른 서비스에 대한 VPC 엔드포인트를 갖도록 VPC를 구성하면 됩니다. VPC에서 DB 클러스터를 사용한 작업 섹션을 참조하세요. VPC에서 NAT를 구성하는 방법에 대한 자세한 내용은 NAT 게이트웨이를 참조하십시오. VPC 엔드포인트 구성에 대한 자세한 내용은 VPC 엔드포인트를 참조하십시오. 또한 S3 게이트웨이 엔드포인트를 생성하여 S3 버킷에 액세스할 수 있습니다. 자세한 내용은 Amazon S3용 게이트웨이 엔드포인트를 참조하세요.

또한 VPC 보안 그룹의 아웃바운드 규칙에서 네트워크 액세스 제어 목록(ACL)에 대한 휘발성 포트를 열어야 할 수도 있습니다. 네트워크 ACL의 휘발성 포트에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 휘발성 포트를 참조하세요.

관련 주제

- Aurora를 다른 AWS 서비스와 통합
- Amazon Aurora DB 클러스터 관리

Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드

`LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 문을 사용하여 Amazon S3 버킷에 저장된 파일에서 데이터를 로드할 수 있습니다. Aurora MySQL에서는 먼저 파일을 로컬 디스크에 저장한 다음 데이터베이스로 가져옵니다. 데이터베이스로 가져오기가 완료되면 로컬 파일은 삭제됩니다.

> **Note**
>
> Aurora Serverless v1에서는 데이터를 텍스트 테이블로 로드할 수 없습니다. Aurora Serverless v2에서는 지원됩니다.

목차

Aurora에 Amazon S3에 대한 액세스 권한 부여

Amazon S3 버킷에서 데이터를 로드하기 전에 Aurora MySQL DB 클러스터에 Amazon S3에 액세스할 권한을 부여해야 합니다.

Aurora MySQL에 Amazon S3에 대한 액세스 권한을 부여하려면

1. Aurora MySQL DB 클러스터에서 Amazon S3에 액세스하도록 허용하는 버킷 및 객체 권한을 제공하는 AWS Identity and Access Management (IAM) 정책을 생성하세요. 지침은 Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 단원을 참조하십시오.

   > **Note**
   >
   > Aurora MySQL 버전 3.05 이상에서는 고객 관리형 AWS KMS keys를 사용하여 암호화된 객체를 로드할 수 있습니다. 이렇게 하려면 IAM 정책에 `kms:Decrypt` 권한을 포함하세요. 자세한 내용은 AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 단원을 참조하십시오.
   > AWS 관리형 키 또는 Amazon S3 관리형 키(SSE-S3)를 사용하여 암호화된 객체를 로드하는 경우 이 권한이 필요하지 않습니다.

2. IAM 역할을 생성하고 Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 섹션을 참조하세요.

3. DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용 중인지 확인하십시오.

   사용자 지정 DB 클러스터 파라미터 그룹 생성에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 단원을 참조하십시오.

4. Aurora MySQL 버전 2에서는 `aurora_load_from_s3_role`이나 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정합니다. `aurora_load_from_s3_role`에 대해 지정된 IAM 역할이 없는 경우, Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.

   Aurora MySQL 버전 3에서는 `aws_default_s3_role`을 사용하면 됩니다.

   클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 이 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 이 파라미터를 설정합니다. Aurora 글로벌 데이터베이스의 기본 클러스터만 데이터를 로드할 수 있더라도 다른 클러스터가 장애 조치 메커니즘에 의해 승격되어 기본 클러스터가 될 수 있습니다.

   DB 클러스터 파라미터에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 단원을 참조하십시오.

5. Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 해

당 DB 클러스터와 연결하십시오. Aurora 글로벌 데이터베이스의 경우 글로벌 데이터베이스의 각 Aurora 클러스터와 해당 역할을 연결합니다. IAM 역할과 DB 클러스터 연결에 대한 자세한 내용은 IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 단원을 참조하십시오.

6. Amazon S3으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 단원을 참조하십시오.

   DB 클러스터가 퍼블릭 액세스 상태가 아니며 VPC 퍼블릭 서브넷에 있다면, 프라이빗 상태입니다. S3 게이트웨이 엔드포인트를 생성하여 S3 버킷에 액세스할 수 있습니다. 자세한 내용은 Amazon S3용 게이트웨이 엔드포인트를 참조하세요.

   Aurora 글로벌 데이터베이스의 경우 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 발신 연결을 활성화합니다.

Amazon Aurora MySQL에서 데이터 로드 권한 부여

`LOAD DATA FROM S3` 또는 `LOAD XML FROM S3` 문을 실행하는 데이터베이스 사용자에게는 둘 중 한 명령문을 실행할 수 있도록 특별한 역할이나 권한이 부여되어야 합니다. Aurora MySQL 버전 3에서는 `AWS_LOAD_S3_ACCESS` 역할을 부여합니다. Aurora MySQL 버전 2에서는 `LOAD FROM S3` 권한을 부여합니다. DB 클러스터의 관리자 사용자에게는 기본적으로 적절한 역할이나 권한이 부여됩니다. 다음 명령문 중 하나를 사용하여 다른 사용자에게 권한을 부여할 수 있습니다.

Aurora MySQL 버전 3에 대해 다음 명령문을 사용합니다.

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Aurora MySQL 버전 3에서 역할을 사용하는 경우 `SET ROLE role_name` 또는 `SET ROLE ALL` 문을 사용하여 해당 역할을 활성화할 수도 있습니다. MySQL 8.0 역할 시스템에 익숙하지 않을 경우 역할 기반 권한 모델에서 자세한 내용을 확인할 수 있습니다. 자세한 내용은 MySQL 참조 설명서의 역할 사용을 참조하세요.
> 이는 현재 활성 세션에만 적용됩니다. 다시 연결할 때 권한을 부여하려면 `SET ROLE` 문을 다시 실행해야 합니다. 자세한 내용은 MySQL 참조 매뉴얼의 SET ROLE 문을 참조하세요.
> `activate_all_roles_on_login` DB 클러스터 파라미터로 사용자가 DB 인스턴스에 연결할 때 모든 역할을 자동으로 활성화할 수 있습니다. 이 파라미터를 설정하면 역할을 활성화하는 데 `SET ROLE` 문을 명시적으로 직접 호출할 필요가 없습니다. 자세한 내용은 MySQL Reference Manual(MySQL 참조 매뉴얼)의 activate_all_roles_on_login을 참조하세요.

하지만 다른 사용자가 저장 프로시저를 직접 호출한 경우에는 저장 프로시저를 시작할 때 명시적으로 SET ROLE ALL을 직접 호출하여 역할을 활성화해야 합니다.

Aurora MySQL 버전 2에 대해 다음 문을 사용합니다.

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

AWS_LOAD_S3_ACCESS 역할 및 LOAD FROM S3 권한은 Amazon Aurora에만 적용되며, 외부 MySQL 데이터베이스 또는 RDS for MySQL DB 인스턴스에서는 사용할 수 없습니다. 복제 소스인 Aurora DB 클러스터와 복제 클라이언트인 MySQL 데이터베이스 간에 복제를 설정한 경우, 역할 또는 권한에 대한 GRANT 문으로 인해 오류가 생겨 복제가 중단됩니다. 오류를 건너뛰고 복제를 계속 진행하셔도 됩니다. RDS for MySQL 인스턴스에서 오류를 건너뛰려면 mysql_rds_skip_repl_error 프로시저를 사용하세요. 외부 MySQL 데이터베이스에서 오류를 건너뛰려면 slave_skip_errors 시스템 변수(Aurora MySQL 버전 2) 또는 replica_skip_errors 시스템 변수(Aurora MySQL 버전 3)를 사용하세요.

Note

데이터베이스 사용자는 데이터를 로드하는 데이터베이스에 대해 INSERT 권한을 보유하고 있어야 합니다.

Amazon S3 버킷으로의 경로(URI) 지정

Amazon S3 버킷에 저장된 파일로의 경로(URI)를 지정하는 구문은 다음과 같습니다.

```
s3-region://amzn-s3-demo-bucket/file-name-or-prefix
```

경로에는 다음 값이 포함됩니다.

- region(선택 사항) – 데이터를 로드할 Amazon S3 버킷이 포함된 AWS 리전입니다. 이 값은 선택 사항입니다. region 값을 지정하지 않으면 Aurora은 DB 클러스터와 동일한 리전에 있는 Amazon S3에서 파일을 로드합니다.
- bucket-name – 로드할 데이터가 포함된 Amazon S3 버킷의 이름입니다. 가상 폴더 경로를 식별하는 객체 접두사가 지원됩니다.
- file-name-or-prefix – Amazon S3 텍스트 파일 또는 XML 파일의 이름, 또는 로드할 하나 이상의 테스트 또는 XML 파일을 식별하는 접두사입니다. 로드할 하나 이상의 텍스트 파일을 식별하는 매니페스트 파일을 지정할 수도 있습니다. Amazon S3에서 텍스트 파일을 로드하기 위해 매니페스트

파일을 사용하는 방법에 대한 자세한 내용은 [매니페스트 파일을 이용해 로드할 데이터 파일 지정](#) 단원을 참조하십시오.

S3 버킷에 있는 파일의 URI를 복사하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 탐색 창에서 버킷을 선택한 다음, 복사하려는 URI가 있는 버킷을 선택합니다.
3. S3에서 로드하려는 접두사 또는 파일을 선택합니다.
4. S3 URI 복사를 선택합니다.

S3에서 데이터 로드

`LOAD DATA FROM S3` 문을 사용하여 쉼표로 구분되는 텍스트 데이터와 같이 MySQL LOAD DATA INFILE 문에서 지원하는 모든 텍스트 파일 형식의 데이터를 로드할 수 있습니다. 압축 파일은 지원되지 않습니다.

> **Note**
> Aurora MySQL DB 클러스터에서 S3에 대한 아웃바운드 연결을 허용하는지 확인하세요. 자세한 내용은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 단원을 참조하십시오.

구문

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
```

```
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

> **Note**
>
> Aurora MySQL 버전 3.05 이상에서는 키워드 `FROM`이 선택 사항입니다.

파라미터

`LOAD DATA FROM S3` 문은 다음과 같은 필수 파라미터와 선택 사항인 파라미터를 사용합니다. MySQL 설명서의 LOAD DATA 문에서 이러한 파라미터에 대한 자세한 내용을 볼 수 있습니다.

FILE | PREFIX | MANIFEST

단일 파일에서 데이터를 로드할지, 지정된 접두사와 일치하는 모든 파일에서 데이터를 로드할지, 지정된 메니페스트의 모든 파일에서 데이터를 로드할지를 식별합니다. `FILE`은 기본값입니다.

S3-URI

로드할 텍스트나 메니페스트 파일의 URI를 지정하거나 사용할 Amazon S3 접두사를 지정합니다. Amazon S3 버킷으로의 경로(URI) 지정에서 설명하는 구문을 사용하여 URI를 지정합니다.

REPLACE | IGNORE

입력 행이 데이터베이스 테이블의 기존 행과 고유 키 값이 동일한 경우 수행할 작업을 결정합니다.

- 입력 행이 테이블의 기존 행을 대체하도록 하려면 `REPLACE`를 지정합니다.
- 입력 행을 무시하려면 `IGNORE`를 지정합니다.

INTO TABLE

입력 행을 로드할 데이터베이스 테이블의 이름을 식별합니다.

PARTITION

모든 입력 행을 쉼표로 구분된 지정된 파티션 이름 목록으로 식별되는 파티션으로 삽입해야 합니다. 입력 행을 지정된 파티션에 삽입할 수 없는 경우 문이 실패하며 오류가 반환됩니다.

CHARACTER SET

입력 파일의 데이터 문자 집합을 식별합니다.

FIELDS | COLUMNS

입력 파일의 필드 또는 열을 구분하는 방법을 식별합니다. 필드는 기본적으로 탭으로 구분됩니다.

LINES

입력 파일의 줄을 구분하는 방법을 식별합니다. 줄은 기본적으로 줄 바꿈 문자('\n')로 구분됩니다.

IGNORE ***number*** LINES | ROWS

입력 파일의 시작 부분에서 특정 줄 또는 행 수를 무시하도록 지정합니다. 예를 들어, `IGNORE 1 LINES`를 사용하여 열 이름이 포함된 첫 헤더 줄을 건너뛰거나 `IGNORE 2 ROWS`를 사용하여 입력 파일의 첫 두 데이터 행을 건너뛸 수 있습니다. `PREFIX`를 사용하는 경우 `IGNORE`는 첫 번째 입력 파일의 시작 부분에서 특정 개수의 행이나 줄을 건너뜁니다.

col_name_or_user_var, ...

로드할 열을 이름을 기준으로 식별하는 사용자 변수 목록 또는 쉼표로 구분된 하나 이상의 열 이름 목록을 지정합니다. 이 목적에 사용되는 사용자 변수의 이름은 @로 시작하는 텍스트 파일의 요소 이름과 일치해야 합니다. 사용자 변수를 사용하여 추후 재사용을 위해 해당 필드 값을 저장할 수 있습니다.

예를 들어, 다음 문은 입력 파일의 첫 번째 열을 `table1`의 첫 번째 열로 로드하고, `table_column2`에 있는 `table1` 열의 값을 100으로 나눈 두 번째 열의 입력 값으로 설정합니다.

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

테이블의 열 값을 입력 파일에 포함되지 않은 값으로 설정하는 쉼표로 구분된 할당 작업 목록을 지정합니다.

예를 들어, 다음 문은 `table1`의 첫 두 열을 입력 파일의 첫 두 열의 값으로 설정한 다음, `column3`에 있는 `table1`의 값을 현재 타임스탬프로 설정합니다.

```
LOAD DATA FROM S3  's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, column2)
```

```
    SET column3 = CURRENT_TIMESTAMP;
```

SET 할당의 오른쪽에서 하위 쿼리를 사용할 수 있습니다. 열에 할당될 값을 반환하는 하위 쿼리의 경우, 스칼라 하위 쿼리만 사용할 수 있습니다. 또한 로드 중인 테이블에서 선택할 때는 하위 쿼리를 사용할 수 없습니다.

Amazon S3 버킷에서 데이터를 로드 중인 경우 `LOAD DATA FROM S3` 문의 `LOCAL` 키워드를 사용할 수 없습니다.

매니페스트 파일을 이용해 로드할 데이터 파일 지정

`LOAD DATA FROM S3` 문과 `MANIFEST` 키워드를 사용하여 DB 클러스터에 있는 테이블에 로드할 텍스트 파일 목록을 표시하는 JSON 형식의 메니페스트 파일을 지정할 수 있습니다.

다음 JSON 스키마는 매니페스트 파일의 형식 및 내용을 설명합니다.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
                },
                "required": [
```

```
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

메니페스트의 각 url에는 접두사뿐 아니라 파일의 버킷 이름과 전체 객체 경로를 포함한 URL을 지정해야 합니다. 메니페스트를 이용하면 다른 버킷, 다른 리전이나 접두사가 다른 파일에서 파일을 불러올 수 있습니다. URL에서 리전이 지정되지 않았다면, 대상 Aurora DB 클러스터의 리전을 사용합니다. 다음은 다른 버킷에서 파일 4개를 로드하는 메니페스트 파일 예제입니다.

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
  ]
}
```

선택 사항인 mandatory 플래그는 파일이 없을 때 LOAD DATA FROM S3의 오류 반환 여부를 결정합니다. mandatory 플래그의 기본값은 false입니다. mandatory 설정 여부와 상관없이, 파일이 발견되지 않으면 LOAD DATA FROM S3은 종료됩니다.

메니페스트 파일은 확장자를 제한하지 않습니다. 다음은 앞 예제에서 사용한 LOAD DATA FROM S3 매니페스트를 이용해 **customer.manifest** 문을 실행하는 예제입니다.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

문이 완료되면 성공적으로 로드한 각 파일이 aurora_s3_load_history에 기록됩니다.

aurora_s3_load_history 테이블을 이용해 로드 파일 확인

LOAD DATA FROM S3 문이 완료될 때마다 aurora_s3_load_history 스키마에 있는 mysql 테이블에 각 로드 파일 항목이 업데이트됩니다.

LOAD DATA FROM S3 문을 실행하면, aurora_s3_load_history 테이블을 쿼리해 로드한 파일을 확인할 수 있습니다. 이 문을 한 번 반복하여 로드한 파일을 확인하려면 WHERE 절을 사용하여 Amazon S3 URI의 기록에서 문에 사용된 매니페스트 파일을 필터링합니다. 이전에 같은 메니페스트 파일을 사용한 적이 있다면, timestamp 필드로 결과를 필터링하십시오.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

다음 표는 aurora_s3_load_history 테이블 필드를 설명합니다.

필드	설명
load_prefix	로드 문에 지정된 URI입니다. 이 URI는 다음에 매핑할 수 있습니다. • LOAD DATA FROM S3 FILE 문에 대한 단일 데이터 파일 • LOAD DATA FROM S3 PREFIX 문에 대한 여러 데이터 파일에 매핑되는 Amazon S3 접두사 • LOAD DATA FROM S3 MANIFEST 문에 대해 로드할 파일 이름이 포함된 단일 매니페스트 파일
file_name	Amazon S3에서 Aurora로 로드한 파일 이름으로, load_prefix 필드에서 확인한 URI를 사용합니다.

필드	설명
version_number	Amazon S3 버킷에 버전 번호가 있는 경우 로드한 file_name 필드로 확인한 파일 버전 번호입니다.
bytes_loaded	로드한 파일의 크기(바이트)입니다.
load_timestamp	LOAD DATA FROM S3 문이 완료된 시점의 타임스탬프입니다.

예시

다음 문은 Aurora DB 클러스터와 동일한 리전에 있는 Amazon S3 버킷에서 데이터를 로드합니다. 이 문은 *amzn-s3-demo-bucket* Amazon S3 버킷에 있는 파일 customerdata.txt에서 쉼표로 구분된 데이터를 읽은 다음 store-schema.customer-table 표로 해당 데이터를 로드합니다.

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

다음 문은 Aurora DB 클러스터와 다른 리전에 있는 Amazon S3 버킷에서 데이터를 로드합니다. 이 문은 us-west-2 리전에 있는 *amzn-s3-demo-bucket* Amazon S3 버킷의 employee-data 객체 접두사와 일치하는 모든 파일에서 쉼표로 구분된 데이터를 읽은 다음 employees 표로 해당 데이터를 로드합니다.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://amzn-s3-demo-bucket/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

다음 문은 q1_sales.json이라는 JSON 메니페스트 파일에서 지정한 파일의 데이터를 sales 테이블로 로드합니다.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://amzn-s3-demo-bucket1/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
```

```
    (MONTH, STORE, GROSS, NET);
```

S3에서 XML 로드

`LOAD XML FROM S3` 문을 사용하여 Amazon S3 버킷에 저장된 XML 파일에서 다음과 같은 세 가지 XML 형식 중 하나로 데이터를 로드할 수 있습니다.

- 열 이름이 `<row>` 요소의 속성입니다. 이 속성 값은 테이블 필드의 콘텐츠를 식별합니다.

  ```
  <row column1="value1" column2="value2" .../>
  ```

- 열 이름이 `<row>` 요소의 하위 요소입니다. 이 하위 요소의 값은 테이블 필드의 콘텐츠를 식별합니다.

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- 열 이름이 name 요소의 `<field>` 요소의 `<row>` 속성에 있습니다. 이 `<field>` 요소의 값은 테이블 필드의 콘텐츠를 식별합니다.

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

구문

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

파라미터

`LOAD XML FROM S3` 문은 다음과 같은 필수 파라미터와 선택 사항인 파라미터를 사용합니다. MySQL 설명서의 [LOAD XML 문]에서 이러한 파라미터에 대한 자세한 내용을 볼 수 있습니다.

FILE | PREFIX

단일 파일에서 데이터를 로드할지, 지정된 접두사와 일치하는 모든 파일에서 데이터를 로드할지 식별합니다. `FILE`은 기본값입니다.

REPLACE | IGNORE

입력 행이 데이터베이스 테이블의 기존 행과 고유 키 값이 동일한 경우 수행할 작업을 결정합니다.

- 입력 행이 테이블의 기존 행을 대체하도록 하려면 `REPLACE`를 지정합니다.
- 입력 행을 무시하려는 경우 `IGNORE`를 지정하십시오. `IGNORE`는 기본값입니다.

INTO TABLE

입력 행을 로드할 데이터베이스 테이블의 이름을 식별합니다.

PARTITION

모든 입력 행을 쉼표로 구분된 지정된 파티션 이름 목록으로 식별되는 파티션으로 삽입해야 합니다. 입력 행을 지정된 파티션에 삽입할 수 없는 경우 문이 실패하며 오류가 반환됩니다.

CHARACTER SET

입력 파일의 데이터 문자 집합을 식별합니다.

ROWS IDENTIFIED BY

입력 파일의 행을 식별하는 요소 이름을 식별합니다. 기본값은 `<row>`입니다.

IGNORE ***number*** LINES | ROWS

입력 파일의 시작 부분에서 특정 줄 또는 행 수를 무시하도록 지정합니다. 예를 들어, `IGNORE 1 LINES`를 사용하여 텍스트 파일의 첫 번째 줄을 건너뛰거나 `IGNORE 2 ROWS`를 사용하여 입력 XML의 첫 두 데이터 행을 건너뛸 수 있습니다.

field_name_or_user_var, ...

로드할 요소를 이름을 기준으로 식별하는 사용자 변수 목록 또는 쉼표로 구분된 하나 이상의 XML 요소 이름 목록을 지정합니다. 이 목적에 사용되는 사용자 변수의 이름은 @로 시작하는 XML 파일의 요소 이름과 일치해야 합니다. 사용자 변수를 사용하여 추후 재사용을 위해 해당 필드 값을 저장할 수 있습니다.

예를 들어, 다음 문은 입력 파일의 첫 번째 열을 `table1`의 첫 번째 열로 로드하고, `table_column2`에 있는 `table1` 열의 값을 100으로 나눈 두 번째 열의 입력 값으로 설정합니다.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

테이블의 열 값을 입력 파일에 포함되지 않은 값으로 설정하는 쉼표로 구분된 할당 작업 목록을 지정합니다.

예를 들어, 다음 문은 `table1`의 첫 두 열을 입력 파일의 첫 두 열의 값으로 설정한 다음, `column3`에 있는 `table1`의 값을 현재 타임스탬프로 설정합니다.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

SET 할당의 오른쪽에서 하위 쿼리를 사용할 수 있습니다. 열에 할당될 값을 반환하는 하위 쿼리의 경우, 스칼라 하위 쿼리만 사용할 수 있습니다. 또한 로드 중인 테이블에서 선택할 때는 하위 쿼리를 사용할 수 없습니다.

Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장

`SELECT INTO OUTFILE S3` 문을 사용하여 Amazon Aurora MySQL DB 클러스터에서 데이터를 쿼리한 후 Amazon S3 버킷에 저장된 텍스트 파일에 저장할 수 있습니다. Aurora MySQL에서는 먼저 파일을 로컬 디스크에 저장한 다음 S3로 내보냅니다. 내보내기가 완료되면 로컬 파일이 삭제됩니다.

Amazon S3 관리형 키(SSE-S3) 또는 AWS KMS key(SSE-KMS: AWS 관리형 키 또는 고객 관리형 키)를 사용하여 Amazon S3 버킷을 암호화할 수 있습니다.

`LOAD DATA FROM S3` 문은 `SELECT INTO OUTFILE S3` 문에 의해 만들어진 파일을 사용하여 데이터를 Aurora DB 클러스터에 로드할 수 있습니다. 자세한 내용은 [Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드](#) 단원을 참조하십시오.

> **Note**
>
> 이 기능은 Aurora Serverless v1 DB 클러스터에는 지원되지 않습니다. Aurora Serverless v2 DB 클러스터에 지원됩니다.
> AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 DB 클러스터 데이터 및 DB 클러스터 스냅샷 데이터를 Amazon S3에 저장할 수 있습니다. 자세한 내용은 Amazon S3로 DB 클러스터 데이터 내보내기 및 Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기 단원을 참조하세요.

목차

Aurora MySQL에 Amazon S3에 대한 액세스 권한 부여

Amazon S3 버킷에 데이터를 저장하기 전에 Aurora MySQL DB 클러스터에 Amazon S3에 액세스할 권한을 부여해야 합니다.

Aurora MySQL에 Amazon S3에 대한 액세스 권한을 부여하려면

1. Aurora MySQL DB 클러스터에서 Amazon S3에 액세스하도록 허용하는 버킷 및 객체 권한을 제공하는 AWS Identity and Access Management (IAM) 정책을 생성하세요. 지침은 Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 단원을 참조하십시오.

   > **Note**
   >
   > Aurora MySQL 버전 3.05 이상에서는 AWS KMS 고객 관리형 키를 사용하여 객체를 암호화할 수 있습니다. 이렇게 하려면 IAM 정책에 `kms:GenerateDataKey` 권한을 포함하세

요. 자세한 내용은 AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 단원을 참조하십시오.
AWS 관리형 키 또는 Amazon S3 관리형 키(SSE-S3)를 사용하여 객체를 암호화하는 경우 이 권한이 필요하지 않습니다.

2. IAM 역할을 생성하고 Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 단원을 참조하십시오.
3. Aurora MySQL 버전 2에서는 `aurora_select_into_s3_role`이나 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정합니다. `aurora_select_into_s3_role`에 대해 지정된 IAM 역할이 없는 경우, Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.

   Aurora MySQL 버전 3에서는 `aws_default_s3_role`을 사용하면 됩니다.

   클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 이 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 이 파라미터를 설정합니다.

   DB 클러스터 파라미터에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 단원을 참조하십시오.
4. Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 해당 DB 클러스터와 연결하십시오.

   Aurora 글로벌 데이터베이스의 경우 글로벌 데이터베이스의 각 Aurora 클러스터와 해당 역할을 연결합니다.

   IAM 역할과 DB 클러스터 연결에 대한 자세한 내용은 IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 단원을 참조하십시오.
5. Amazon S3으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

   Aurora 글로벌 데이터베이스의 경우 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 발신 연결을 활성화합니다.

Aurora MySQL에서 데이터 저장 권한 부여

`SELECT INTO OUTFILE S3` 문을 실행하는 데이터베이스 사용자에게는 특별한 역할이나 권한이 부여되어야 합니다. Aurora MySQL 버전 3에서는 `AWS_SELECT_S3_ACCESS` 역할을 부여합니다. Aurora MySQL 버전 2에서는 `SELECT INTO S3` 권한을 부여합니다. DB 클러스터의 관리자 사용자에게는 기본적으로 적절한 역할이나 권한이 부여됩니다. 다음 명령문 중 하나를 사용하여 다른 사용자에게 권한을 부여할 수 있습니다.

Aurora MySQL 버전 3에 대해 다음 명령문을 사용합니다.

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Aurora MySQL 버전 3에서 역할을 사용하는 경우 `SET ROLE role_name` 또는 `SET ROLE ALL` 문을 사용하여 해당 역할을 활성화할 수도 있습니다. MySQL 8.0 역할 시스템에 익숙하지 않을 경우 역할 기반 권한 모델에서 자세한 내용을 확인할 수 있습니다. 자세한 내용은 MySQL 참조 설명서의 역할 사용을 참조하세요.
> 이는 현재 활성 세션에만 적용됩니다. 다시 연결할 때 권한을 부여하려면 `SET ROLE` 문을 다시 실행해야 합니다. 자세한 내용은 MySQL 참조 매뉴얼의 SET ROLE 문을 참조하세요.
> `activate_all_roles_on_login` DB 클러스터 파라미터로 사용자가 DB 인스턴스에 연결할 때 모든 역할을 자동으로 활성화할 수 있습니다. 이 파라미터를 설정하면 역할을 활성화하는 데 `SET ROLE` 문을 명시적으로 직접 호출할 필요가 없습니다. 자세한 내용은 MySQL Reference Manual(MySQL 참조 매뉴얼)의 activate_all_roles_on_login을 참조하세요.
> 하지만 다른 사용자가 저장 프로시저를 직접 호출한 경우에는 저장 프로시저를 시작할 때 명시적으로 `SET ROLE ALL`을 직접 호출하여 역할을 활성화해야 합니다.

Aurora MySQL 버전 2에 대해 다음 문을 사용합니다.

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_SELECT_S3_ACCESS` 역할 및 `SELECT INTO S3` 권한은 Amazon Aurora MySQL에만 적용되며 MySQL 데이터베이스 또는 RDS for MySQL DB 인스턴스에서는 사용할 수 없습니다. 복제 소스인 Aurora MySQL DB 클러스터와 복제 클라이언트인 MySQL 데이터베이스 간에 복제를 설정한 경우, 역할 또는 권한에 대한 `GRANT` 문으로 인해 오류가 생겨 복제가 중단됩니다. 오류를 건너뛰고 복제를 계속 진행하셔도 됩니다. RDS for MySQL DB 인스턴스에서 오류를 건너뛰려면

mysql_rds_skip_repl_error 프로시저를 사용하십시오. 외부 MySQL 데이터베이스에서 오류를 건너뛰려면 slave_skip_errors 시스템 변수(Aurora MySQL 버전 2) 또는 replica_skip_errors 시스템 변수(Aurora MySQL 버전 3)를 사용하세요.

Amazon S3 버킷의 경로 지정

Amazon S3 버킷에서 데이터 및 매니페스트 파일을 저장할 경로를 지정하는 구문은 다음과 같이 `LOAD DATA FROM S3 PREFIX` 문에 사용된 것과 비슷합니다.

```
s3-region://bucket-name/file-prefix
```

경로에는 다음 값이 포함됩니다.

- `region`(선택 사항) – 데이터를 저장할 Amazon S3 버킷이 포함된 AWS 리전입니다. 이 값은 선택 사항입니다. `region` 값을 지정하지 않으면 Aurora은 DB 클러스터와 동일한 리전에 있는 Amazon S3에 파일을 저장합니다.
- `bucket-name` – 데이터를 저장할 Amazon S3 버킷의 이름입니다. 가상 폴더 경로를 식별하는 객체 접두사가 지원됩니다.
- `file-prefix` – Amazon S3에 저장할 파일을 식별하는 Amazon S3 객체 접두사입니다.

`SELECT INTO OUTFILE S3` 문으로 만들어진 데이터 파일은 다음 경로를 사용합니다. 여기서 *00000*은 0부터 시작하는 5자리 정수입니다.

```
s3-region://bucket-name/file-prefix.part_00000
```

예를 들어 `SELECT INTO OUTFILE S3` 문이 데이터 파일을 저장할 경로로 `s3-us-west-2://bucket/prefix`를 지정하고 3개의 데이터 파일을 만드는 것으로 가정합시다. 지정된 Amazon S3 버킷에는 다음 데이터 파일이 포함됩니다.

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

데이터 파일을 나열할 매니페스트 만들기

`SELECT INTO OUTFILE S3` 문을 `MANIFEST ON` 옵션과 함께 사용하여 문에 의해 만들어진 텍스트 파일을 나열하는 매니페스트 파일(JSON 형식)을 만들 수 있습니다. `LOAD DATA FROM S3` 문은 매니

페스트 파일을 사용하여 데이터 파일을 다시 Aurora MySQL DB 클러스터에 로드할 수 있습니다. 매니페스트 파일을 사용하여 데이터 파일을 Amazon S3에서 Aurora MySQL DB 클러스터로 로드하는 방법에 대한 자세한 내용은 매니페스트 파일을 이용해 로드할 데이터 파일 지정 단원을 참조하십시오.

`SELECT INTO OUTFILE S3` 문으로 만들어진 매니페스트 파일에 포함된 데이터 파일은 만들어진 순서대로 나열됩니다. 예를 들어 `SELECT INTO OUTFILE S3` 문이 데이터 파일을 저장할 경로로 `s3-us-west-2://bucket/prefix`를 지정하고 데이터 파일 3개와 매니페스트 파일을 만드는 것으로 가정합시다. 지정된 Amazon S3 버킷에는 다음 정보를 포함하는 매니페스트 파일 `s3-us-west-2://bucket/prefix.manifest`가 있습니다.

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

`SELECT INTO OUTFILE S3` 문을 사용하여 DB 클러스터에서 데이터를 쿼리한 후 Amazon S3 버킷에 저장된 구분 기호로 구분된 텍스트 파일에 직접 저장할 수 있습니다.

압축 파일은 지원되지 않습니다. 암호화된 파일은 Aurora MySQL 버전 2.09.0부터 지원됩니다.

구문

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
```

```
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

파라미터

`SELECT INTO OUTFILE S3` 문은 Aurora에 고유한 다음과 같은 필수 및 선택 파라미터를 사용합니다.

s3-uri

사용할 Amazon S3 접두사의 URI를 지정합니다. Amazon S3 버킷의 경로 지정에 설명된 구문을 사용하세요.

FORMAT {CSV|TEXT} [HEADER]

필요에 따라 데이터를 CSV 형식으로 저장합니다.

`TEXT` 옵션은 기본값이며 기존 MySQL 내보내기 형식을 생성합니다.

`CSV` 옵션은 쉼표로 구분된 데이터 값을 생성합니다. CSV 형식은 RFC-4180의 사양을 따릅니다. 선택적 키워드인 `HEADER`를 지정하는 경우 출력 파일에는 헤더 행이 한 줄 포함됩니다. 이 헤더 행의

레이블은 SELECT 문의 열 이름에 해당합니다. AWS 기계 학습 서비스에서 사용할 훈련 데이터 모델에 CSV 파일을 사용할 수 있습니다. AWS 기계 학습 서비스에 내보낸 Aurora 데이터를 사용하는 방법에 대한 자세한 내용은 SageMaker AI 모델 훈련을 위해 Amazon S3로 데이터 내보내기(고급) 섹션을 참조하세요.

MANIFEST {ON | OFF}

Amazon S3에서 매니페스트 파일이 만들어졌는지 여부를 나타냅니다. 매니페스트 파일은 LOAD DATA FROM S3 MANIFEST 문으로 Aurora DB 클러스터에 데이터를 로드하는 데 사용할 수 있는 JavaScript Object Notation(JSON) 파일입니다. For more information about LOAD DATA FROM S3 MANIFEST, see Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드.

쿼리에서 MANIFEST ON을 지정할 경우 모든 데이터 파일이 만들어지고 업로드된 후 Amazon S3에서 매니페스트 파일이 만들어집니다. 매니페스트 파일은 다음 경로를 사용하여 만들어집니다.

```
s3-region://bucket-name/file-prefix.manifest
```

매니페스트 파일 내용의 형식에 대한 자세한 내용은 데이터 파일을 나열할 매니페스트 만들기 단원을 참조하십시오.

OVERWRITE {ON | OFF}

지정된 Amazon S3 버킷의 기존 파일을 덮어쓸지 여부를 나타냅니다. OVERWRITE ON을 지정할 경우 s3-uri에 지정된 URI에서 파일 접두사와 일치하는 기존 파일이 덮어쓰입니다. 그렇지 않으면 오류가 발생합니다.

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

서버 측 암호화를 Amazon S3 관리형 키(SSE-S3) 또는 AWS KMS keys(SSE-KMS, AWS 관리형 키 및 고객 관리형 키 포함)와 함께 사용할지를 나타냅니다. SSE_S3 및 SSE_KMS 설정은 Aurora MySQL 버전 3.05 이상에서 사용할 수 있습니다.

다음 예시와 같이 ENCRYPTION 절 대신 aurora_select_into_s3_encryption_default 세션 변수를 사용할 수도 있습니다. SQL 절이나 세션 변수를 사용하되, 둘 다 사용하지는 마세요.

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

SSE_S3 및 SSE_KMS 설정은 Aurora MySQL 버전 3.05 이상에서 사용할 수 있습니다.

다음 값으로 aurora_select_into_s3_encryption_default를 설정하는 경우:

- OFF - S3 버킷의 기본 암호화 정책이 적용됩니다. aurora_select_into_s3_encryption_default의 기본값은 OFF입니다.
- ON 또는 SSE_S3 - Amazon S3 관리형 키(SSE-S3)를 사용하여 S3 객체가 암호화됩니다.
- SSE_KMS - S3 객체는 AWS KMS key를 사용하여 암호화됩니다.

  이 경우 세션 변수 aurora_s3_default_cmk_id도 포함합니다. 예:

  ```
  set session aurora_select_into_s3_encryption_default={SSE_KMS};
  set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
  ```

  - aurora_s3_default_cmk_id가 NULL인 경우 S3 객체는 AWS 관리형 키를 사용하여 암호화됩니다.
  - aurora_s3_default_cmk_id가 비어 있지 않은 cmk_id 문자열인 경우 S3 객체는 고객 관리형 키를 사용하여 암호화됩니다.

    cmk_id의 값은 빈 문자열일 수 없습니다.

SELECT INTO OUTFILE S3 명령을 사용할 때 Aurora는 다음과 같이 암호화를 결정합니다.

- SQL 명령에 ENCRYPTION 절이 있는 경우 Aurora는 ENCRYPTION의 값에만 의존하며 세션 변수를 사용하지 않습니다.
- ENCRYPTION 절이 없는 경우 Aurora는 세션 변수의 값을 사용합니다.

자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 관리형 키(SSE-S3)를 사용한 서버 측 암호화 사용 및 AWS KMS 키(SSE-KMS)를 사용한 서버 측 암호화 사용을 참조하세요.

MySQL 설명서의 SELECT 문 및 LOAD DATA 문에서 다른 파라미터에 대한 자세한 내용을 확인할 수 있습니다.

고려 사항

Amazon S3 버킷에 만들어지는 파일 개수는 SELECT INTO OUTFILE S3 문에서 선택된 데이터의 양과 Aurora MySQL의 파일 크기 임계값에 따라 달라집니다. 기본 파일 크기 임계값은 6기가바이트(GB)입니다. 문에서 선택된 데이터가 파일 크기 임계값보다 작으면 파일이 한 개 만들어지고, 그렇지 않으면 파일이 여러 개 만들어집니다. 이 문으로 만들어지는 파일에 대한 다른 고려 사항은 다음과 같습니다.

- Aurora MySQL은 데이터 파일의 행이 파일 경계를 너머 분할되지 않게 해줍니다. 파일이 여러 개인 경우 마지막 파일을 제외하고 각 데이터 파일의 크기는 기본적으로 파일 크기 임계값에 근접합니다. 하지만 파일 크기 임계값을 준수하려면 행이 두 데이터 파일로 분할되는 경우가 간혹 있습니다. 이런 경우 Aurora MySQL이 행을 원래대로 유지하는 데이터 파일을 만들지만 파일 크기 임계값이 초과될 수 있습니다.
- Aurora MySQL의 각 `SELECT` 문은 원자성 트랜잭션으로 실행되므로 대량 데이터 세트를 선택하는 `SELECT INTO OUTFILE S3` 문은 실행하는 데 시간이 약간 걸릴 수 있습니다. 어떤 이유든 문이 실패할 경우 처음부터 다시 문을 실행해야 합니다. 하지만 문이 실패할 경우 이미 Amazon S3에 업로드된 파일은 지정된 Amazon S3 버킷에 유지됩니다. 처음부터 다시 시작하는 대신 다른 문을 사용하여 나머지 데이터를 업로드할 수 있습니다.
- 선택할 데이터의 양이 클 경우(25GB 이상) 여러 `SELECT INTO OUTFILE S3` 문을 사용하여 데이터를 Amazon S3에 저장하는 것이 좋습니다. 각 문은 저장할 데이터의 서로 다른 부분을 선택해야 하며, 데이터 파일을 저장할 때 사용할 `file_prefix` 파라미터에도 서로 다른 `s3-uri`를 지정해야 합니다. 여러 문으로 선택할 데이터를 파티셔닝하면 한 문의 오류에서 쉽게 복구할 수 있습니다. 하나의 문에 오류가 발생하면 데이터의 일부만 다시 선택하여 Amazon S3에 업로드하면 됩니다. 여러 문을 사용할 경우 한 트랜잭션을 장시간 실행하지 않아도 되므로 성과를 개선할 수 있습니다.
- `SELECT INTO OUTFILE S3` 파라미터에 동일한 `file_prefix`를 사용하는 여러 `s3-uri` 문이 동시에 실행되어 Amazon S3로 데이터를 선택할 경우 동작이 정의되지 않습니다.
- 테이블 스키마 또는 파일 메타데이터와 같은 메타데이터는 Aurora MySQL이 Amazon S3으로 업로드하지 않습니다.
- 실패를 복구하는 경우와 같이 `SELECT INTO OUTFILE S3` 쿼리를 재실행하는 경우가 있을 수 있습니다. 이러한 경우 Amazon S3 버킷에서 `s3-uri`에 지정된 동일한 파일 접두사를 포함하는 기존의 데이터 파일을 모두 제거하거나 `OVERWRITE ON` 쿼리에서 `SELECT INTO OUTFILE S3`을 포함해야 합니다.

`SELECT INTO OUTFILE S3` 문은 성공 또는 실패 시 일반적인 MySQL 오류 번호 및 응답을 반환합니다. MySQL 오류 번호 및 응답에 액세스할 수 없을 경우 완료 여부를 확인할 수 있는 가장 간단한 방법은 문에 `MANIFEST ON`을 지정하는 것입니다. 매니페스트 파일은 문이 기록하는 마지막 파일입니다. 즉, 매니페스트 파일이 있으면 문이 완료된 것입니다.

현재 `SELECT INTO OUTFILE S3` 문이 실행되는 동안 진행률을 직접 모니터링하는 방법은 없습니다. 하지만 이 문을 사용하여 Aurora MySQL에서 Amazon S3으로 대량의 데이터를 기록할 때 문에서 선택된 데이터의 크기를 알고 있는 경우 Amazon S3에 만들어진 데이터 파일을 모니터링하여 진행률을 추정할 수 있습니다.

이렇게 하기 위해 명령문에서 선택한 6GB 데이터마다, 지정된 Amazon S3 버킷에 데이터 파일이 한 개 만들어진다는 점을 이용할 수 있습니다. 선택된 데이터 크기를 6GB로 나누어 만들어질 데이터 파일 수를 추측합니다. 그런 다음 문이 실행되는 동안 Amazon S3에 업로드된 파일 수를 모니터링하여 진행률을 추정할 수 있습니다.

예시

다음 문은 employees 테이블에서 모든 데이터를 선택하고 Aurora MySQL DB 클러스터와 다른 리전에 속한 Amazon S3 버킷에 데이터를 저장합니다. 이 문은 각 필드가 쉼표(,) 문자로 끝나고 각 줄이 줄바꿈(\n) 문자로 끝나는 데이터 파일을 만듭니다. 이 문은 sample_employee_data 파일 접두사와 일치하는 파일이 지정된 Amazon S3 버킷에 존재할 경우 오류를 반환합니다.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

다음 문은 employees 테이블에서 모든 데이터를 선택하고 Aurora MySQL DB 클러스터와 같은 리전에 속한 Amazon S3 버킷에 데이터를 저장합니다. 이 문은 각 필드가 쉼표(,) 문자로 끝나고 각 줄이 줄바꿈(\n) 문자로 끝나는 데이터 파일을 만들고 매니페스트 파일도 만듭니다. 이 문은 sample_employee_data 파일 접두사와 일치하는 파일이 지정된 Amazon S3 버킷에 존재할 경우 오류를 반환합니다.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

다음 문은 employees 테이블에서 모든 데이터를 선택하고 Aurora DB 클러스터와 다른 리전에 속한 Amazon S3 버킷에 데이터를 저장합니다. 이 문은 각 필드가 쉼표(,) 문자로 끝나고 각 줄이 줄바꿈(\n) 문자로 끝나는 데이터 파일을 만듭니다. 이 문은 지정된 Amazon S3 버킷에서 sample_employee_data 파일 접두사와 일치하는 기존 파일을 모두 덮어씁니다.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

다음 문은 employees 테이블에서 모든 데이터를 선택하고 Aurora MySQL DB 클러스터와 같은 리전에 속한 Amazon S3 버킷에 데이터를 저장합니다. 이 문은 각 필드가 쉼표(,) 문자로 끝나고 각 줄이 줄바꿈(\n) 문자로 끝나는 데이터 파일을 만들고 매니페스트 파일도 만듭니다. 이 문은 지정된 Amazon S3 버킷에서 sample_employee_data 파일 접두사와 일치하는 기존 파일을 모두 덮어씁니다.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출

네이티브 함수 lambda_sync 또는 lambda_async를 사용하여 Amazon Aurora MySQL 호환 버전 DB 클러스터에서 AWS Lambda 함수를 호출할 수 있습니다. Aurora MySQL에서 Lambda 함수를 호출하기 전에 Aurora DB 클러스터가 Lambda에 액세스해야 합니다. Aurora MySQL에 액세스 권한을 부여하는 방법에 대한 자세한 내용은 Aurora에 Lambda에 대한 액세스 권한 부여 섹션을 참조하세요. lambda_sync 및 lambda_async 저장 함수에 대한 자세한 내용은 Aurora MySQL 네이티브 함수로 Lambda 함수 호출 섹션을 참조하세요.

또한 저장 프로시저를 사용하여 AWS Lambda 함수를 호출할 수 있습니다. 그러나 저장 프로시저는 더 이상 사용되지 않습니다. 다음 Aurora MySQL 버전 중 하나를 사용 중인 경우 Aurora MySQL 네이티브 함수를 사용할 것을 적극 권장합니다.

- MySQL 5.7 호환 클러스터의 경우 Aurora MySQL 버전 2.
- MySQL 8.0 호환 클러스터의 경우 Aurora MySQL 버전 3.01 이상. Aurora MySQL 버전 3에서는 저장 프로시저를 사용할 수 없습니다.

Aurora에 Lambda 액세스 권한을 제공하고 Lambda 함수를 간접 호출하는 방법에 대한 내용은 다음 주제를 참조하세요.

주제

Aurora에 Lambda에 대한 액세스 권한 부여

Aurora MySQL DB 클러스터에서 Lambda 함수를 호출하기 전에 먼저 클러스터에 Lambda 액세스 권한을 부여해야 합니다.

Aurora MySQL에 Lambda에 대한 액세스 권한을 부여하려면

1. Aurora MySQL DB 클러스터에서 Lambda 함수를 호출하도록 허용하는 권한을 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하세요. 지침은 AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성 섹션을 참조하세요.
2. IAM 역할을 생성하고 AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 섹션을 참조하세요.
3. `aws_default_lambda_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정하십시오.

   클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 동일한 설정을 적용합니다.

   DB 클러스터 파라미터에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 단원을 참조하십시오.
4. Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Lambda 함수를 호출하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 해당 DB 클러스터와 연결하십시오. IAM 역할과 DB 클러스터 연결에 대한 자세한 내용은 IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 단원을 참조하십시오.

   클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 해당 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.
5. Lambda으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

   클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 발신 연결을 활성화합니다.

Aurora MySQL 네이티브 함수로 Lambda 함수 호출

> **Note**
>
> Aurora MySQL 버전 2 또는 Aurora MySQL 3.01 이상을 사용하는 경우 `lambda_sync` 및 `lambda_async` 네이티브 함수를 호출할 수 있습니다. Aurora MySQL 버전에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

네이티브 함수 `lambda_sync` 및 `lambda_async`를 호출하여 Aurora MySQL DB 클러스터에서 AWS Lambda 함수를 호출할 수 있습니다. 이 방식은 Aurora MySQL에서 실행 중인 데이터베이스를 다른 AWS 서비스와 통합하려는 경우에 유용합니다. 예를 들어 데이터베이스의 특정 테이블에 행이 삽입될 때마다 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알림을 보낼 수 있습니다.

목차

Lambda 함수를 호출하기 위해 네이티브 함수로 작업

`lambda_sync` 및 `lambda_async` 함수는 내장된 네이티브 함수이며 동기적 또는 비동기적으로 Lambda 함수를 호출합니다. 다른 작업으로 넘어가기 전에 Lambda 함수의 결과를 알아야 하는 경우, 동기식 함수 `lambda_sync`를 사용합니다. 다른 작업으로 넘어가기 전에 Lambda함수의 결과를 알 필요가 없는 경우, 비동기식 함수 `lambda_async`를 사용합니다.

Aurora MySQL 버전 3에서의 역할 부여

Aurora MySQL 버전 3에서는 네이티브 함수를 호출하는 사용자에게는 AWS_LAMBDA_ACCESS 역할이 부여되어야 합니다. 사용자에게 이 역할을 부여하려면 관리자 사용자로 DB 인스턴스에 연결하고 다음 명령문을 실행합니다.

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

다음 명령문을 실행하여 이 역할을 호출할 수 있습니다.

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> Aurora MySQL 버전 3에서 역할을 사용하는 경우 `SET ROLE role_name` 또는 `SET ROLE ALL` 문을 사용하여 해당 역할을 활성화할 수도 있습니다. MySQL 8.0 역할 시스템에 익숙하지 않을 경우 [역할 기반 권한 모델]에서 자세한 내용을 확인할 수 있습니다. 자세한 내용은 MySQL 참조 설명서의 [역할 사용]을 참조하세요.
> 이는 현재 활성 세션에만 적용됩니다. 다시 연결할 때 권한을 부여하려면 `SET ROLE` 문을 다시 실행해야 합니다. 자세한 내용은 MySQL 참조 매뉴얼의 [SET ROLE 문]을 참조하세요.
> `activate_all_roles_on_login` DB 클러스터 파라미터로 사용자가 DB 인스턴스에 연결할 때 모든 역할을 자동으로 활성화할 수 있습니다. 이 파라미터를 설정하면 역할을 활성화하는 데 `SET ROLE` 문을 명시적으로 직접 호출할 필요가 없습니다. 자세한 내용은 MySQL Reference Manual(MySQL 참조 매뉴얼)의 [activate_all_roles_on_login]을 참조하세요.
> 하지만 다른 사용자가 저장 프로시저를 직접 호출한 경우에는 저장 프로시저를 시작할 때 명시적으로 `SET ROLE ALL`을 직접 호출하여 역할을 활성화해야 합니다.

Lambda 함수를 간접적으로 호출하려고 할 때 다음과 같은 오류가 발생하면 `SET ROLE` 문을 실행합니다.

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

`mysql.users` 테이블 항목에 표시된 대로 올바른 사용자에게 역할을 부여하고 있는지 확인합니다. 이름이 동일하지만 호스트가 다른 사용자가 여러 명 있을 수 있습니다. `lambda_sync` 함수를 간접 호출하는 애플리케이션 또는 호스트에 따라 MySQL은 `host` 열 항목을 기준으로 가장 일치하는 사용자를 선택합니다.

Aurora MySQL 버전 2에서의 권한 부여

Aurora MySQL 버전 2에서 네이티브 함수를 호출하는 사용자에게는 `INVOKE LAMBDA` 권한이 부여되어야 합니다. 사용자에게 이 권한을 부여하려면 관리자 사용자로 DB 인스턴스에 연결하고 다음 명령문을 실행합니다.

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

다음 명령문을 실행하여 이 권한을 호출할 수 있습니다.

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

lambda_sync 함수의 구문

`lambda_sync` 호출 유형으로 `RequestResponse` 함수를 동기식으로 호출합니다. 함수가 JSON 페이로드에서 Lambda 호출 결과를 반환합니다. 함수에는 다음과 같은 구문이 있습니다.

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_sync 함수의 파라미터

`lambda_sync` 함수에는 다음과 같은 파라미터가 있습니다.

lambda_function_ARN

호출할 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

JSON_payload

JSON 형식으로 호출된 Lambda 함수의 페이로드입니다.

> **Note**
>
> Aurora MySQL 버전 3은 MySQL 8.0의 JSON 구문 분석 함수를 지원합니다. 그러나 Aurora MySQL 버전 2에는 이러한 함수가 포함되어 있지 않습니다. Lambda 함수가 숫자 또는 문자열과 같은 원자 값을 반환할 때 JSON 구문 분석이 필요하지 않습니다.

lambda_sync 함수의 예

lambda_sync를 기반으로 하는 다음 쿼리는 함수 ARN을 사용하여 Lambda 함수 BasicTestLambda를 동기식으로 호출합니다. 함수에 대한 페이로드는 {"operation": "ping"}입니다.

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

lambda_async 함수의 구문

lambda_async 호출 유형으로 Event 함수를 비동기식으로 호출합니다. 함수가 JSON 페이로드에서 Lambda 호출 결과를 반환합니다. 함수에는 다음과 같은 구문이 있습니다.

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_async 함수의 파라미터

lambda_async 함수에는 다음과 같은 파라미터가 있습니다.

lambda_function_ARN

호출할 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

JSON_payload

JSON 형식으로 호출된 Lambda 함수의 페이로드입니다.

> **Note**
>
> Aurora MySQL 버전 3은 MySQL 8.0의 JSON 구문 분석 함수를 지원합니다. 그러나 Aurora MySQL 버전 2에는 이러한 함수가 포함되어 있지 않습니다. Lambda 함수가 숫자 또는 문자열과 같은 원자 값을 반환할 때 JSON 구문 분석이 필요하지 않습니다.

lambda_async 함수의 예

`lambda_async`를 기반으로 하는 다음 쿼리는 함수 ARN을 사용하여 Lambda 함수 `BasicTestLambda`를 비동기식으로 호출합니다. 함수에 대한 페이로드는 `{"operation": "ping"}`입니다.

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

트리거 내에서 Lambda 함수 호출

트리거를 사용하여 데이터 수정 문에서 Lambda을 호출할 수 있습니다. 다음 예제에서는 `lambda_async` 기본 함수를 사용하고 결과를 변수에 저장합니다.

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> 트리거는 SQL 문당 한 번이 아니라 수정된 행당 한 번 실행되며 한 번에 한 행씩 실행됩니다. 트리거가 실행되면 프로세스는 동기식입니다. 트리거가 완료될 때만 데이터 수정 문이 반환합니다.
>
> 쓰기 트래픽이 높은 테이블의 트리거에서 AWS Lambda 함수를 호출할 때는 주의해야 합니다. `INSERT`, `UPDATE` 및 `DELETE` 트리거는 행별로 활성화됩니다. `INSERT`, `UPDATE` 또는 `DELETE` 트리거가 있는 테이블에서 쓰기가 많은 워크로드는 AWS Lambda 함수에 다량의 호출을 야기합니다.

Aurora MySQL 저장 프로시저(사용되지 않음)로 Lambda 함수 호출

mysql.lambda_async 프로시저를 호출하여 Aurora MySQL DB 클러스터에서 AWS Lambda 함수를 호출할 수 있습니다. 이 방식은 Aurora MySQL에서 실행 중인 데이터베이스를 다른 AWS 서비스와 통합하려는 경우에 유용합니다. 예를 들어 데이터베이스의 특정 테이블에 행이 삽입될 때마다 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알림을 보낼 수 있습니다.

목차

Aurora MySQL 버전 고려 사항

Aurora MySQL 버전 2부터 이러한 저장 프로시저 대신 네이티브 함수 메서드를 사용하여 Lambda 함수를 호출할 수 있습니다. 네이티브 함수에 대한 자세한 내용은 Lambda 함수를 호출하기 위해 네이티브 함수로 작업 단원을 참조하십시오.

Aurora MySQL 버전 2에서 mysql.lambda_async 저장 프로시저는 더 이상 지원되지 않습니다. 대신 네이티브 Lambda 함수를 사용하는 것을 권장합니다.

Aurora MySQL 버전 3에서 저장 프로시저는 사용할 수 없습니다.

Lambda 함수(사용되지 않음) 호출을 위한 mysql.lambda_async 프로시저로 작업

mysql.lambda_async 프로시저는 Lambda 함수를 비동기 방식으로 호출하는 기본 제공 저장 프로시저입니다. 이 프로시저를 사용하려면 데이터베이스 사용자가 EXECUTE 저장 프로시저에 대한 mysql.lambda_async 권한이 있어야 합니다.

구문

mysql.lambda_async 프로시저에는 다음과 같은 구문이 있습니다.

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
```

```
)
```

파라미터

`mysql.lambda_async` 프로시저에는 다음과 같은 파라미터가 있습니다.

lambda_function_ARN

호출할 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

lambda_function_input

호출되는 Lambda 함수에 대한 입력 문자열(JSON 형식)입니다.

예시

트리거나 클라이언트 코드와 같은 여러 소스에서 호출할 수 있는 저장 프로시저의 `mysql.lambda_async` 프로시저에 대한 호출을 래핑하는 것이 좋습니다. 이 방식을 사용하면 임피던스 불일치 문제를 방지하고 Lambda 함수를 보다 쉽게 호출할 수 있습니다.

> **Note**
> 쓰기 트래픽이 높은 테이블의 트리거에서 AWS Lambda 함수를 호출할 때는 주의해야 합니다. `INSERT`, `UPDATE` 및 `DELETE` 트리거는 행별로 활성화됩니다. `INSERT`, `UPDATE` 또는 `DELETE` 트리거가 있는 테이블에서 쓰기가 많은 워크로드는 AWS Lambda 함수에 다량의 호출을 야기합니다.
> `mysql.lambda_async` 절차에 대한 호출은 비동기식이지만 트리거는 동기식입니다. 다량의 트리거 활성화를 야기하는 명령문은 AWS Lambda 기능 종료에 대한 호출을 기다리지 않고, 다만 트리거 종료를 기다린 다음에 클라이언트로 제어를 돌려줍니다.

Example 예: AWS Lambda 함수를 호출하여 이메일 보내기

다음 예제에서는 Lambda 함수를 사용하여 이메일을 보내기 위해 데이터베이스 코드에서 호출할 수 있는 저장 프로시저를 생성합니다.

AWS Lambda 함수

```
import boto3
```

```
ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

저장 프로시저

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

저장 프로시저를 호출하여 AWS Lambda 함수 호출

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example 예: AWS Lambda 함수를 호출하여 트리거에서 이벤트 게시

다음 예제에서는 Amazon SNS를 사용하여 이벤트를 게시하는 저장 프로시저를 생성합니다. 이 코드는 테이블에 행이 추가되면 트리거에서 프로시저를 호출합니다.

AWS Lambda 함수

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

저장 프로시저

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

표

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

트리거

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

알림 트리거를 위하여 테이블에 행 삽입

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시

일반, 느린, 감사, 오류 로그 데이터를 Amazon CloudWatch Logs의 로그 그룹에 게시하도록 Aurora MySQL DB 클러스터를 구성할 수 있습니다. CloudWatch Logs를 통해 로그 데이터에 대한 실시간 분석을 수행할 수 있고, CloudWatch를 사용하여 경보를 만들고 지표를 볼 수 있습니다. CloudWatch Logs를 사용하여 내구성이 뛰어난 스토리지에 로그 레코드를 저장할 수 있습니다.

CloudWatch Logs로 로그를 게시하려면 각 로그를 활성화해야 합니다. 오류 로그는 기본적으로 활성화되어 있지만 다른 유형의 로그는 명시적으로 활성화해야 합니다. MySQL에서 로그를 활성화하는 내용은 MySQL 설명서의 일반 쿼리와 느린 쿼리 로그 출력 대상 선택을 참조하십시오. Aurora MySQL 감사 로그 활성화에 대한 자세한 내용은 고급 감사 활성화 단원을 참조하십시오.

> **Note**
>
> - 로그 데이터 내보내기를 비활성화하면 Aurora가 기존 로그 그룹 또는 로그 스트림을 삭제하지 않습니다. 로그 데이터 내보내기를 비활성화하면 CloudWatch Logs에서 기존 로그 데이터를 계속 사용할 수 있으며, 로그 보존에 따라 저장된 감사 로그 데이터 비용이 발생합니다. CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 로그 스트림 및 로그 그룹을 삭제할 수 있습니다.
> - 감사 로그를 CloudWatch Logs에 게시하는 다른 방법은 고급 감사를 활성화한 다음 사용자 지정 DB 클러스터 파라미터 그룹을 만들고 `server_audit_logs_upload` 파라미터를 1로 설정하는 것입니다. `server_audit_logs_upload` DB 클러스터 파라미터의 기본값은 `0`입니다. 고급 감사 활성화에 대한 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용](#) 섹션을 참조하세요.
>
>   대안인 이 방법을 사용하는 경우 CloudWatch Logs에 액세스하고 `aws_default_logs_role` 클러스터 수준 파라미터를 이 역할에 대한 ARN으로 설정하는 IAM 역할을 보유해야 합니다. 역할에 대한 상세 정보는 [AWS 서비스에 액세스할 수 있는 IAM 역할 설정](#) 단원을 참조하십시오. 하지만 `AWSServiceRoleForRDS` 서비스 연결 역할이 있는 경우 CloudWatch Logs에 대한 액세스를 제공하고 모든 사용자 정의 역할을 무시합니다. Amazon RDS에 대한 서비스 연결 역할에 대한 자세한 내용은 [Amazon Aurora에 서비스 연결 역할 사용](#) 섹션을 참조하세요.
> - 감사 로그를 CloudWatch Logs로 내보내고 싶지 않은 경우, 감사 로그를 내보내는 모든 방법이 비활성화되었는지 확인하십시오. AWS Management Console, AWS CLI, RDS API, `server_audit_logs_upload` 파라미터가 그러한 방법에 해당합니다.
> - Aurora Serverless v1 DB 클러스터의 절차는 프로비저닝된 DB 인스턴스 또는 Aurora Serverless v2 DB 인스턴스의 절차와 약간 다릅니다. Aurora Serverless v1 클러스터는 사용자가 구성 파라미터를 통해 활성화하는 모든 로그를 자동으로 업로드합니다.
>
>   따라서 DB 클러스터 파라미터 그룹에서 다양한 로그 유형을 활성화하고 비활성화하는 방식으로 Aurora Serverless v1 DB 클러스터에 대한 로그 업로드를 활성화 또는 비활성화합니다. AWS Management Console, AWS CLI 또는 RDS API를 통해 클러스터 자체의 설정을 수정할 수는 없습니다. Aurora Serverless v1 클러스터에서 MySQL 로그 활성화 및 비활성화에 대한 자세한 내용은 [Aurora Serverless v1 파라미터 그룹](#) 섹션을 참조하세요.

콘솔

콘솔에서 프로비저닝된 클러스터에 대한 Aurora MySQL 로그를 CloudWatch Logs에 게시할 수 있습니다.

콘솔에서 Aurora MySQL 로그를 게시하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 로그 데이터를 게시하려는 DB 클러스터의 Aurora MySQL을 선택합니다.
4. 수정을 선택합니다.
5. 로그 내보내기 섹션에서 CloudWatch Logs에 게시하기 시작할 로그를 선택합니다.
6. 계속을 선택한 후, 요약 페이지에서 Modify DB Cluster(DB 클러스터 수정)를 선택합니다.

AWS CLI

AWS CLI에서 프로비저닝된 클러스터에 대한 Aurora MySQL 로그를 게시할 수 있습니다. 이를 위해서는 modify-db-cluster AWS CLI 명령을 다음 옵션과 함께 실행해야 합니다.

- `--db-cluster-identifier`—DB 클러스터 식별자입니다.
- `--cloudwatch-logs-export-configuration`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

또 다음 AWS CLI 명령 중 하나를 실행하여 Aurora MySQL 로그를 게시할 수 있습니다.

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

다음 옵션으로 AWS CLI 명령 중 하나를 실행합니다.

- `--db-cluster-identifier`—DB 클러스터 식별자입니다.
- `--engine`—데이터베이스 엔진입니다.
- `--enable-cloudwatch-logs-exports`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

실행하는 AWS CLI 명령에 따라 다른 옵션이 필요할 수 있습니다.

Example

다음 명령은 CloudWatch Logs에 로그 파일을 게시하도록 기존 Aurora MySQL DB 클러스터를 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Example

다음 명령은 CloudWatch Logs에 로그 파일을 게시하도록 Aurora MySQL DB 클러스터를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
```

```
    --enable-cloudwatch-logs-exports
'["error","general","slowquery","audit","instance"]'
```

RDS API

RDS API에서 프로비저닝된 클러스터에 대한 Aurora MySQL 로그를 게시할 수 있습니다. 이를 위해서는 다음 옵션과 함께 ModifyDBCluster 작업을 실행해야 합니다.

- `DBClusterIdentifier`—DB 클러스터 식별자입니다.
- `CloudwatchLogsExportConfiguration`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

또한 다음 RDS API 작업 중 하나를 실행하여 RDS API로 Aurora MySQL 로그를 게시할 수 있습니다.

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

다음 파라미터로 RDS API 작업을 실행합니다.

- `DBClusterIdentifier`—DB 클러스터 식별자입니다.
- `Engine`—데이터베이스 엔진입니다.
- `EnableCloudwatchLogsExports`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

실행하는 AWS CLI 명령에 따라 다른 파라미터가 필요할 수 있습니다.

Amazon CloudWatch에서 로그 이벤트 모니터링

Aurora MySQL 로그 이벤트를 활성화한 후에 Amazon CloudWatch Logs에서 이 이벤트를 모니터링할 수 있습니다. 다음 접두사 밑에 Aurora DB 클러스터의 새 로그 그룹이 자동으로 생성됩니다. 여기서 *cluster-name*은 DB 클러스터 이름, *log_type*은 로그 유형을 나타냅니다.

```
/aws/rds/cluster/cluster-name/log_type
```

예를 들어 `mydbcluster`라는 이름의 DB 클러스터에 느린 쿼리 로그를 포함하도록 내보내기 함수를 구성하면, 느린 쿼리 데이터가 `/aws/rds/cluster/mydbcluster/slowquery` 로그 그룹에 저장됩니다.

클러스터의 모든 인스턴스에 있는 이벤트가 서로 다른 로그 스트림을 사용하는 로그 그룹으로 이동합니다. 다음 조건 중 어디에 해당하는지에 따라 달라집니다.

- 지정된 이름의 로그 그룹이 존재합니다.

  Aurora는 기존 로그 그룹을 사용하여 클러스터의 로그 데이터를 내보냅니다. AWS CloudFormation 같은 자동 구성을 사용하여 미리 정의된 로그 보존 기간, 지표 필터 및 고객 액세스 권한이 있는 로그 그룹을 생성할 수 있습니다.
- 지정된 이름의 로그 그룹이 존재하지 않습니다.

  인스턴스의 로그 파일에서 일치하는 로그 항목이 감지되면 Aurora MySQL은 CloudWatch Logs에 새 로그 그룹을 자동으로 생성합니다. 이 로그 그룹에는 기본 로그 보존 기간인 [만료되지 않음(Never Expire)]이 사용됩니다.

  로그 보존 기간을 변경하려면 CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용합니다. CloudWatch Logs의 로그 보존 기간 변경에 대한 자세한 내용은 CloudWatch Logs에서 로그 데이터 보존 변경을 참조하십시오.

DB 클러스터의 로그 이벤트 내에서 정보를 검색하려면 CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용합니다. 로그 데이터 검색 및 필터링에 관한 자세한 내용은 로그 데이터 검색 및 필터를 참조하십시오.

Amazon Aurora MySQL 랩 모드

> **Important**
>
> 특정 DDL 작업의 효율성을 개선하는 빠른 DDL 최적화를 활성화하도록 랩 모드가 Aurora MySQL 버전 2에 도입되었습니다. Aurora MySQL 버전 3에서는 랩 모드가 제거되었으며, 빠른 DDL이 인스턴트 DDL이라는 MySQL 8.0 기능으로 대체되었습니다.

Aurora 랩 모드는 현재 Aurora 데이터베이스 버전에서 사용 가능한 Aurora 기능을 활성화하는 데 사용되지만 기본적으로 비활성화되어 있습니다. 프로덕션 DB 클러스터에서 Aurora 랩 모드 기능을 사용하지 않는 것이 좋지만 개발 및 테스트 환경에서 Aurora 랩 모드를 사용하여 DB 클러스터에 대해 이러한 기능을 활성화할 수 있습니다. Aurora 랩 모드가 활성화되어 있을 때 사용 가능한 Aurora 기능에 대한 자세한 내용은 Aurora 랩 모드 기능 단원을 참조하십시오.

`aurora_lab_mode` 파라미터는 기본 파라미터 그룹에 속하는 인스턴스 수준 파라미터입니다. 기본 파라미터 그룹에서는 이 파라미터가 `0`(비활성)으로 설정됩니다. Aurora 랩 모드를 활성화하려면 사용자 지정 파라미터 그룹을 생성하고 사용자 지정 파라미터 그룹에서 `aurora_lab_mode` 파라미터를 1(활성)로 설정한 후, 사용자 지정 파라미터 그룹을 사용하도록 Aurora 클러스터에서 하나 이상의 DB 클러스터를 수정하십시오. 그런 다음, 랩 모드 기능을 시도하기 위해 해당되는 인스턴스 엔드포인트에 연결하십시오. DB 파라미터 그룹 수정에 대한 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하십시오. 파라미터 그룹 및 Amazon Aurora에 대한 자세한 내용은 Aurora MySQL 구성 파라미터 단원을 참조하십시오.

Aurora 랩 모드 기능

현재 Aurora 랩 모드를 활성화하면 다음 Aurora 기능을 사용할 수 있습니다. 이러한 기능을 사용하려면 먼저 Aurora 랩 모드를 활성화해야 합니다.

빠른 DDL

이 기능을 사용하면 `ALTER TABLE` *`tbl_name`* `ADD COLUMN` *`col_name`* *`column_definition`* 작업을 거의 동시에 실행할 수 있습니다. 이 작업은 테이블을 복사하거나 다른 DML 명령문에 영향을 거의 주지 않고 완료됩니다. 이는 테이블 복사를 위해 임시 스토리지를 사용하지 않으므로, 소형 인스턴스 클래스의 대형 테이블에 대해서도 DDL 문을 유용하게 만듭니다.

현재 빠른 DDL은 테이블 끝에서 기본값 없이 null이 허용된 열에 대해서만 지원됩니다. 이 기능 사용에 대한 자세한 내용은 빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정 단원을 참조하세요.

Amazon Aurora MySQL 모범 사례

이 주제에서는 Amazon Aurora MySQL DB 클러스터 사용 또는 이 클러스터로의 데이터 마이그레이션과 관련된 모범 사례와 옵션에 대해 설명합니다. 이 항목의 정보는 Amazon Aurora DB 클러스터 관리에서 찾을 수 있는 몇 가지 지침과 절차를 요약하고 반복합니다.

목차

- Amazon CloudWatch 지표를 사용한 Aurora MySQL에 대한 DB 인스턴스 사용량 평가

연결되어 있는 DB 인스턴스 확인

Aurora MySQL DB 클러스터에 어떤 DB 인스턴스가 연결되었는지 판별하려면 다음 예제와 같이 `innodb_read_only` 전역 변수를 점검하십시오.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

리더 DB 인스턴스에 연결된 경우 `innodb_read_only` 변수는 `ON`(으)로 설정됩니다. 라이터 DB 인스턴스(예: 프로비저닝된 클러스터의 기본 인스턴스)에 연결되어 있는 경우 이 설정은 `OFF`입니다.

이 접근 방식은 애플리케이션 코드에 논리를 추가하여 작업의 균형을 조정하거나 쓰기 작업에 올바른 연결이 사용되고 있는지 확인하려는 경우에 유용할 수 있습니다.

Aurora MySQL 성능 및 크기 조정에 대한 모범 사례

Aurora MySQL 클러스터의 성능과 확장성을 개선하기 위해 적용할 수 있는 모범 사례는 다음과 같습니다.

주제

- 개발 및 테스트에 T 인스턴스 클래스 사용
- 비동기식 키 프리페치를 사용하여 Aurora MySQL 인덱싱된 조인 쿼리 최적화
- 해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화
- Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정
- 타임스탬프 작업 최적화
- 가상 인덱스 ID 오버플로 오류

개발 및 테스트에 T 인스턴스 클래스 사용

`db.t2`, `db.t3` 또는 `db.t4g` DB 인스턴스 클래스를 사용하는 Amazon Aurora MySQL 인스턴스는 연장된 시간 동안 높은 워크로드를 지원하지 않는 애플리케이션에 가장 적합합니다. T 인스턴스는 중간 정도의 기본 성능을 발휘하면서 워크로드의 필요에 따라 성능을 크게 높이는 버스트 기능을 제공하도록 설계되었습니다. 이러한 인스턴스는 CPU의 최대 성능을 자주 또는 일관적으로 사용하지 않지만 가끔 순간적인 버스트가 필요한 워크로드에 적합합니다. T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 버스트 가능 성능 인스턴스를 참조하세요.

Aurora 클러스터가 40TB보다 큰 경우 T 인스턴스 클래스를 사용하지 마세요. 데이터베이스에 많은 양의 데이터가 있는 경우 스키마 개체를 관리하기 위한 메모리 오버헤드가 T 인스턴스의 용량을 초과할 수 있습니다.

MySQL 성능 스키마를 Amazon Aurora MySQL T 인스턴스에서 활성화하지 마세요. 성능 스키마가 활성화된 경우 인스턴스의 메모리가 부족할 수 있습니다.

> **Tip**
>
> 데이터베이스가 때로는 유휴 상태이지만 상당한 워크로드가 있을 때도 있는 경우 T 인스턴스의 대안으로 Aurora Serverless v2를 사용할 수 있습니다. Aurora Serverless v2를 사용하는 경우, 용량 범위를 정의하면 Aurora가 현재 워크로드에 따라 데이터베이스를 자동으로 확장 또는 축소합니다. 자세한 내용은 Aurora Serverless v2 사용하기 단원을 참조하세요. Aurora Serverless v2에서 사용할 수 있는 데이터베이스 엔진 버전은 Aurora Serverless v2 요구 사항 및 제한 사항 섹션을 참조하세요.

T 인스턴스를 Aurora MySQL DB 클러스터의 DB 인스턴스로 사용할 때는 다음을 권장합니다.

- DB 클러스터 내의 모든 인스턴스에 동일한 DB 인스턴스 클래스를 사용합니다. 예를 들어 라이터 인스턴스에 `db.t2.medium`을(를) 사용하는 경우 리더 인스턴스에 대해서도 `db.t2.medium`을(를) 사용하는 것이 좋습니다.
- 메모리 관련 구성 설정을 조정하지 마세요(예: `innodb_buffer_pool_size`). Aurora 는 T 인스턴스의 메모리 버퍼에 대해 고도로 조정된 기본값 집합을 사용합니다. 이러한 특수 기본값은 메모리가 제한된 인스턴스에서 Aurora 를 실행하는 데 필요합니다. T 인스턴스에서 메모리 관련 설정을 변경하면 버퍼 크기를 늘리려는 경우에도 메모리 부족 조건이 발생할 가능성이 훨씬 큽니다.
- CPU 크레딧 잔고(`CPUCreditBalance`)를 모니터링하여 지속 가능한 수준에 있는지 확인합니다. 즉 CPU 크레딧이 사용되고 있는 속도와 동일한 속도로 축적되고 있는지 확인합니다.

  한 인스턴스에 CPU 크레딧을 소진하면 사용 가능한 CPU의 즉각적인 하락과 그 인스턴스에 대한 읽기 및 쓰기 지연 시간의 증가가 표시됩니다. 이로 인해 인스턴스의 전반적인 성능이 크게 떨어집니다.

  CPU 크레딧 잔고가 지속 가능한 수준에 있지 않다면 DB 인스턴스를 수정하여 지원되는 R DB 인스턴스 클래스 중 하나를 사용하도록 하는 것이 좋습니다(컴퓨팅 확장).

  지표 모니터링에 대한 자세한 정보는 Amazon RDS 콘솔에서 지표 보기 단원을 참조하십시오.
- 라이터 인스턴스와 리더 인스턴스 간의 복제본 지연(`AuroraReplicaLag`)을 모니터링합니다.

라이터 인스턴스보다 먼저 리더 인스턴스에 CPU 크레딧이 부족하면 결과 지연으로 인해 리더 인스턴스가 자주 다시 시작될 수 있습니다. 애플리케이션에 리더 인스턴스의 많은 양의 읽기 작업이 분산되어 있는 동시에 라이터 인스턴스의 쓰기 작업의 양이 최소한일 때 일반적으로 나타나는 결과입니다.

복제 지연이 지속적으로 증가하는 경우에는 DB 클러스터의 리더 인스턴스에 대한 CPU 크레딧 잔고가 소진되지 않도록 해야 합니다.

CPU 크레딧 잔고가 지속 가능한 수준에 있지 않다면 DB 인스턴스를 수정하여 지원되는 R DB 인스턴스 클래스 중 하나를 사용하도록 하는 것이 좋습니다(컴퓨팅 확장).

- 바이너리 로깅이 활성화된 DB 클러스터에 대해서는 트랜잭션당 삽입의 수를 100만 개 이하로 유지해야 합니다.

  DB 클러스터에 대한 DB 클러스터 파라미터 그룹에서 `binlog_format` 파라미터가 `OFF`가 아닌 값으로 설정된 경우, DB 클러스터는 삽입할 행이 100만 개 이상 포함된 트랜잭션을 수신하면 메모리 부족 문제를 겪을 수 있습니다. 여유 메모리(`FreeableMemory`) 지표를 모니터링하여 DB 클러스터에 사용 가능 메모리가 부족한지 확인할 수 있습니다. 그런 다음 쓰기 작업(`VolumeWriteIOPS`) 지표를 점검하여 라이터 인스턴스가 많은 양의 쓰기 작업을 수신 중인지 확인할 수 있습니다. 그렇다면 애플리케이션을 업데이트하여 트랜잭션 내 삽입 수를 100만 개 미만으로 제한하는 것이 좋습니다. 또는 지원되는 R DB 인스턴스 클래스(컴퓨팅 확장) 중 하나를 사용하도록 인스턴스를 수정할 수 있습니다.

비동기식 키 프리페치를 사용하여 Aurora MySQL 인덱싱된 조인 쿼리 최적화

Amazon MySQL은 비동기식 키 프리페치(AKP) 기능을 사용하여 여러 인덱스 간에 테이블을 조인하는 쿼리 성능을 높일 수 있습니다. 이 기능은 JOIN 쿼리에서 Batched Key Access(BKA) 조인 알고리즘과 Multi-Range Read(MRR) 최적화 기능을 사용해야 하는 쿼리를 실행하면서 필요한 행을 예측하여 성능을 높이는 효과가 있습니다. BKA 및 MRR에 대한 자세한 정보는 MySQL 설명서에서 [Block Nested-Loop and Batched Key Access Joins](#) 및 [Multi-Range Read Optimization](#)을 참조하십시오.

쿼리가 AKP 기능을 이용하기 위해서는 BKA와 MRR이 모두 필요합니다. 일반적으로 JOIN 절이 보조 인덱스를 사용하지만 기본 인덱스의 열도 일부 필요할 때 이러한 쿼리가 발생합니다. 예를 들어 JOIN 절이 작은 용량의 외부 테이블과 큰 용량의 내부 테이블 사이에서 인덱스 값을 기준으로 한 등가 조인을 나타내고, 테이블 용량이 커질수록 인덱스 선택의 폭이 매우 제한적일 때 AKP를 사용할 수 있습니다. AKP는 JOIN 절을 평가하는 동안 BKA 및 MRR과 함께 보조-기본 인덱스 조회를 실행합니다. 동시에 쿼리를 실행하는 데 필요한 행까지 식별합니다. 그런 다음 쿼리를 실행하기에 앞서 백그라운드 스레드를 사용하여 식별된 행이 포함된 페이지를 메모리에 비동기식으로 로드합니다.

AKP는 Aurora MySQL 버전 2.10 이상 및 버전 3에서 사용할 수 있습니다. Aurora MySQL 버전에 대한 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.

비동기식 키 미리 가져오기(AKP) 활성화

MySQL 서버 변수 aurora_use_key_prefetch를 on으로 설정하여 AKP 기능을 활성화할 수 있습니다. 기본적으로 이 값은 on로 설정됩니다. 하지만 먼저 BKA 조인 알고리즘을 사용 설정하고 비용 기반 MRR 기능을 사용 중지해야만 AKP를 사용 설정할 수 있습니다. 이를 위해서는 optimizer_switch MySQL 서버 변수의 값을 다음과 같이 설정해야 합니다.

- batched_key_access를 on으로 설정합니다. 이 값은 BKA 조인 알고리즘의 사용을 제어합니다. 기본적으로 이 값은 off로 설정됩니다.
- mrr_cost_based를 off으로 설정합니다. 이 값은 비용 기반 MRR 기능의 사용을 제어합니다. 기본적으로 이 값은 on로 설정됩니다.

현재는 세션 수준에서만 위의 두 값을 설정할 수 있습니다. 다음은 SET 문에서 위의 두 값을 설정하여 현재 세션에 AKP를 활성화하는 방법을 설명한 예제입니다.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

마찬가지로 다음 예제와 같이 SET 문을 사용하여 AKP와 BKA 조인 알고리즘을 비활성화한 후 현재 세션에 비용 기반 MRR 기능을 다시 활성화할 수 있습니다.

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

batched_key_access 및 mrr_cost_based 옵티마이저 스위치에 대한 자세한 정보는 MySQL 설명서에서 Switchable Optimizations를 참조하십시오.

비동기식 키 미리 가져오기를 위한 쿼리 최적화

쿼리의 AKP 기능 사용 여부를 확인할 수 있습니다. 이렇게 하려면 쿼리를 실행하기 전에 EXPLAIN 문을 사용하여 쿼리를 프로파일링하면 됩니다. EXPLAIN 문이 지정한 쿼리에 사용할 실행 계획에 대한 정보를 제공합니다.

Extra 문 출력에서 EXPLAIN 열은 실행 계획에 추가되는 정보에 대한 설명입니다. AKP 기능이 쿼리에 사용할 테이블에 적용되는 경우 이 열에 다음 값 중 하나가 포함됩니다.

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

EXPLAIN을 사용하여 AKP를 이용할 수 있는 쿼리의 실행 계획을 확인하는 예는 다음과 같습니다.

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-----------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                              |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-----------------------------------------------------------+
```

```
|  1 | PRIMARY     | nation   | ALL  | PRIMARY                | NULL           | NULL
 | NULL                             |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                             |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey           | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY                | NULL           | NULL
 | NULL                             |   25 |   100.00 | Using where
                             |
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                             |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey           | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
+----+-------------+----------+------+-----------------------+---------------
+---------+----------------------------------+------+----------
+---------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

EXPLAIN 출력 형식에 대한 자세한 내용은 MySQL 설명서의 확장된 EXPLAIN 출력 형식을 참조하세요.

해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화

동등 조인을 사용하여 많은 양의 데이터를 조인해야 하는 경우 해시 조인을 통해 쿼리 성능을 향상시킬 수 있습니다. Aurora MySQL의 해시 조인을 활성화할 수 있습니다.

해시 조인 열은 복잡한 표현식이 될 수 있습니다. 해시 조인 열에서 다음과 같은 방식으로 데이터 유형을 비교할 수 있습니다.

- `int`, `bigint`, `numeric` 및 `bit` 등과 같은 정확한 숫자 데이터 형식 범주의 모든 항목을 비교할 수 있습니다.
- `float` 및 `double`과 같은 대략적인 숫자 데이터 형식 범주의 모든 항목을 비교할 수 있습니다.
- 문자열 유형에 동일한 문자 세트와 콜레이션이 있는 경우 문자열 유형간에 항목을 비교할 수 있습니다.
- 유형이 동일한 경우 날짜 및 타임스탬프 데이터 형식으로 항목을 비교할 수 있습니다.

> **Note**
>
> 다른 범주의 데이터 유형은 비교할 수 없습니다.

Aurora MySQL의 해시 조인에는 다음의 제한 사항이 적용됩니다.

- 왼쪽 오른쪽 외부 조인은 Aurora MySQL 버전 2에서는 지원되지 않지만, 버전 3에서는 지원됩니다.
- 하위 쿼리가 구체화되지 않는 한 하위 쿼리와 같은 Semijoin은 지원되지 않습니다.
- 다중 테이블의 업데이트 또는 삭제는 지원되지 않습니다.

> **Note**
>
> 단일 테이블의 업데이트 또는 삭제는 지원되지 않습니다.

- BLOB 및 공간 데이터 유형 열은 해시 조인의 조인 열일 수 없습니다.

해시 조인 활성화

해시 조인 활성화 방법:

- Aurora MySQL 버전 2 - DB 파라미터 또는 DB 클러스터 파라미터 `aurora_disable_hash_join`을 `0`으로 설정합니다. `aurora_disable_hash_join`을 비활성화하면 `optimizer_switch`의 값이 `hash_join=on`이 됩니다.
- Aurora MySQL 버전 3 - MySQL 서버 파라미터 `optimizer_switch`를 `block_nested_loop=on`으로 설정합니다.

해시 조인은 Aurora MySQL 버전 3에서 기본적으로 활성화되어 있으며 Aurora MySQL 버전 2에서 기본적으로 비활성화되어 있습니다. 다음은 Aurora MySQL 버전 3에서 해시 조인을 활성화하는 방법을 설명한 예입니다. 먼저 `select @@optimizer_switch` 문을 발행하여 다른 설정이 SET 파라미터 문자열에 있는지 확인합니다. `optimizer_switch` 파라미터에서 설정 하나를 업데이트하면 다른 설정을 지우거나 수정하지 못합니다.

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> Aurora MySQL 버전 3의 경우 해시 조인 지원은 모든 부 버전에서 사용할 수 있으며 기본적으로 켜져 있습니다.
> Aurora MySQL 버전 2의 경우 모든 마이너 버전에 해시 조인이 지원됩니다. Aurora MySQL 2 버전에서 해시 조인 기능은 항상 `aurora_disable_hash_join` 값에 의해 제어됩니다.

이 설정을 사용하면 옵티마이저는 비용, 쿼리 특성 및 리소스 가용성을 기반으로 해시 조인을 사용하도록 선택합니다. 비용 견적이 정확하지 않으면 옵티마이저가 해시 조인을 선택하게 할 수 있습니다. 그렇게 하려면 MySQL 서버 변수 `hash_join_cost_based`를 `off`으로 설정합니다. 다음은 옵티마이저가 해시 조인을 선택하도록 하는 방법을 설명한 예제입니다.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> 이 설정은 비용 기반 옵티마이저의 결정보다 우선합니다. 이 설정은 테스트 및 개발에 유용할 수 있지만, 프로덕션 환경에서는 사용하지 않는 것이 좋습니다.

해시 조인에 대한 쿼리 최적화

쿼리가 해시 조인을 활용할 수 있는지 확인하려면 `EXPLAIN` 문을 사용하여 쿼리를 먼저 프로파일링하십시오. `EXPLAIN` 문이 지정한 쿼리에 사용할 실행 계획에 대한 정보를 제공합니다.

`Extra` 문 출력에서 `EXPLAIN` 열은 실행 계획에 추가되는 정보에 대한 설명입니다. 해시 조인이 쿼리에 사용할 테이블에 적용되는 경우 이 열에 다음과 비슷한 값이 포함됩니다.

- `Using where; Using join buffer (Hash Join Outer table` *`table1_name`*`)`
- `Using where; Using join buffer (Hash Join Inner table` *`table2_name`*`)`

다음은 EXPLAIN을 사용하여 해시 조인 쿼리의 실행 계획을 확인하는 예제입니다.

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
```

```
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                          |
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)  |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2) |
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
3 rows in set (0.04 sec)
```

출력에서 `Hash Join Inner table`은 해시 테이블을 작성하는 데 사용하는 테이블이며 `Hash Join Outer table`은 해시 테이블을 프로브하는 데 사용하는 테이블입니다.

확장된 EXPLAIN 출력 형식에 대한 자세한 정보는 MySQL 제품 설명서에서 Extended EXPLAIN Output Format을 참조하십시오.

Aurora MySQL 2.08 이상에서는 SQL 힌트를 사용하여 쿼리에서 해시 조인을 사용하는지 여부와 조인의 빌드 및 프로브 측에 사용할 테이블에 영향을 줄 수 있습니다. 자세한 내용은 Aurora MySQL 힌트을 참조하세요.

Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정

MySQL DB 인스턴스에서 Amazon Aurora를 사용하여 Amazon Aurora의 읽기 조정 기능을 활용하고 MySQL DB 인스턴스에 대한 읽기 작업을 확장할 수 있습니다. Aurora을 사용하여 MySQL DB 인스턴스에 대한 읽기 조정을 수행하려면 Aurora MySQL DB 클러스터를 생성한 후 이 클러스터를 MySQL DB 인스턴스의 읽기 전용 복제본으로 설정합니다. 그런 다음 Aurora MySQL 클러스터에 연결하여 읽기 쿼리를 처리합니다. 이러한 설정은 RDS for MySQL DB 인스턴스 또는 Amazon RDS 외부에서 실행 중인 MySQL 데이터베이스에 적용됩니다. 자세한 내용은 Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 규모 조정 섹션을 참조하세요.

타임스탬프 작업 최적화

시스템 변수 `time_zone`의 값이 `SYSTEM`으로 설정되면 시간대 계산이 필요한 각 MySQL 함수 호출이 시스템 라이브러리를 호출합니다. 이러한 `TIMESTAMP` 값을 거의 동시에 반환하거나 변경하는 SQL 문

을 실행하면 지연 시간, 잠금 경합 및 CPU 사용량이 증가할 수 있습니다. 자세한 내용은 MySQL 설명서의 time_zone을 참조하세요.

이러한 동작을 방지하려면 `time_zone` DB 클러스터 파라미터의 값을 UTC로 변경하는 것이 좋습니다. 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

`time_zone` 파라미터는 동적이지만(데이터베이스 서버를 다시 시작할 필요가 없음), 새 값은 최신 연결에만 사용됩니다. 모든 연결이 새 `time_zone` 값을 사용하도록 업데이트하려면 DB 클러스터 파라미터를 업데이트한 후 애플리케이션 연결을 해제했다가 다시 연결하는 것이 좋습니다.

가상 인덱스 ID 오버플로 오류

Aurora MySQL은 가상 인덱스 ID 값을 8비트로 제한하여 MySQL의 실행 취소 형식으로 인한 문제를 방지합니다. 인덱스가 가상 인덱스 ID 제한을 초과하는 경우 클러스터를 사용하지 못할 수 있습니다. 인덱스가 가상 인덱스 ID 제한에 가까워지거나 가상 인덱스 ID 제한을 초과하여 인덱스를 만들려고 하면 RDS에서 오류 코드 63955 또는 경고 코드 63955가 발생할 수 있습니다. 가상 인덱스 ID 제한 오류를 해결하려면 논리적 덤프와 복원을 사용하여 데이터베이스를 다시 만드는 것이 좋습니다.

Amazon Aurora MySQL의 논리적 덤프 및 복원에 대한 자세한 내용은 Migrate very large databases to Amazon Aurora MySQL using MyDumper and MyLoader를 참조하세요. Amazon Aurora에서 오류 로그에 액세스하는 것에 대한 자세한 내용은 Amazon Aurora 로그 파일 모니터링 섹션을 참조하세요.

Aurora MySQL 고가용성을 위한 모범 사례

Aurora MySQL 클러스터의 유효성을 개선하기 위해 적용할 수 있는 모범 사례는 다음과 같습니다.

주제

- MySQL 데이터베이스에서 Amazon Aurora를 재해 복구용으로 사용
- 감소된 중단 시간으로 MySQL에서 Amazon Aurora MySQL로 마이그레이션
- Aurora MySQL DB 인스턴스의 성능 저하, 자동 재시작 및 장애 조치 방지

MySQL 데이터베이스에서 Amazon Aurora를 재해 복구용으로 사용

MySQL DB 인스턴스에서 Amazon Aurora를 사용하여 재해 복구용 오프사이트 백업을 생성할 수 있습니다. MySQL DB 인스턴스의 재해 복구용으로 Aurora을 사용하려면 Amazon Aurora DB 클러스터를 생성한 후 이 클러스터를 MySQL DB 인스턴스의 읽기 전용 복제본으로 설정합니다. 이러한 설정은 RDS for MySQL DB 인스턴스 또는 Amazon RDS 외부에서 실행 중인 MySQL 데이터베이스에 적용됩니다.

> **Important**
>
> MySQL DB 인스턴스와 Amazon Aurora MySQL DB 클러스터 간의 복제를 설정할 때 필요할 경우 정상 상태를 유지하고 수리하는지 확인하도록 복제를 모니터링해야 합니다.

Amazon Aurora MySQL DB 클러스터를 생성하고 이 클러스터를 MySQL DB 인스턴스의 읽기 전용 복제본으로 설정하는 방법에 대한 자세한 정보는 Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정의 절차를 따르십시오.

재해 복구 모델에 대한 자세한 내용은 Amazon Aurora MySQL 클러스터에 가장 적합한 재해 복구 옵션을 선택하는 방법을 참조하세요.

감소된 중단 시간으로 MySQL에서 Amazon Aurora MySQL로 마이그레이션

라이브 애플리케이션을 지원하는 MySQL 데이터베이스에서 Amazon Aurora MySQL DB 클러스터로 데이터를 가져올 때 마이그레이션하는 동안 서비스가 중단되는 시간을 줄일 수 있습니다. 그렇게 하려면 Amazon Relational Database Service 사용 설명서의 가동 중지 시간을 단축하여 MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기에 나온 절차를 따르세요. 이 절차는 대규모 데이터베이스로 작업하는 경우에 특히 유용합니다. 이 프로시저를 사용하여 네트워크에서 AWS로 전달되는 데이터의 양을 최소화하여 가져오기 비용을 줄일 수 있습니다.

이 절차에는 데이터베이스 데이터의 복사본을 Amazon EC2 인스턴스로 전송하고 데이터를 새 RDS for MySQL DB 인스턴스로 가져오는 작업을 수행하는 단계가 나열되어 있습니다. Amazon Aurora가 MySQL과 호환되므로 Amazon Aurora DB 클러스터를 대상 Amazon RDS MySQL DB 인스턴스로 대신 사용할 수 있습니다.

Aurora MySQL DB 인스턴스의 성능 저하, 자동 재시작 및 장애 조치 방지

워크로드가 많거나 워크로드가 DB 인스턴스의 할당된 리소스 이상으로 급증하는 경우 애플리케이션과 Aurora 데이터베이스를 실행하는 데 사용되는 리소스가 소진될 수 있습니다. CPU 사용률, 메모리 사용량, 사용된 데이터베이스 연결 수와 같은 데이터베이스 인스턴스에 대한 지표를 얻으려면 Amazon CloudWatch, 성능 개선 도우미 및 향상된 모니터링에서 제공하는 지표를 참조할 수 있습니다. DB 인스턴스 모니터링에 대한 자세한 내용은 Amazon Aurora 클러스터에서 지표 모니터링 섹션을 참조하세요.

워크로드가 사용 중인 리소스를 소진하면 DB 인스턴스가 느려지거나, 다시 시작되거나, 다른 DB 인스턴스로 장애 조치될 수 있습니다. 이를 방지하려면 리소스 사용률을 모니터링하고, DB 인스턴스에서 실행되는 워크로드를 검사하고, 필요한 경우 최적화하세요. 최적화가 인스턴스 지표를 개선하지 못하고 리소스 소진을 완화할 수 없다면 한도에 도달하기 전에 DB 인스턴스를 스케일 업하는 것을 고려해

보세요. 사용 가능한 DB 인스턴스 클래스 및 사양에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

Aurora MySQL에서 MySQL 기능에 대한 권장 사항

다음 기능은 MySQL 호환성을 위해 Aurora MySQL에서 사용할 수 있습니다. 그러나 Aurora 환경에서는 성능, 확장성, 안정성 또는 호환성 문제가 있습니다. 따라서 이러한 기능을 사용할 때 정해진 가이드라인을 따르는 것이 좋습니다. 예를 들어 프로덕션 Aurora 배포에는 특정 기능을 사용하지 않는 것이 좋습니다.

주제

Aurora MySQL에서 다중 스레드 복제 사용

다중 스레드 이진 로그 복제를 사용하면 SQL 스레드가 릴레이 로그에서 이벤트를 읽고 SQL 작업자 스레드에서 적용하도록 이벤트를 대기열에 넣습니다. SQL 작업자 스레드는 코디네이터 스레드에서 관리합니다. 가능한 경우 이진 로그 이벤트가 병렬로 적용됩니다.

다중 스레드 복제는 Aurora MySQL 버전 3, Aurora MySQL 버전 2.12.1 이상에서 지원됩니다.

Aurora MySQL 버전 3.04 미만에서 Aurora는 Aurora MySQL DB 클러스터가 이진 로그 복제의 읽기 전용 복제본으로 사용될 때 기본적으로 단일 스레드 복제를 사용합니다.

Aurora MySQL 버전 2의 초기 버전은 MySQL Community Edition에서 다중 스레드 복제와 관련된 여러 문제를 물려받았습니다. 이러한 버전의 경우 프로덕션 환경에서 다중 스레드 복제를 사용하지 않는 것이 좋습니다.

다중 스레드 복제를 사용할 경우 철저히 테스트할 것을 권장합니다.

Amazon Aurora에서의 복제에 대한 자세한 정보는 Amazon Aurora를 사용한 복제 단원을 참조하십시오. Aurora MySQL의 다중 스레드 복제에 대한 자세한 내용은 다중 스레드 이진 로그 복제 섹션을 참조하세요.

네이티브 MySQL 함수로 AWS Lambda 함수 호출

네이티브 MySQL 함수 `lambda_sync` 및 `lambda_async`를 사용하여 Lambda 함수를 호출하는 것이 좋습니다.

더 이상 사용되지 않는 `mysql.lambda_async` 프로시저를 사용하는 경우 저장 프로시저의 `mysql.lambda_async` 프로시저로 호출하는 것이 좋습니다. 트리거 또는 클라이언트 코드와 같은 여러 소스에서 저장 프로시저를 호출할 수 있습니다. 이 접근 방식을 통해 임피던스 불일치 문제를 방지하고 데이터베이스 프로그래머가 Lambda 함수를 보다 쉽게 호출할 수 있습니다.

Amazon Aurora에서 Lambda 함수 호출에 대한 자세한 정보는 [Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출](#) 단원을 참조하십시오.

Amazon Aurora MySQL에서 XA 트랜잭션 방지

XA가 `PREPARED` 상태인 경우 복구 시간이 길어질 수 있으므로 Aurora MySQL에서 XA(eXtended Architecture) 트랜잭션을 사용하지 않는 것이 좋습니다. Aurora MySQL에서 XA 트랜잭션을 사용해야 하는 경우 다음 모범 사례를 따르십시오.

- XA 트랜잭션을 `PREPARED` 상태로 열어두지 마십시오.
- XA 트랜잭션을 가능한 작게 유지하십시오.

MySQL에서 XA 트랜잭션 사용에 대한 자세한 정보는 MySQL 설명서의 [XA TRANSACTIONS](#)를 참조하십시오.

DML 문 동안 외래 키 켜기 유지

`foreign_key_checks` 변수가 `0`(off)으로 설정되어 있을 때는 데이터 정의 언어(DDL) 문을 실행하지 않는 것이 좋습니다.

외래 키의 일시적 위반이 필요한 행을 삽입하거나 업데이트해야 하는 경우, 다음 단계에 따르십시오.

1. `foreign_key_checks`를 `0`으로 설정합니다.
2. 데이터 조작 언어(DML)를 변경합니다.
3. 완료된 변경이 외래 키 제약 조건을 위반하지 않아야 합니다.
4. `foreign_key_checks`를 1(on)로 설정합니다.

또한 외래 키 제약 조건에 대한 다음과 같은 다른 모범 사례에 따르십시오.

- 클라이언트 애플리케이션이 init_connect 변수의 일부로 foreign_key_checks 변수를 0으로 설정하지 않아야 합니다.
- mysqldump와 같은 논리적 백업으로부터 복원이 실패하거나 불완전할 경우, 같은 세션에서 다른 작업을 시작하기 전에 foreign_key_checks가 1로 설정되어 있는지 확인합니다. 논리적 백업은 시작할 때 foreign_key_checks를 0으로 설정합니다.

로그 버퍼를 플러시하는 빈도 구성

MySQL Community Edition에서 트랜잭션의 내구성을 높이려면 InnoDB 로그 버퍼를 내구성 있는 스토리지로 플러시해야 합니다. innodb_flush_log_at_trx_commit 파라미터를 사용하여 로그 버퍼가 디스크로 플러시되는 빈도를 구성합니다.

innodb_flush_log_at_trx_commit 파라미터를 기본값인 1로 설정하면 트랜잭션이 커밋될 때마다 로그 버퍼가 플러시됩니다. 이 설정은 데이터베이스 ACID 준수 상태를 유지하는 데 도움이 됩니다. 기본 설정인 1을 그대로 유지하는 것이 좋습니다.

innodb_flush_log_at_trx_commit을 기본값이 아닌 값으로 변경하면 데이터 조작 언어(DML) 지연 시간을 줄이는 데 도움이 되지만 로그 레코드의 내구성은 저하될 수 있습니다. 이러한 내구성 결여로 인해 데이터베이스 ACID 규정을 준수하지 못하게 됩니다. 서버 재시작 시 데이터 손실 위험을 예방하기 위해 데이터베이스를 ACID 규정을 준수하는 것이 좋습니다. 이 파라미터에 대한 자세한 내용은 MySQL 설명서에서 innodb_flush_log_at_trx_commit을 참조하세요.

Aurora MySQL에서는 다시 실행 로그 처리가 스토리지 계층으로 오프로드되므로 DB 인스턴스에서 로그 파일을 플러시하지 않습니다. 쓰기가 실행되면 라이터 DB 인스턴스에서 Aurora 클러스터 볼륨으로 직접 다시 실행 로그가 전송됩니다. 네트워크를 통과하는 유일한 쓰기는 다시 실행 로그 레코드입니다. 데이터베이스 계층에서 작성되는 페이지는 없습니다.

기본적으로 트랜잭션을 커밋하는 각 스레드는 Aurora 클러스터 볼륨의 확인을 기다립니다. 이 확인은 이 레코드와 모든 이전 다시 실행 로그 레코드가 기록되어 쿼럼을 얻었음을 나타냅니다. 로그 레코드를 유지하고 쿼럼을 얻으면 자동 커밋이든 명시적 커밋이든 관계없이 트랜잭션의 내구성이 확보됩니다. Aurora 스토리지 아키텍처에 대한 자세한 내용은 Amazon Aurora 스토리지에 관한 이해를 돕는 문서를 참조하세요.

Aurora MySQL은 MySQL Community Edition처럼 로그를 데이터 파일로 플러시하지 않습니다. 하지만 innodb_flush_log_at_trx_commit 파라미터를 사용하여 다시 실행 로그 레코드를 Aurora 클러스터 볼륨에 쓸 때 내구성 제약을 완화할 수 있습니다.

Aurora MySQL 버전 2의 경우:

- `innodb_flush_log_at_trx_commit` = 0 또는 2 - 데이터베이스는 다시 실행 로그 레코드가 Aurora 클러스터 볼륨에 기록된다는 확인을 기다리지 않습니다.
- `innodb_flush_log_at_trx_commit` = 1 - 데이터베이스는 다시 실행 로그 레코드가 Aurora 클러스터 볼륨에 기록된다는 확인을 기다립니다.

Aurora MySQL 버전 3의 경우:

- `innodb_flush_log_at_trx_commit` = 0 - 데이터베이스는 다시 실행 로그 레코드가 Aurora 클러스터 볼륨에 기록된다는 확인을 기다리지 않습니다.
- `innodb_flush_log_at_trx_commit` = 1 또는 2 - 데이터베이스는 다시 실행 로그 레코드가 Aurora 클러스터 볼륨에 기록된다는 확인을 기다립니다.

따라서 Aurora MySQL 버전 2에서 값을 0 또는 2로 설정했을 때와 동일한 기본값이 아닌 동작이 작동하려면 Aurora MySQL 버전 3에서 파라미터를 0으로 설정하세요.

이러한 설정은 클라이언트에 대한 DML 지연 시간을 줄일 수 있지만 장애 조치 또는 재시작 시 데이터가 손실될 수도 있습니다. 따라서 `innodb_flush_log_at_trx_commit` 파라미터를 기본값 1로 유지하는 것이 좋습니다.

MySQL Community Edition과 Aurora MySQL 모두에서 데이터 손실이 발생할 수 있지만 아키텍처가 다르기 때문에 데이터베이스마다 동작이 다릅니다. 이러한 아키텍처 차이로 인해 데이터 손실 정도가 달라질 수 있습니다. 데이터베이스가 ACID 규정을 준수하도록 하려면 `innodb_flush_log_at_trx_commit`을 항상 1로 설정하세요.

> **Note**
>
> Aurora MySQL 버전 3에서는 `innodb_flush_log_at_trx_commit`을 1이 아닌 값으로 변경하려면 먼저 `innodb_trx_commit_allow_data_loss`의 값을 1로 변경해야 합니다. 이렇게 하면 데이터 손실의 위험을 인정하는 것으로 간주됩니다.

Aurora MySQL 교착 상태 최소화 및 문제 해결

고유한 보조 인덱스 또는 외래 키에 대한 제약 조건 위반이 정기적으로 발생하는 워크로드를 실행하는 사용자가 동일한 데이터 페이지의 레코드를 동시에 수정할 경우, 교착 상태가 증가하고 잠금 대기 시간 초과가 발생할 수 있습니다. 이러한 교착 상태 및 시간 초과는 MySQL Community Edition 버그 수정으로 인한 것입니다.

이 수정 사항은 MySQL Community Edition 버전 5.7.26 이상에 포함되었으며 Aurora MySQL 버전 2.10.3 이상에 백포트되었습니다. 이 수정 사항은 InnoDB 테이블의 레코드 변경 사항에 대해 이러한 유형의 데이터 조작 언어(DML) 작업에 대한 추가 잠금을 구현하여 직렬화 기능을 적용하는 데 필요합니다. 이 문제는 이전 MySQL Community Edition 버그 수정으로 인해 발생한 교착 상태 문제를 조사하는 과정에서 발견되었습니다.

이 수정 사항으로 InnoDB 스토리지 엔진의 튜플(행) 업데이트의 부분 롤백에 대한 내부 처리가 변경되었습니다. 외래 키 또는 고유한 보조 인덱스에서 제약 조건 위반을 생성하는 작업은 부분 롤백을 초래합니다. 여기에는 동시 `INSERT...ON DUPLICATE KEY UPDATE`, `REPLACE INTO`,, `INSERT IGNORE` 문(upserts)이 포함되며 이에 국한되지 않습니다.

이 컨텍스트에서 부분 롤백은 애플리케이션 레벨 트랜잭션의 롤백을 의미하지 않으며, 제약 조건 위반이 발생할 때 클러스터링된 인덱스에 대한 변경 사항의 내부 InnoDB 롤백을 의미합니다. 예를 들어 upsert 작업 중에 중복된 키 값이 발견된 경우를 가정해 보겠습니다.

일반적인 삽입 작업에서 InnoDB는 각 인덱스에 대해 클러스터링된 보조 인덱스 항목을 원자적으로 생성합니다. InnoDB가 upsert 작업 중에 고유한 보조 인덱스에서 중복 값을 감지하면 클러스터링된 인덱스에 삽입된 항목을 되돌리고(부분 롤백), 기존 중복 행에 업데이트를 적용해야 합니다. 이 내부 부분 롤백 단계에서 InnoDB는 작업의 일부로 간주되는 각 레코드를 잠가야 합니다. 이 수정 사항은 부분 롤백 후 추가 잠금을 도입하여 트랜잭션 직렬화 기능을 보장합니다.

InnoDB 교착 상태 최소화

다음과 같은 접근 방식을 사용하여 데이터베이스 인스턴스의 교착 상태 빈도를 줄일 수 있습니다. 더 많은 예제는 MySQL 설명서에서 찾을 수 있습니다.

1. 교착 상태가 발생할 가능성을 줄이려면 관련 변경 사항을 적용한 후 즉시 트랜잭션을 커밋하세요. 이렇게 하려면 큰 트랜잭션(커밋 간 다중 행 업데이트)을 작은 트랜잭션으로 나누면 됩니다. 행을 일괄적으로 삽입할 경우 특히 앞서 언급한 upsert 작업을 사용할 때는 배치 삽입 크기를 줄이세요.

   발생 가능한 부분 롤백 수를 줄이려면 다음 방법 중 일부를 시도해 봅니다.

   a. 배치 삽입 작업을 한 번에 한 행씩 삽입하는 방식으로 바꿉니다. 이렇게 하면 충돌이 발생할 수 있는 트랜잭션으로 인한 잠금 유지 시간을 줄일 수 있습니다.

   b. `REPLACE INTO`를 사용하는 대신 SQL 문을 다음과 같은 다중 문 트랜잭션으로 다시 작성합니다.

      ```
      BEGIN;
      DELETE conflicting rows;
      INSERT new rows;
      COMMIT;
      ```

c. INSERT...ON DUPLICATE KEY UPDATE를 사용하는 대신 SQL 문을 다음과 같은 다중 문 트랜잭션으로 다시 작성합니다.

```
BEGIN;
SELECT rows that conflict on secondary indexes;
UPDATE conflicting rows;
INSERT new rows;
COMMIT;
```

2. 잠금을 유지할 가능성이 있는 활성 또는 유휴 상태의 장기 실행 트랜잭션을 방지합니다. 여기에는 커밋되지 않은 트랜잭션으로 장기간 열려 있을 가능성이 있는 대화형 MySQL 클라이언트 세션이 포함됩니다. 트랜잭션 크기 또는 배치 크기를 최적화할 경우 동시성, 중복 수, 테이블 구조와 같은 여러 요인에 따라 미치는 영향이 달라질 수 있습니다. 모든 변경 사항은 워크로드를 기반으로 구현하고 테스트해야 합니다.
3. 상황에 따라 두 트랜잭션이 하나 또는 여러 테이블에서 서로 다른 순서로 동일한 데이터 세트에 액세스하려고 할 경우 교착 상태가 발생할 수 있습니다. 이를 방지하려면 같은 순서로 데이터에 액세스하도록 트랜잭션을 수정하여 액세스를 직렬화합니다. 예를 들어, 완료할 트랜잭션 대기열을 생성합니다. 이런 접근 방식은 여러 트랜잭션이 동시에 발생할 때 교착 상태를 방지하는 데 도움이 될 수 있습니다.
4. 신중하게 선택한 인덱스를 테이블에 추가하면 선택성이 향상되고 행에 액세스할 필요가 줄어들기 때문에 잠금 횟수도 감소합니다.
5. 격차 잠금이 발생할 경우 세션 또는 트랜잭션에 맞게 트랜잭션 격리 수준을 READ COMMITTED로 수정하여 이를 방지할 수 있습니다. InnoDB 격리 수준 및 해당 동작에 대한 자세한 내용은 MySQL 설명서의 트랜 잭션 격리 수준을 참조하세요.

> **Note**
>
> 교착 상태가 발생할 가능성을 줄이기 위한 예방 조치를 취할 수는 있지만, 교착 상태는 예상된 데이터베이스 동작이며 계속 발생할 수 있습니다. 애플리케이션에는 교착 상태가 발생했을 때 이를 처리하는 데 필요한 로직이 있어야 합니다. 예를 들어, 애플리케이션에서 재시도 및 백오프 로직을 구현합니다. 문제의 근본 원인을 해결하는 것이 가장 좋지만 교착 상태가 발생할 경우 애플리케이션에는 대기 후 다시 시도할 수 있는 옵션이 있습니다.

InnoDB 교착 상태 모니터링

애플리케이션 트랜잭션이 순환 대기를 야기하는 방식으로 테이블 수준 및 행 수준 잠금을 시도할 경우 MySQL에서 교착 상태가 발생할 수 있습니다. InnoDB 스토리지 엔진은 해당 조건을 즉시 감지하고 트랜잭션 중 하나를 자동으로 롤백하기 때문에 InnoDB 교착 상태가 가끔 발생하는 건 문제가 되지 않습니다. 단, 교착 상태가 자주 발생할 경우에는 애플리케이션을 검토 및 수정하여 성능 문제를 완화하고 교착 상태를 방지하는 것이 좋습니다. 교착 상태 감지가 켜져 있으면(기본값) InnoDB는 트랜잭션 교착 상태를 자동으로 감지하고 하나 이상의 트랜잭션을 롤백하여 교착 상태를 해제합니다. InnoDB는 롤백할 작은 트랜잭션을 선택하려는 시도를 합니다. 이때 트랜잭션의 크기는 삽입되거나, 업데이트되거나, 삭제된 행 수에 따라 결정됩니다.

- `SHOW ENGINE` 문 - `SHOW ENGINE INNODB STATUS \G` 문에는 마지막 재시작 이후 데이터베이스에서 발생한 가장 최근의 교착 상태에 대한 세부 정보가 포함됩니다.
- MySQL 오류 로그 - `SHOW ENGINE` 문의 출력이 부적절하여 교착 상태가 자주 발생할 경우 innodb_print_all_deadlocks DB 클러스터 파라미터를 켤 수 있습니다.

  이 파라미터를 켜면 InnoDB 사용자 트랜잭션의 모든 교착 상태에 대한 정보가 Aurora MySQL 오류 로그에 기록됩니다.
- Amazon CloudWatch 지표 - 또한 CloudWatch 지표 `Deadlocks`를 사용하여 교착 상태를 사전에 모니터링하는 것이 좋습니다. 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 단원을 참조하십시오.
- Amazon CloudWatch Logs - Amazon CloudWatch Logs를 통해 지표를 보고, 로그 데이터를 분석하고, 실시간 경보를 생성할 수 있습니다. 자세한 내용은 Amazon CloudWatch를 사용하여 Amazon Aurora MySQL 및 Amazon RDS for MySQL의 오류를 모니터링하고 Amazon SNS를 사용하여 알림 보내기를 참조하세요.

  `innodb_print_all_deadlocks`를 켠 상태로 CloudWatch Logs Logs를 사용하면 교착 상태 횟수가 지정된 임계값을 초과할 때 알려주는 경보를 구성할 수 있습니다. 임계값을 정의하려면 추세를 관찰하고 정상 워크로드를 기준으로 한 값을 사용하는 것이 좋습니다.
- 성능 개선 도우미 - 성능 개선 도우미를 사용하면 `innodb_deadlocks` 및 `innodb_lock_wait_timeout` 지표를 모니터링할 수 있습니다. 이러한 지표에 대한 자세한 내용은 기본이 아닌 Aurora MySQL용 카운터 섹션을 참조하세요.

Amazon CloudWatch 지표를 사용한 Aurora MySQL에 대한 DB 인스턴스 사용량 평가

CloudWatch 지표를 사용하여 유 인스턴스 처리량을 모니터링하고 DB 인스턴스 클래스가 애플리케이션에 제공하는 리소스가 충분한지 판단할 수 있습니다. DB 인스턴스 클래스 한도에 대한 자세한 내용은 [Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양](#) 섹션을 참조하세요. DB 인스턴스 클래스의 사양을 찾아 네트워크 성능을 찾습니다.

DB 인스턴스 사용량이 인스턴스 클래스 한도에 가까워지면 성능이 느려질 수 있습니다. CloudWatch 지표를 통해 이러한 상황을 확인할 수 있으므로 더 큰 인스턴스 클래스로 수동 스케일 업할 계획을 세울 수 있습니다.

다음 CloudWatch 지표 값을 조합하여 인스턴스 클래스 한도에 가까워졌는지 확인하세요.

- NetworkThroughput – Aurora DB 클러스터의 각 인스턴스에 대해 클라이언트가 수신 및 전송한 네트워크 처리량(throughput)입니다. 이 처리량(throughput) 값에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkThroughput – Aurora DB 클러스터의 각 인스턴스에서 Aurora 스토리지 하위 시스템으로 수신 및 전송한 네트워크 처리량(throughput)입니다.

NetworkThroughput을 StorageNetworkThroughput에 추가하여 Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신 및 전송한 네트워크 처리량(throughput)을 확인하세요. 인스턴스에 대한 인스턴스 클래스 한도는 이 두 지표를 합한 값보다 커야 합니다.

다음 지표를 사용하여 전송 및 수신 시 클라이언트 애플리케이션의 네트워크 트래픽에 대한 추가 세부 정보를 검토할 수 있습니다.

- NetworkReceiveThroughput – Aurora MySQL DB 클러스터의 각 DB 인스턴스가 클라이언트로부터 수신한 네트워크 처리량입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- NetworkTransmitThroughput – Aurora DB 클러스터의 각 인스턴스가 클라이언트로 전송한 네트워크 처리량(throughput)입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkReceiveThroughput – DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신한 네트워크 처리량(throughput)입니다.
- StorageNetworkTransmitThroughput – DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송한 네트워크 처리량(throughput)입니다.

이러한 지표를 모두 더해 네트워크 사용량을 DB 인스턴스 클래스 한도와 비교하는 방법을 평가합니다. 인스턴스 클래스 한도는 이러한 결합 지표를 합한 값보다 커야 합니다.

스토리지에 대한 네트워크 한도 및 CPU 사용량은 직접 관련이 있습니다. 네트워크 처리량이 증가하면 CPU 사용량도 증가합니다. CPU 및 네트워크 사용량을 모니터링하면 리소스가 소진되는 방식 및 이유에 대한 정보를 얻을 수 있습니다.

네트워크 사용량을 최소화하려면 다음을 고려하면 됩니다.

- 규모가 더 큰 DB 인스턴스 클래스를 사용합니다.
- 쓰기 요청을 일괄적으로 나누어 전체 트랜잭션을 줄입니다.
- 읽기 전용 워크로드를 읽기 전용 인스턴스로 지정합니다.
- 사용하지 않는 인덱스를 삭제합니다.

Amazon Aurora MySQL 데이터베이스 성능 문제 해결

이 주제에서는 몇 가지 일반적인 Aurora MySQL DB 성능 문제와 문제를 해결하거나 정보를 수집하여 이러한 문제를 신속하게 해결하는 방법을 중점적으로 다룹니다. 데이터베이스 성능은 다음 두 가지 범주로 나뉩니다.

- 서버 성능 - 전체 데이터베이스 서버가 느리게 실행됩니다.
- 쿼리 성능 - 하나 이상의 쿼리를 실행하는 데 시간이 더 오래 걸립니다.

AWS 모니터링 옵션

문제 해결에 도움이 되는 다음 AWS 모니터링 옵션을 사용하는 것이 좋습니다.

- Amazon CloudWatch – Amazon CloudWatch는 AWS 리소스와 AWS에서 실행 중인 애플리케이션을 실시간으로 모니터링합니다. CloudWatch를 사용하여 리소스 및 애플리케이션에 대해 측정할 수 있는 변수인 지표를 수집하고 추적할 수 있습니다. 자세한 내용은 Amazon CloudWatch란 무엇인가요?를 참조하세요.

  AWS Management Console에서 DB 인스턴스에 대한 모든 시스템 지표 및 프로세스 정보를 볼 수 있습니다. 일반, 느린, 감사, 오류 로그 데이터를 Amazon CloudWatch Logs의 로그 그룹에 게시하도록 Aurora MySQL DB 클러스터를 구성할 수 있습니다. 이를 통해 추세를 확인하고, 호스트가 영향을 받는 경우 로그를 유지 관리하고, 이상 또는 변경 사항을 쉽게 식별할 수 있도록 '정상' 성능에 대한 기준을 만들 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
- 향상된 모니터링 - Aurora MySQL 데이터베이스에 대해 Amazon CloudWatch 지표를 추가로 활성화하려면 향상된 모니터링을 켜세요. Aurora DB 클러스터를 생성하거나 수정할 때 향상된 모니터링 활성화를 선택합니다. 이를 통해 Aurora는 성능 지표를 CloudWatch에 게시할 수 있습니다. 사용 가능한 주요 지표로는 CPU 사용량, 데이터베이스 연결, 스토리지 사용량, 쿼리 지연 시간 등이 있습니다. 이러한 지표는 성능 병목 현상을 파악하는 데 도움이 됩니다.

  DB 인스턴스에 대해 전송되는 정보의 양은 확장 모니터링에 대해 정의된 세부 단위에 직접적으로 비례합니다. 모니터링 간격이 작을수록 OS 측정치가 더 자주 보고되고 모니터링 비용이 증가합니다. 비용을 관리하려면 AWS 계정의 여러 인스턴스에 대해 서로 다른 세부 단위를 설정합니다. 인스턴스 생성 시 기본 시간 세부 단위는 60초입니다. 자세한 내용은 Enhanced Monitoring 비용 단원을 참조하십시오.
- 성능 개선 도우미 - 모든 데이터베이스 직접 호출 지표를 볼 수 있습니다. 여기에는 DB 잠금, 대기, 처리된 행 수가 포함되며, 이 모든 정보를 문제 해결에 사용할 수 있습니다. Aurora DB 클러스터를 만

들거나 수정할 때 성능 개선 도우미 활성화를 선택합니다. 기본적으로 성능 개선 도우미의 데이터 보존 기간은 7일이지만 장기적인 성능 추세를 분석하도록 사용자 지정할 수 있습니다. 7일 넘게 보존하려면 유료 티어로 업그레이드해야 합니다. 자세한 내용은 성능 개선 도우미 요금을 참조하세요. 각 Aurora DB 인스턴스의 데이터 보존 기간을 개별적으로 설정할 수 있습니다. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하십시오.

Aurora MySQL 데이터베이스 성능 문제의 가장 일반적인 원인

다음 단계를 사용하여 Aurora MySQL 데이터베이스의 성능 문제를 해결할 수 있습니다. 이러한 단계는 논리적인 조사 순서로 나열되어 있지만 순서대로 따라야 하는 것은 아닙니다. 하나의 발견으로 여러 단계를 건너뛸 수 있으며 일련의 조사 경로가 가능해집니다.

1. 워크로드 - 데이터베이스 워크로드를 이해합니다.
2. 로깅 - 모든 데이터베이스 로그를 검토합니다.
3. 데이터베이스 연결 - 애플리케이션과 데이터베이스 간의 연결이 안정적인지 확인합니다.
4. 쿼리 성능 - 쿼리 실행 계획을 검토하여 변경되었는지 확인합니다. 코드 변경으로 인해 계획이 변경될 수 있습니다.

Aurora MySQL 데이터베이스의 워크로드 문제 해결

데이터베이스 워크로드는 읽기 및 쓰기로 볼 수 있습니다. '일반적인' 데이터베이스 워크로드를 이해하면 변화하는 수요에 맞춰 쿼리와 데이터베이스 서버를 튜닝할 수 있습니다. 성능이 변경될 수 있는 이유는 다양하므로 먼저 무엇이 변경되었는지 파악해야 합니다.

- 메이저 버전 또는 마이너 버전 업그레이드가 있었나요?

  메이저 버전 업그레이드에는 쿼리 실행 계획을 변경할 수 있는 엔진 코드, 특히 최적화 프로그램의 변경 사항이 포함됩니다. 데이터베이스 버전, 특히 메이저 버전을 업그레이드할 때는 데이터베이스 워크로드를 분석하고 그에 따라 튜닝하는 것이 매우 중요합니다. 튜닝에는 테스트 결과에 따라 쿼리를 최적화 및 재작성하거나 파라미터 설정을 추가 및 업데이트하는 작업이 포함될 수 있습니다. 영향을 미치는 원인을 이해하면 해당 영역에 집중할 수 있습니다.

  자세한 내용은 MySQL 설명서의 What is new in MySQL 8.0과 Server and status variables and options added, deprecated, or removed in MySQL 8.0 및 Aurora MySQL 버전 2와 Aurora MySQL 버전 3의 비교 섹션을 참조하세요.
- 처리 중인 데이터(행 수)가 증가했나요?
- 동시에 실행되는 쿼리가 더 많나요?
- 스키마 또는 데이터베이스 변경 사항이 있나요?
- 코드 결함이나 수정이 있었나요?

목차

인스턴스 호스트 지표

CPU, 메모리, 네트워크 활동과 같은 인스턴스 호스트 지표를 모니터링하면 워크로드 변경 여부를 이해하는 데 도움이 됩니다. 워크로드 변화를 이해하는 데는 두 가지 주요 개념이 있습니다.

- 사용률 - CPU 또는 디스크와 같은 디바이스의 사용량. 시간 기반 또는 용량 기반일 수 있습니다.
  - 시간 기반 - 특정 관찰 기간 동안 리소스가 많이 사용된 시간
  - 용량 기반 - 시스템 또는 구성 요소가 제공할 수 있는 처리량(용량의 백분율)
- 포화도 - 리소스에 처리할 수 있는 것보다 더 많은 작업이 필요한 정도. 용량 기준 사용량이 100%에 도달하면 추가 작업을 처리할 수 없으므로 대기열에 추가되어야 합니다.

CPU 사용량

다음 도구를 사용하여 CPU 사용량 및 포화도를 파악할 수 있습니다.

- CloudWatch는 `CPUUtilization` 지표를 제공합니다. 이 수치가 100%에 도달하면 인스턴스가 포화 상태입니다. 하지만 CloudWatch 지표는 1분 단위로 평균을 낸 값이며 세분성이 부족합니다.

  CloudWatch 지표에 대한 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표](#) 섹션을 참조하세요.
- 향상된 모니터링은 운영 체제 `top` 명령으로 반환된 지표를 제공합니다. 로드 평균과 다음 CPU 상태를 1초 단위로 보여줍니다.
  - `Idle (%)` = 유휴 시간
  - `IRQ (%)` = 소프트웨어 중단
  - `Nice (%)` = [Niced](#) 우선순위가 있는 프로세스의 Nice 시간
  - `Steal (%)` = 다른 테넌트에게 서비스를 제공하는 데 소요된 시간(가상화 관련)
  - `System (%)` = 시스템 시간
  - `User (%)` = 사용자 시간
  - `Wait (%)` = I/O 대기

  향상된 모니터링 지표에 대한 자세한 내용은 [Aurora의 지표](#) 섹션을 참조하세요.

메모리 사용량

시스템이 메모리 부족 상태이고 리소스 사용량이 포화 상태에 이르면 페이지 스캔, 페이징, 교체 및 메모리 부족 오류가 많이 발생합니다.

다음 도구를 사용하여 메모리 사용량 및 포화도를 파악할 수 있습니다.

CloudWatch는 일부 OS 캐시와 현재 사용 가능한 메모리를 비워서 회수할 수 있는 메모리 양을 보여주는 `FreeableMemory` 지표를 제공합니다.

CloudWatch 지표에 대한 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표](#) 섹션을 참조하세요.

향상된 모니터링은 메모리 사용 문제를 식별하는 데 도움이 되는 다음과 같은 지표를 제공합니다.

- `Buffers (KB)` - 스토리지 디바이스에 쓰기 전에 I/O 요청을 버퍼링하는 데 사용되는 메모리의 양 (KB)
- `Cached (KB)` - 파일 시스템 기반 I/O를 캐시하는 데 사용된 메모리의 양
- `Free (KB)` - 할당되지 않은 메모리의 양(KB)
- `Swap` - 캐시됨, 사용할 수 있음, 합계

예를 들어 DB 인스턴스에서 `Swap` 메모리를 사용하는 경우 워크로드의 총 메모리 용량은 현재 인스턴스에서 사용할 수 있는 양보다 큽니다. DB 인스턴스의 크기를 늘리거나 메모리 사용량을 줄이도록 워크로드를 튜닝하는 것이 좋습니다.

향상된 모니터링 지표에 대한 자세한 내용은 [Aurora의 지표](#) 섹션을 참조하세요.

성능 스키마 및 `sys` 스키마를 사용하여 메모리를 사용하는 연결 및 구성 요소를 확인하는 방법에 대한 자세한 내용은 [Aurora MySQL 데이터베이스의 메모리 사용량 문제 해결](#) 섹션을 참조하세요.

네트워크 처리량

CloudWatch는 총 네트워크 처리량에 대해 다음과 같은 지표를 제공하며, 모두 1분 단위로 평균을 낸 값입니다.

- `NetworkReceiveThroughput` - Aurora DB 클러스터의 각 인스턴스가 클라이언트에서 수신하는 네트워크 처리량
- `NetworkTransmitThroughput` - Aurora DB 클러스터의 각 인스턴스가 클라이언트로 전송하는 네트워크 처리량
- `NetworkThroughput` - Aurora DB 클러스터의 각 인스턴스가 클라이언트에서 수신하고 클라이언트로 전송하는 네트워크 처리량
- `StorageNetworkReceiveThroughput` - DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신하는 네트워크 처리량

- StorageNetworkTransmitThroughput - Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송하는 네트워크 처리량
- StorageNetworkThroughput - Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템과 송수신하는 네트워크 처리량

CloudWatch 지표에 대한 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 섹션을 참조하세요.

향상된 모니터링은 network 수신(RX) 및 전송(TX) 그래프를 최대 1초의 세부 단위로 제공합니다.

향상된 모니터링 지표에 대한 자세한 내용은 Aurora의 지표 섹션을 참조하세요.

데이터베이스 지표

워크로드 변화에 대한 다음 CloudWatch 지표를 살펴보세요.

- BlockedTransactions - 데이터베이스에서 1초마다 차단되는 평균 트랜잭션 수
- BufferCacheHitRatio - 버퍼 캐시에서 처리하는 요청 비율
- CommitThroughput - 초당 커밋 작업의 평균 수
- DatabaseConnections - 데이터베이스 인스턴스에 대한 클라이언트 네트워크 연결 수
- Deadlocks - 데이터베이스 1초마다 발생하는 평균 교착 수
- DMLThroughput - 초당 평균 삽입, 업데이트 및 삭제 수
- ResultSetCacheHitRatio - 쿼리 캐시에서 처리하는 요청 비율
- RollbackSegmentHistoryListLength - 삭제 표시 레코드로 커밋된 트랜잭션을 기록하는 실행 취소 로그
- RowLockTime - InnoDB 테이블에 대한 행 잠금을 획득하는 데 걸린 총 시간
- SelectThroughput - 초당 평균 선택 쿼리 수

CloudWatch 지표에 대한 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 섹션을 참조하세요.

워크로드를 검토할 때는 다음 질문을 고려해 보세요.

1. 인스턴스 크기를 8xlarge에서 4xlarge로 줄이거나 db.r5에서 db.r6으로 변경하는 등 최근에 DB 인스턴스 클래스를 변경했나요?

2. 클론을 생성하여 문제를 재현할 수 있나요? 아니면 해당 인스턴스에서만 발생하나요?
3. 서버 리소스 고갈, 높은 CPU 또는 메모리 고갈이 있나요? 그렇다면 추가 하드웨어가 필요할 수 있습니다.
4. 하나 이상의 쿼리가 더 오래 걸리나요?
5. 변경 사항이 업그레이드, 특히 메이저 버전 업그레이드로 인해 발생했나요? 그렇다면 업그레이드 전후 지표를 비교하세요.
6. 리더 DB 인스턴스 수에 변화가 있나요?
7. 일반, 감사 또는 바이너리 로깅을 활성화했나요? 자세한 내용은 Aurora MySQL 데이터베이스에 대한 로깅 단원을 참조하십시오.
8. 바이너리 로그(binlog) 복제 사용을 활성화, 비활성화 또는 변경나요?
9. 다수의 행 잠금이 있는 장기 실행 트랜잭션이 있나요? InnoDB 기록 목록 길이(HLL)에서 장기 실행 트랜잭션의 징후를 확인하세요.

   자세한 내용은 InnoDB 기록 목록 길이가 크게 늘어남 및 Amazon Aurora MySQL DB 클러스터에서 SELECT 쿼리가 느리게 실행되는 이유는 무엇인가요? 블로그 게시물을 참조하세요.

   a. 쓰기 트랜잭션으로 인해 큰 HLL이 발생한 경우 UNDO 로그가 누적되고 있다는 의미입니다(정기적으로 정리되지 않음). 대규모 쓰기 트랜잭션의 경우 이 누적이 빠르게 증가할 수 있습니다. MySQL에서는 UNDO가 SYSTEM 테이블스페이스에 저장됩니다. SYSTEM 테이블스페이스는 축소할 수 없습니다. UNDO 로그로 인해 SYSTEM 테이블스페이스가 몇 GB, 심지어는 몇 TB까지 증가할 수 있습니다. 삭제 후에는 데이터를 논리적으로 백업(덤프)하여 할당된 공간을 해제한 다음 덤프를 새 DB 인스턴스로 가져오세요.
   b. 읽기 트랜잭션(장기 실행 쿼리)으로 인해 큰 HLL이 발생하는 경우 쿼리가 많은 양의 임시 공간을 사용하고 있다는 의미일 수 있습니다. 재부팅하여 임시 공간을 해제하세요. 성능 개선 도우미 DB 지표를 검토하여 Temp 섹션의 변경 사항(예: `created_tmp_tables`)이 있는지 확인하세요. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하십시오.
10. 장기 실행 트랜잭션을 더 적은 행을 수정하는 더 작은 트랜잭션으로 분할할 수 있나요?
11. 차단된 트랜잭션에 변경 사항이 있거나 교착 상태가 증가했나요? 성능 개선 도우미 DB 지표를 검토하여 Locks 섹션의 상태 변수 변경 사항(예: `innodb_row_lock_time`, `innodb_row_lock_waits`, `innodb_dead_locks`)이 있는지 확인하세요. 1분 또는 5분의 간격을 사용하세요.
12. 대기 이벤트가 늘어났나요? 1분 또는 5분 간격을 사용하여 성능 개선 도우미의 대기 이벤트와 대기 유형을 검사하세요. 상위 대기 이벤트를 분석하고 이러한 이벤트가 워크로드 변경 또는 데이터베이스 경합과 상관관계가 있는지 확인하세요. 예를 들어, `buf_pool mutex`는 버퍼 풀 경합을 나타냅니다. 자세한 내용은 대기 이벤트로 Aurora MySQL 튜닝 단원을 참조하십시오.

Aurora MySQL 데이터베이스의 메모리 사용량 문제 해결

CloudWatch, 향상된 모니터링 및 성능 개선 도우미는 운영 체제 수준에서 데이터베이스 프로세스에서 사용하는 메모리 양 등의 메모리 사용량에 대한 유용한 개요를 제공하지만, 이러한 메모리 사용량을 유발할 수 있는 엔진 내 연결 또는 구성 요소를 분석할 수는 없습니다.

이 문제를 해결하기 위해 성능 스키마와 sys 스키마를 사용할 수 있습니다. Aurora MySQL 버전 3에서 메모리 계측은 성능 스키마가 사용 설정되면 기본적으로 활성화됩니다. Aurora MySQL 버전 2에서는 성능 스키마 메모리 사용량의 메모리 계측만 기본적으로 사용 설정됩니다. 성능 스키마에서 메모리 사용량을 추적하고 성능 스키마 메모리 계측을 사용 설정하는 데 사용할 수 있는 테이블에 대한 자세한 내용은 MySQL 설명서의 메모리 요약 테이블을 참조하시기 바랍니다. 성능 개선 도우미에서 성능 스키마를 사용하는 것에 대한 자세한 내용은 Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요 섹션을 참조하세요.

성능 스키마에서 현재 메모리 사용량을 추적할 수 있는 자세한 정보가 나와 있지만, MySQL sys 스키마에는 메모리 사용 위치를 빠르게 찾아내는 데 사용할 수 있는 성능 스키마 테이블 상단에 뷰가 제공됩니다.

sys 스키마에서 다음 뷰를 사용하여 연결, 구성 요소 및 쿼리별로 메모리 사용량을 추적할 수 있습니다.

보기	설명
memory_by_host_by_current_bytes	호스트별 엔진 메모리 사용량에 대한 정보를 제공합니다. 이 정보를 통해 메모리를 사용하는 애플리케이션 서버 또는 클라이언트 호스트를 식별할 수 있습니다.
memory_by_thread_by_current_bytes	스레드 ID별 엔진 메모리 사용량에 대한 정보를 제공합니다. MySQL의 스레드 ID는 클라이언트 연결 또는 백그라운드 스레드일 수 있습니다. sys.processlist 뷰 또는 performance_schema.threads 테이블을 사용하여 스레드 ID를 MySQL 연결 ID에 매핑할 수 있습니다.
memory_by_user_by_current_bytes	사용자별 엔진 메모리 사용량에 대한 정보를 제공합니다. 이 정보를 통해 메모리를 사용하는 사용자 계정 또는 클라이언트를 식별할 수 있습니다.

보기	설명
memory_global_by_current_bytes	엔진 구성 요소별 엔진 메모리 사용량에 대한 정보를 제공합니다. 이 정보를 통해 엔진 버퍼 또는 구성 요소별 메모리 사용량을 전체적으로 식별할 수 있습니다. 예를 들어 InnoDB 버퍼 풀의 `memory/innodb/buf_buf_pool` 이벤트 또는 준비된 문에 대한 `memory/sql/Prepared_statement::main_mem_root` 이벤트를 볼 수 있습니다.
memory_global_total	데이터베이스 엔진에서 추적된 총 메모리 사용량에 대한 개요를 제공합니다.

Aurora MySQL 버전 3.05 이상에서는 성능 스키마 문 요약 테이블에서 문 다이제스트를 통해 최대 메모리 사용량을 추적할 수도 있습니다. 문 요약 테이블에는 정규화된 문 요약과 실행에 대한 집계된 통계가 포함되어 있습니다. 이 `MAX_TOTAL_MEMORY` 열을 통해 통계가 마지막으로 재설정된 이후 또는 데이터베이스 인스턴스가 재시작된 이후 쿼리 다이제스트에서 사용된 최대 메모리를 식별할 수 있습니다. 이를 통해 메모리를 많이 사용할 수 있는 특정 쿼리를 식별할 수 있습니다.

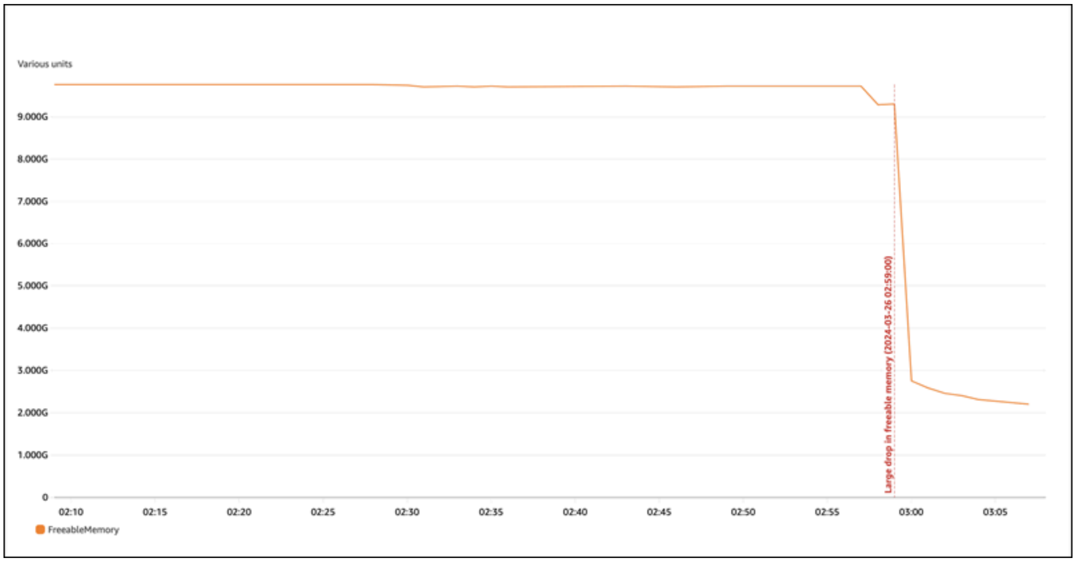
> **Note**
>
> 성능 스키마와 `sys` 스키마는 서버의 현재 메모리 사용량과 연결 및 엔진 구성 요소당 사용된 메모리의 상한선을 보여줍니다. 성능 스키마는 메모리에 유지 관리되므로 DB 인스턴스가 다시 시작될 때 정보가 재설정됩니다. 시간이 지나도 기록을 계속해서 유지하려면 성능 스키마 외부에서 이 데이터를 검색 및 저장하도록 구성하는 것이 좋습니다.

주제

- 예제 1: 지속적인 높은 메모리 사용량
- 예제 2: 일시적인 메모리 급증
- 예시 3: 여유 메모리가 지속적으로 삭제되고 회수되지 않음

예제 1: 지속적인 높은 메모리 사용량

전반적으로 CloudWatch에서 `FreeableMemory`를 살펴보면 2024-03-26 02:59 UTC를 기준으로 메모리 사용량이 크게 증가한 것을 확인할 수 있습니다.



이것이 전부는 아닙니다. 어떤 구성 요소가 메모리를 가장 많이 사용하고 있는지 확인하려면 데이터베이스에 로그인하여 `sys.memory_global_by_current_bytes`를 살펴보시기 바랍니다. 이 표에는 MySQL이 추적하는 메모리 이벤트 목록과 이벤트별 메모리 할당 정보가 포함되어 있습니다. 각 메모리 추적 이벤트는 `memory/%`로 시작되며 이벤트와 관련된 엔진 구성 요소/특성에 대한 기타 정보가 뒤따릅니다.

예를 들어 `memory/performance_schema/%`는 성능 스키마와 관련된 메모리 이벤트용이고 `memory/innodb/%`는 InnoDB용 등입니다. 이벤트 이름 지정 규칙에 대한 자세한 내용은 MySQL 설명서의 [성능 스키마 계측 이름 지정 규칙]을 참조하세요.

다음 쿼리에서는 `current_alloc`을 기반으로 유력한 원인을 찾을 수 있지만 많은 `memory/performance_schema/%` 이벤트도 확인할 수 있습니다.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-----------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                 |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
```

```
+-------------------------------------------------------------------------------
+----------------+----------------+--------------------+-------------+------------
+----------------+
| memory/sql/Prepared_statement::main_mem_root                                  |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB    | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                      |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB        |
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error           |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB       |
| memory/performance_schema/memory_summary_by_thread_by_event_name             |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB        |
| memory/innodb/memory                                                          |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB        |
| memory/sql/String::value                                                      |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes      |
| memory/sql/TABLE                                                              |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB        |
+-------------------------------------------------------------------------------
+----------------+----------------+--------------------+-------------+------------
+----------------+
10 rows in set (0.02 sec)
```

앞서 설명했듯이 성능 스키마는 메모리에 저장됩니다. 즉, performance_schema 메모리 계층에서 추적할 수도 있습니다.

> **Note**
>
> 성능 스키마가 메모리를 많이 사용하고 있고 메모리 사용량을 제한하려는 경우 요구 사항에 따라 데이터베이스 파라미터를 조정할 수 있습니다. 자세한 내용은 MySQL 설명서의 성능 스키마 메모리 할당 모델을 참조하세요.

가독성을 위해 동일한 쿼리를 다시 실행하되 성능 스키마 이벤트는 제외할 수 있습니다. 출력은 다음과 같이 표시됩니다.

- memory/sql/Prepared_statement::main_mem_root에서 주로 메모리가 사용됩니다.
- current_alloc 열에는 MySQL이 현재 이 이벤트에 4.91GiB를 할당하고 있음이 나와 있습니다.
- high_alloc column에는 통계가 마지막으로 재설정된 이후 또는 서버가 다시 시작된 이후 4.91GiB가 최고 current_alloc 수치임이 나와 있습니다. 즉, memory/sql/Prepared_statement::main_mem_root가 현재 해당 최대치에 있다는 의미입니다.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB          |      512823 | 4.91 GiB    | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB          |           4 | 79.07 MiB   | 19.77 MiB      |
| memory/innodb/memory                          |         17096 | 31.68 MiB     | 1.90
 KiB           |       22498 | 37.60 MiB   | 1.71 KiB       |
| memory/sql/String::value                      |        122277 | 27.94 MiB     |  239
 bytes         |      124699 | 29.47 MiB   |  247 bytes     |
| memory/sql/TABLE                              |          9927 | 24.67 MiB     | 2.55
 KiB           |        9929 | 24.68 MiB   | 2.55 KiB       |
| memory/innodb/lock0lock                       |          8888 | 19.71 MiB     | 2.27
 KiB           |        8888 | 19.71 MiB   | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes         |      257631 | 16.24 MiB   |   66 bytes     |
| memory/mysys/KEY_CACHE                        |             3 | 16.00 MiB     | 5.33
 MiB           |           3 | 16.00 MiB   | 5.33 MiB       |
| memory/innodb/sync0arr                        |             3 | 7.03 MiB      | 2.34
 MiB           |           3 | 7.03 MiB    | 2.34 MiB       |
| memory/sql/THD::main_mem_root                 |           815 | 6.56 MiB      | 8.24
 KiB           |         849 | 7.19 MiB    | 8.67 KiB       |
+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
10 rows in set (0.06 sec)
```

이벤트 이름을 보면 이 메모리가 준비된 문에 사용되고 있음을 알 수 있습니다. 어떤 연결이 이 메모리를 사용하고 있는지 알아보려면 memory_by_thread_by_current_bytes를 확인하시기 바랍니다.

다음 예제에서 각 연결에는 약 7MiB가 할당되어 있으며 최고 수치는 약 6.29MiB(current_max_alloc)입니다. 이 예제에서는 준비된 문을 포함하는 80개의 테이블과 800개의 연결을 가진 sysbench를 사용하고 있으므로 납득 가능한 수치입니다. 이 시나리오에서 메모리 사용량을 줄이려면 준비된 문의 애플리케이션 사용량을 최적화하여 메모리 사용량을 줄일 수 있습니다.

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+--------------------------------------------+--------------------
+-------------------+-------------------+-------------------+-----------------+
| thread_id | user                                       | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+--------------------------------------------+--------------------
+-------------------+-------------------+-------------------+-----------------+
|        46 | rdsadmin@localhost                         |                405 | 8.47 MiB
          | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                          |               1749 | 6.72 MiB
          | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
|       101 | reinvent@10.0.4.4                          |               1845 | 6.71 MiB
          | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
|        55 | reinvent@10.0.4.4                          |               1674 | 6.68 MiB
          | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                          |               1416 | 6.66 MiB
          | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                          |               1759 | 6.66 MiB
          | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                          |               1428 | 6.64 MiB
          | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                          |               1389 | 6.62 MiB
          | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
|       116 | reinvent@10.0.4.4                          |               1333 | 6.61 MiB
          | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                          |               1448 | 6.59 MiB
          | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
|        98 | reinvent@10.0.4.4                          |               1440 | 6.57 MiB
          | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                          |               1433 | 6.57 MiB
          | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
|        62 | reinvent@10.0.4.4                          |               1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                          |               1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                          |               1346 | 6.54 MiB
          | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
```

```
|       105 | reinvent@10.0.4.4                       |             1347 | 6.54 MiB
          | 4.97 KiB           | 6.29 MiB           | 13.34 MiB        |
|        73 | reinvent@10.0.4.4                       |             1335 | 6.54 MiB
          | 5.02 KiB           | 6.29 MiB           | 13.23 MiB        |
|        54 | reinvent@10.0.4.4                       |             1510 | 6.53 MiB
          | 4.43 KiB           | 6.29 MiB           | 13.49 MiB        |
.
                                                                         .
.
                                                                         .
.
                                                                         .
|       812 | reinvent@10.0.4.4                       |             1259 | 6.38 MiB
          | 5.19 KiB           | 6.29 MiB           | 13.05 MiB        |
|       214 | reinvent@10.0.4.4                       |             1279 | 6.38 MiB
          | 5.10 KiB           | 6.29 MiB           | 12.90 MiB        |
|       325 | reinvent@10.0.4.4                       |             1254 | 6.38 MiB
          | 5.21 KiB           | 6.29 MiB           | 12.99 MiB        |
|       705 | reinvent@10.0.4.4                       |             1273 | 6.37 MiB
          | 5.13 KiB           | 6.29 MiB           | 13.03 MiB        |
|       530 | reinvent@10.0.4.4                       |             1268 | 6.37 MiB
          | 5.15 KiB           | 6.29 MiB           | 12.92 MiB        |
|       307 | reinvent@10.0.4.4                       |             1263 | 6.37 MiB
          | 5.17 KiB           | 6.29 MiB           | 12.87 MiB        |
|       738 | reinvent@10.0.4.4                       |             1260 | 6.37 MiB
          | 5.18 KiB           | 6.29 MiB           | 13.00 MiB        |
|       819 | reinvent@10.0.4.4                       |             1252 | 6.37 MiB
          | 5.21 KiB           | 6.29 MiB           | 13.01 MiB        |
|        31 | innodb/srv_purge_thread                 |            17810 | 3.14 MiB
          |  184 bytes         | 2.40 MiB           | 205.69 MiB       |
|        38 | rdsadmin@localhost                      |              599 | 1.76 MiB
          | 3.01 KiB           | 1.00 MiB           | 25.58 MiB        |
|         1 | sql/main                                |             3756 | 1.32 MiB
          |  367 bytes         | 355.78 KiB         | 6.19 MiB         |
|       854 | rdsadmin@localhost                      |               46 | 1.08 MiB
          | 23.98 KiB          | 1.00 MiB           | 5.10 MiB         |
|        30 | innodb/clone_gtid_thread                |             1596 | 573.14
 KiB        |  367 bytes         | 254.91 KiB         | 970.69 KiB       |
|        40 | rdsadmin@localhost                      |              235 | 245.19
 KiB        | 1.04 KiB           | 128.88 KiB         | 808.64 KiB       |
|       853 | rdsadmin@localhost                      |               96 | 94.63
 KiB         | 1009 bytes         | 29.73 KiB          | 422.45 KiB       |
|        36 | rdsadmin@localhost                      |               33 | 36.29
 KiB         | 1.10 KiB           | 16.08 KiB          | 74.15 MiB        |
```

```
|        33 | sql/event_scheduler                          |                 3 | 16.27
 KiB        | 5.42 KiB           | 16.04 KiB          | 16.27 KiB          |
|        35 | sql/compress_gtid_table                      |                 8 | 14.20
 KiB        | 1.77 KiB           | 8.05 KiB           | 18.62 KiB          |
|        25 | innodb/fts_optimize_thread                   |                12 | 1.86 KiB
         |  158 bytes          |  648 bytes          | 1.98 KiB           |
|        23 | innodb/srv_master_thread                     |                11 | 1.23 KiB
         |  114 bytes          |  361 bytes          | 24.40 KiB          |
|        24 | innodb/dict_stats_thread                     |                11 | 1.23 KiB
         |  114 bytes          |  361 bytes          | 1.35 KiB           |
|         5 | innodb/io_read_thread                        |                 1 |  144
 bytes         |  144 bytes         |  144 bytes         |  144 bytes         |
|         6 | innodb/io_read_thread                        |                 1 |  144
 bytes         |  144 bytes         |  144 bytes         |  144 bytes         |
|         2 | sql/aws_oscar_log_level_monitor              |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|         4 | innodb/io_ibuf_thread                        |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|         7 | innodb/io_write_thread                       |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|         8 | innodb/io_write_thread                       |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|         9 | innodb/io_write_thread                       |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        10 | innodb/io_write_thread                       |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        11 | innodb/srv_lra_thread                        |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        12 | innodb/srv_akp_thread                        |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        18 | innodb/srv_lock_timeout_thread               |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |  248 bytes         |
|        19 | innodb/srv_error_monitor_thread              |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        20 | innodb/srv_monitor_thread                    |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        21 | innodb/buf_resize_thread                     |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        22 | innodb/btr_search_sys_toggle_thread          |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        32 | innodb/dict_persist_metadata_table_thread    |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
|        34 | sql/signal_handler                           |                 0 |    0
 bytes         |    0 bytes         |    0 bytes         |    0 bytes         |
```

```
+-----------+---------------------------------------------+--------------------
+-------------------+-------------------+-------------------+-----------------+
831 rows in set (2.48 sec)
```

앞서 설명한 것처럼 여기서 스레드 ID(thd_id) 값은 서버 백그라운드 스레드 또는 데이터베이스 연결을 참조할 수 있습니다. 스레드 ID 값을 데이터베이스 연결 ID에 매핑하려면 `performance_schema.threads` 테이블 또는 `sys.processlist` 뷰를 사용할 수 있으며, 연결 ID는 `conn_id`입니다.

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+-------------------+----------+---------+-----------------+------
+-------------------------------------------------+
| thd_id | conn_id | user              | db       | command | state           | time |
 last_wait                                       |
+--------+---------+-------------------+----------+---------+-----------------+------
+-------------------------------------------------+
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL            |    0 |
 idle                                            |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating        |    0 |
 wait/io/table/sql/handler                       |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating        |    0 |
 wait/io/table/sql/handler                       |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL            |    0 |
 idle                                            |
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL            |    0 |
 idle                                            |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating        |    0 |
 wait/io/table/sql/handler                       |
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating        |    0 |
 wait/io/table/sql/handler                       |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
.
                                                 .
```

```
.
                                                      .
.
                                                      .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                                |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                           |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                              |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                           |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                         |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection                |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                                |
+--------+---------+-------------------+----------+---------+----------------+------
+--------------------------------------------------+
804 rows in set (5.51 sec)
```

이제 sysbench 워크로드를 중지하여 연결을 닫고 메모리를 해제합니다. 이벤트를 다시 확인해 보면 메모리가 해제된 것을 확인할 수 있지만, high_alloc을 통해 최고 수치를 계속 확인할 수 있습니다. high_alloc 열은 현재 할당된 메모리만 보여주는 current_alloc의 메모리 사용량을 즉시 식별하지 못할 수도 있는 단기 메모리 사용량 급증을 식별하는 데 매우 유용할 수 있습니다.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+-------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+-------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB          |     512823 | 4.91 GiB   | 10.04 KiB      |
+-----------------------------------------------+---------------+---------------
+-------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

high_alloc을 재설정하려는 경우 performance_schema 메모리 요약 테이블을 잘라낼 수 있지만 이렇게 하면 모든 메모리 계측이 재설정됩니다. 자세한 내용은 MySQL 설명서의 성능 스키마 일반 테이블 특성을 참조하세요.

다음 예제에서는 잘린 후 high_alloc이 재설정되는 것을 볼 수 있습니다.

```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |          17 | 253.80 KiB | 14.93 KiB      |
+-----------------------------------------------+---------------+---------------
+-------------------+------------+------------+----------------+
1 row in set (0.00 sec)
```

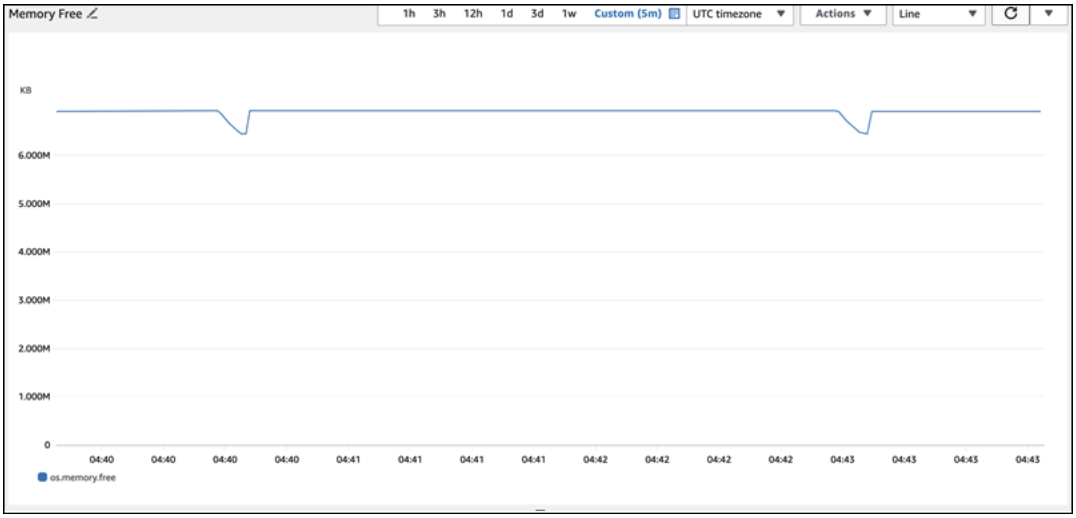
예제 2: 일시적인 메모리 급증

또 다른 일반적인 현상은 데이터베이스 서버의 메모리 사용량이 일시적으로 급증하는 것입니다. 사용 가능한 메모리가 주기적으로 떨어지면 메모리가 이미 비워졌기 때문에 sys.memory_global_by_current_bytes에서 current_alloc을 사용할 때 문제를 해결하기가 어려울 수 있습니다.

Note

성능 스키마 통계를 재설정하거나 데이터베이스 인스턴스를 다시 시작한 경우 sys 또는 performance_schema에서 이 정보를 사용할 수 없습니다. 이 정보를 유지하려면 외부 지표 수집을 구성하는 것이 좋습니다.

향상된 모니터링의 다음 os.memory.free 지표 그래프는 7초간의 단기 메모리 사용량 급증을 보여줍니다. 향상된 모니터링을 통해 1초 정도의 짧은 간격으로 모니터링할 수 있어 이와 같은 일시적 급증을 포착하는 데 적합합니다.



여기서 메모리 사용량의 원인을 진단하는 데 도움이 되도록 sys 메모리 요약 뷰의 high_alloc과 성능 스키마 문 요약 테이블을 함께 사용하여 문제가 되는 세션 및 연결을 식별할 수 있습니다.

예상한 바와 같이 현재 메모리 사용량이 많지 않으므로 current_alloc의 sys 스키마 뷰에서 주요 문제 인자를 확인할 수 없습니다.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+-----------------+
| event_name                                                                   |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+-----------------+
| memory/innodb/hash0hash                                                      |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB   | 19.77 MiB      |
| memory/innodb/os0event                                                       |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB   |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB   | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                       |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB   | 5.33 MiB       |
| memory/performance_schema/events_statements_history_long                     |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB   | 14.34 MiB      |
```

```
| memory/performance_schema/events_errors_summary_by_thread_by_error           |
   257 | 13.07 MiB      | 52.06 KiB         |         257 | 13.07 MiB   | 52.06 KiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB      | 11.81 MiB         |           1 | 11.81 MiB   | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    |
     1 | 9.77 MiB       | 9.77 MiB          |           1 | 9.77 MiB    | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.digest_text         |
     1 | 9.77 MiB       | 9.77 MiB          |           1 | 9.77 MiB    | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text            |
     1 | 9.77 MiB       | 9.77 MiB          |           1 | 9.77 MiB    | 9.77 MiB        |
+-----------------------------------------------------------------------------
+---------------+---------------+------------------+------------+------------
+-----------------+
10 rows in set (0.01 sec)
```

뷰를 high_alloc 기준으로 정렬하여 펼치면 이제 memory/temptable/physical_ram 구성 요소가 매우 적합한 후보임을 알 수 있습니다. 최대 용량으로는 515.00MiB를 소비했습니다.

이름에서 알 수 있듯이 memory/temptable/physical_ram은 MySQL 8.0에 도입된 MySQL의 TEMP 스토리지 엔진의 메모리 사용량을 측정합니다. MySQL에서 임시 테이블을 사용하는 방법에 대한 자세한 내용은 MySQL 설명서의 MySQL의 내부 임시 테이블 사용을 참조하세요.

> **Note**
>
> 이 예제에서는 sys.x$memory_global_by_current_bytes 뷰를 사용합니다.

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+-----------------------------------------------------------------------------
+---------------------+-----------------+
| event_name                                                                   |
 currently allocated | high-water mark |
+-----------------------------------------------------------------------------
+---------------------+-----------------+
| memory/temptable/physical_ram                                                | 4.00
 MiB                | 515.00 MiB      |
| memory/innodb/hash0hash                                                      | 79.07
 MiB              | 79.07 MiB       |
```

```
| memory/innodb/os0event                                               | 63.95
 MiB            | 63.95 MiB        |
| memory/performance_schema/events_statements_summary_by_digest        | 40.28
 MiB            | 40.28 MiB        |
| memory/mysys/KEY_CACHE                                               | 16.00
 MiB            | 16.00 MiB        |
| memory/performance_schema/events_statements_history_long             | 14.34
 MiB            | 14.34 MiB        |
| memory/performance_schema/events_errors_summary_by_thread_by_error   | 13.07
 MiB            | 13.07 MiB        |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB            | 11.81 MiB        |
| memory/performance_schema/events_statements_summary_by_digest.digest_text   | 9.77
 MiB             | 9.77 MiB         |
| memory/performance_schema/events_statements_history_long.sql_text           | 9.77
 MiB             | 9.77 MiB         |
+-----------------------------------------------------------------------------
+---------------------+-----------------+
10 rows in set (0.00 sec)
```

예제 1: 지속적인 높은 메모리 사용량에서는 각 연결의 현재 메모리 사용량을 확인하여 문제의 메모리 사용을 담당하는 연결을 확인했습니다. 이 예제에서는 메모리가 이미 비워졌으므로 현재 연결의 메모리 사용량을 확인하는 것은 유용하지 않습니다.

더 자세히 알아보고 문제가 되는 문, 사용자, 호스트를 찾기 위해 성능 스키마를 사용합니다. 성능 스키마에는 이벤트 이름, 문 다이제스트, 호스트, 스레드, 사용자 등 다양한 차원으로 구분된 여러 문 요약 테이블이 포함되어 있습니다. 각 뷰를 통해 특정 문이 실행되는 위치와 수행하는 작업을 더 자세히 살펴볼 수 있습니다. 이 섹션은 MAX_TOTAL_MEMORY를 중점적으로 다루지만, 사용 가능한 모든 열에 대한 자세한 내용은 성능 스키마 문 요약 테이블 설명서에서 확인할 수 있습니다.

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+------------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+------------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
| events_statements_summary_by_digest                        |
| events_statements_summary_by_host_by_event_name            |
| events_statements_summary_by_program                       |
| events_statements_summary_by_thread_by_event_name          |
| events_statements_summary_by_user_by_event_name            |
| events_statements_summary_global_by_event_name             |
```

```
+--------------------------------------------------------------+
7 rows in set (0.00 sec)
```

먼저 events_statements_summary_by_digest를 확인해 MAX_TOTAL_MEMORY를 살펴봅니다.

여기서 다음 내용을 확인할 수 있습니다.

- 다이제스트 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a가 포함된 쿼리가 이 메모리 사용량에 적합한 것으로 보입니다. MAX_TOTAL_MEMORY는 5537450710으로, sys.x$memory_global_by_current_bytes의 memory/temptable/physical_ram 이벤트에서 살펴본 최고 수치와 일치합니다.
- 이는 4번(COUNT_STAR) 실행되었는데, 처음은 2024-03-26 04:08:34.943256에, 마지막은 2024-03-26 04:43:06.998310에 실행되었습니다.

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+------------------------------------------------------------------
+------------+------------------+----------------------------
+----------------------------+
| SCHEMA_NAME | DIGEST                                                           |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                 | LAST_SEEN
  |
+-------------+------------------------------------------------------------------
+------------+------------------+----------------------------
+----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |         537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |           3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |           3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |           3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |           3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+------------------------------------------------------------------
+------------+------------------+----------------------------
+----------------------------+
```

```
5 rows in set (0.00 sec)
```

이제 문제가 되는 다이제스트를 알았으므로 쿼리 텍스트, 실행한 사용자, 실행 위치 등의 세부 정보를 더 자세히 확인할 수 있습니다. 반환된 다이제스트 텍스트를 바탕으로 이는 임시 테이블 4개를 만들고 테이블 스캔을 4회 수행하는 일반적인 테이블 표현식(CTE)이며, 매우 비효율적이라는 것을 알 수 있습니다.

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;

*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

events_statements_summary_by_digest 테이블 및 기타 성능 스키마 문 요약 테이블에 대한 자세한 내용은 MySQL 설명서의 [문 요약 테이블](#)을 참조하세요.

[EXPLAIN](#) 또는 [EXPLAIN ANALYZE](#) 문을 실행하여 자세한 내용을 확인할 수도 있습니다.

> **Note**
>
> EXPLAIN ANALYZE는 EXPLAIN보다 많은 정보를 제공할 수 있지만 쿼리를 실행하기도 하므로 주의해야 합니다.

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;
```

```
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

하지만 누가 실행했을까요? 성능 스키마에서 `destructive_operator` 사용자가 537450710의 `MAX_TOTAL_MEMORY`를 보유한 것을 확인할 수 있는데, 이 역시 이전 결과와 일치합니다.

> **Note**
> 성능 스키마는 메모리에 저장되므로 감사의 유일한 소스로 사용해서는 안 됩니다. 문의 실행 내역과 사용자 기록을 관리해야 하는 경우 [Aurora 고급 감사]를 사용 설정하는 것이 좋습니다. 메모리 사용량에 대한 정보도 유지해야 하는 경우 이러한 값을 내보내고 저장하도록 모니터링을 구성하는 것이 좋습니다.

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+-------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+-------------------+
| destructive_operator | statement/sql/select      |          4 |         537450710 |
| rdsadmin             | statement/sql/select      |       4172 |           3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |           3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |           3459965 |
| rdsadmin             | statement/sql/show_status |         75 |           1914976 |
+----------------------+---------------------------+------------+-------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

예시 3: 여유 메모리가 지속적으로 삭제되고 회수되지 않음

InnoDB 데이터베이스 엔진은 다양한 구성 요소에 대해 다양한 특수 메모리 추적 이벤트를 사용합니다. 이러한 특정 이벤트를 통해 주요 InnoDB 하위 시스템의 메모리 사용량을 세부적으로 추적할 수 있습니다. 예를 들면 다음과 같습니다.

- `memory/innodb/buf0buf` - InnoDB 버퍼 풀에 대한 메모리 할당 모니터링을 전담합니다.
- `memory/innodb/ibuf0ibuf` - 특히 InnoDB 변경 버퍼와 관련된 메모리 변경 사항을 추적합니다.

메모리의 상위 소비자를 식별하기 위해 `sys.memory_global_by_current_bytes`를 쿼리할 수 있습니다.

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.28 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

결과는 `memory/innodb/memory`가 현재 할당된 메모리의 5.28GiB를 사용하는 상위 소비자임을 보여줍니다. 이 이벤트는 앞에서 언급한 `memory/innodb/buf0buf`와 같이 보다 구체적인 대기 이벤트와 관련이 없는 다양한 InnoDB 구성 요소에서 메모리 할당을 위한 범주 역할을 합니다.

InnoDB 구성 요소가 메모리의 기본 소비자임을 확인한 후 다음 MySQL 명령을 사용하여 세부 정보를 자세히 살펴볼 수 있습니다.

```
SHOW ENGINE INNODB STATUS \G;
```

SHOW ENGINE INNODB STATUS 명령은 다양한 InnoDB 구성 요소에 대한 자세한 메모리 사용량 통계를 포함하여 InnoDB 스토리지 엔진에 대한 포괄적인 상태 보고서를 제공합니다. 이를 통해 가장 많은 메모리를 소비하는 특정 InnoDB 구조 또는 작업을 식별할 수 있습니다. 자세한 내용은 MySQL 설명서에서 InnoDB in-memory structures를 참조하세요.

InnoDB 상태 보고서의 `BUFFER POOL AND MEMORY` 섹션을 분석하면 `memory/innodb/memory`가 추적하는 메모리의 89%를 차지하는 5,051,647,748바이트(4.7GiB)가 딕셔너리 객체 캐시에 할당되어 있는 것을 확인할 수 있습니다.

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5051647748
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

딕셔너리 객체 캐시는 이전에 액세스한 데이터 딕셔너리 객체를 메모리에 저장하여 객체 재사용을 가능하게 하고 성능을 개선하는 공유 글로벌 캐시입니다. 딕셔너리 객체 캐시에 메모리가 많이 할당되어 있다는 것은 데이터 딕셔너리 캐시에 데이터베이스 객체가 많다는 것을 의미합니다.

데이터 딕셔너리 캐시가 주된 소비자임을 알았으므로 이제 데이터 딕셔너리 캐시에서 열린 테이블을 검사합니다. 테이블 정의 캐시에서 테이블 수를 찾으려면 글로벌 상태 변수 open_table_definitions을 쿼리합니다.

```
mysql> show global status like 'open_table_definitions';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Open_table_definitions | 20000 |
+------------------------+-------+
1 row in set (0.00 sec)
```

자세한 내용은 MySQL 설명서의 MySQL에서 테이블을 열고 닫는 방법을 참조하십시오.

DB 클러스터 또는 DB 인스턴스 파라미터 그룹에서 table_definition_cache 파라미터를 제한하여 데이터 딕셔너리 캐시의 테이블 정의 수를 제한할 수 있습니다. Aurora MySQL의 경우 이 값은 테이블 정의 캐시의 테이블 수에 대한 소프트 제한 역할을 합니다. 기본값은 인스턴스 클래스에 따라 달라지며 다음과 같이 설정됩니다.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

테이블 수가 table_definition_cache 한도를 초과하면 가장 오래전에 사용됨(LRU) 메커니즘이 캐시에서 테이블을 없애고 제거합니다. 그러나 외래 키 관계와 관련된 테이블은 LRU 목록에 배치되지 않으므로 제거가 불가능합니다.

현재 시나리오에서는 [FLUSH TABLES](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables)을 실행하여 테이블 정의 캐시를 지웁니다. 이 작업을 수행하면 다음과 같이 [Open_table_definitions](https://dev.mysql.com/doc/refman/8.0/en/server-status-variables.html#statvar_Open_table_definitions) 글로벌 상태 변수가 2만에서 12로 크게 감소합니다.

```
mysql> show global status like 'open_table_definitions';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Open_table_definitions | 12    |
+------------------------+-------+
1 row in set (0.00 sec)
```

이러한 감소에도 불구하고 memory/innodb/memory에 대한 메모리 할당은 5.18GiB로 높게 유지되고 할당된 딕셔너리 메모리도 변경되지 않는 것으로 관찰됩니다. 이는 다음 쿼리 결과에서 확실히 알 수 있습니다.

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-------------------------------------------------------------------+---------------+
| event_name                                                        | current_alloc |
+-------------------------------------------------------------------+---------------+
| memory/innodb/memory                                              | 5.18 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                            | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                  | 388.95 MiB    |
| memory/innodb/std                                                 | 226.88 MiB    |
| memory/innodb/fil0fil                                             | 198.49 MiB    |
| memory/sql/binlog_io_cache                                        | 128.00 MiB    |
| memory/innodb/mem0mem                                             | 96.82 MiB     |
```

```
| memory/innodb/dict0dict                                          | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                       | 88.00 MiB     |
+------------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5001599639
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

이렇게 지속적으로 높은 메모리 사용량은 외래 키 관계와 관련된 테이블 때문일 수 있습니다. 이러한 테이블은 제거를 위해 LRU 목록에 배치되지 않으므로 테이블 정의 캐시를 플러싱한 후에도 메모리 할당이 높은 상태를 유지하는 이유가 설명됩니다.

이 문제를 해결하려면:

1. 데이터베이스 스키마, 특히 외래 키 관계를 검토하고 최적화하세요.
2. 딕셔너리 객체를 수용할 수 있는 메모리가 더 많은 더 큰 DB 인스턴스 클래스로 이동하는 것을 고려해 보세요.

다음 단계에 따라 메모리 할당 패턴을 이해하면 Aurora MySQL DB 인스턴스의 메모리 사용량을 더 잘 관리하고 메모리 압박으로 인한 잠재적 성능 문제를 방지할 수 있습니다.

Aurora MySQL 데이터베이스의 메모리 부족 문제 해결

The Aurora MySQL `aurora_oom_response` 인스턴스 수준 파라미터를 사용하면 DB 인스턴스가 시스템 메모리를 모니터링하고 다양한 명령문 및 연결에서 소비되는 메모리를 예측할 수 있습니다. 시스템 메모리가 부족해지면 시스템은 이 메모리를 해제하려고 시도하는 작업 목록을 수행할 수 있습니다. 메모리 부족(OOM) 문제로 인한 데이터베이스 재시작을 피하려고 시도하는 과정에서 그러한 작업 목록을 수행합니다. 이 인스턴스 수준 파라미터는 메모리가 부족할 때 DB 인스턴스가 수행하는 쉼표로 구분된 작업 문자열을 사용합니다. `aurora_oom_response` 파라미터는 Aurora MySQL 버전 2 및 3에서 지원됩니다.

다음 값과 그 조합을 aurora_oom_response 파라미터에 사용할 수 있습니다. 문자열을 비워두면 어떠한 작업도 수행하지 않는다는 뜻이며 해당 기능이 사실상 해제되어 데이터베이스가 OOM으로 인해 재시작되기 쉽습니다.

- decline – DB 인스턴스 메모리가 부족해지면 새 쿼리를 거부합니다.
- kill_connect - 많은 양의 메모리를 사용하는 데이터베이스 연결을 닫고 현재 트랜잭션과 데이터 정의 언어(DDL) 문을 종료합니다. 이 응답은 Aurora MySQL 버전 2에서 지원되지 않습니다.

  자세한 내용은 MySQL 설명서의 [KILL statement](#)를 참조하세요.
- kill_query – 인스턴스 메모리가 하한값 이상이 될 때까지 메모리 사용량이 많은 순서로 쿼리를 종료합니다. DDL 문이 종료되지 않습니다.

  자세한 내용은 MySQL 설명서의 [KILL statement](#)를 참조하세요.
- print - 많은 양의 메모리를 사용하는 쿼리만 인쇄합니다.
- tune – 내부 테이블 캐시를 조정하여 일부 메모리를 시스템으로 돌려줍니다. Aurora MySQL은 메모리가 부족해지면 table_open_cache, table_definition_cache 같은 캐시에 사용하는 메모리를 줄입니다. 그리고 시스템의 메모리가 부족해지지 않게 되면 이러한 캐시에 사용하는 메모리를 정상 수준으로 되돌립니다.

  자세한 내용은 MySQL 설명서의 [table_open_cache](#) 및 [table_definition_cache](#)를 참조하세요.
- tune_buffer_pool - 버퍼 풀의 크기를 줄여 일부 메모리를 확보하고 데이터베이스 서버가 연결을 처리하는 데 사용할 수 있도록 합니다. 이 응답은 Aurora MySQL 버전 3.06 이상에서 지원됩니다.

  tune_buffer_pool을 aurora_oom_response 파라미터 값 중 kill_query 또는 kill_connect와 짝지어야 합니다. 그러지 않으면 파라미터 값에 tune_buffer_pool을 포함하더라도 버퍼 풀 크기가 조정되지 않습니다.

3.06 미만의 Aurora MySQL 버전에서 메모리가 4GiB 이하인 DB 인스턴스 클래스의 경우, 인스턴스에 메모리 부족 현상이 발생하면 기본 작업에 print, tune, decline, kill_query 등이 포함됩니다. 메모리가 4GiB보다 큰 DB 인스턴스 클래스의 경우, 파라미터 값은 기본적으로 비어 있습니다(비활성화됨).

Aurora MySQL 버전 3.06 이상에서는 메모리가 4GiB 이하인 DB 인스턴스 클래스의 경우, Aurora MySQL은 메모리를 가장 많이 소비하는 연결도 종료합니다(kill_connect). 메모리가 4GiB보다 큰 DB 인스턴스 클래스의 경우, 기본 파라미터 값은 print입니다.

Aurora MySQL 버전 3.09 이상에서 메모리가 4GiB보다 큰 DB 인스턴스 클래스의 경우, 기본 파라미터 값은 print,decline,kill_connect입니다.

메모리 부족 문제가 자주 발생하는 경우, `performance_schema`가 활성화되면 메모리 요약 테이블을 사용하여 메모리 사용량을 모니터링할 수 있습니다.

OOM과 관련된 Amazon CloudWatch 지표에 대해서는 Amazon Aurora에 대한 인스턴스 수준 지표의 내용을 참조하세요. OOM과 관련된 글로벌 상태 변수는 Aurora MySQL 글로벌 상태 변수의 내용을 참조하세요.

Aurora MySQL 데이터베이스에 대한 로깅

Aurora MySQL 로그는 데이터베이스 활동 및 오류에 대한 필수 정보를 제공합니다. 이러한 로그를 활성화하면 문제를 식별 및 해결하고, 데이터베이스 성능을 이해하고, 데이터베이스 활동을 감사할 수 있습니다. 데이터베이스의 성능과 가용성을 최적화하려면 모든 Aurora MySQL DB 인스턴스에 대해 이러한 로그를 활성화하는 것이 좋습니다. 다음과 같은 유형의 로그를 활성화할 수 있습니다. 각 로그에는 데이터베이스 처리에 미치는 영향을 파악할 수 있는 특정 정보가 포함되어 있습니다.

- 오류 - Aurora MySQL에서는 시작, 종료 및 오류 발생 시에만 오류 로그에 데이터가 로그됩니다. DB 인스턴스는 오류 로그에 새 항목이 기록되지 않는 상태로 몇 시간이나 며칠씩 작동할 수 있습니다. 최근 항목이 보이지 않으면 이는 서버에서 로그에 입력될 만한 오류가 발생하지 않았기 때문입니다. 오류 로깅은 기본적으로 활성화됩니다. 자세한 내용은 Aurora MySQL 오류 로그 단원을 참조하십시오.
- 일반 - 일반 로그는 데이터베이스 엔진에서 실행되는 모든 SQL 문을 포함하여 데이터베이스 활동에 대한 자세한 정보를 제공합니다. 일반 로깅을 활성화하고 로깅 파라미터를 설정하는 방법에 대한 자세한 내용은 MySQL 설명서의 Aurora MySQL 느린 쿼리 로그 및 일반 로그 및 The general query log를 참조하세요.

  > **Note**
  >
  > 일반 로그는 크기가 매우 커져 스토리지를 차지할 수 있습니다. 자세한 내용은 Aurora MySQL용 로그 교체 및 보존 단원을 참조하십시오.

- 느린 쿼리 - 느린 쿼리 로그는 실행하는 데 long_query_time초가 넘게 걸리고 검토할 행이 최소 min_examined_row_limit개여야 하는 SQL 문으로 구성됩니다. 느린 쿼리 로그를 사용하면 실행 시간이 오래 걸리고 따라서 최적화가 필요한 쿼리를 찾을 수 있습니다.

  [`long_query_time`]의 기본값은 10초입니다. 먼저 높은 값으로 시작하여 가장 느린 쿼리를 식별한 다음 세부 튜닝을 위해 낮은 값으로 내려가는 것이 좋습니다.

  `log_slow_admin_statements` 및 `log_queries_not_using_indexes`와 같은 관련 파라미터를 사용할 수도 있습니다. `rows_examined`를 `rows_returned`에 비교하세요. `rows_examined`가 `rows_returned`보다 훨씬 크면 해당 쿼리가 잠재적으로 차단될 수 있습니다.

  Aurora MySQL 버전 3에서는 자세한 내용을 확인할 수 있도록 `log_slow_extra`를 설정할 수 있습니다. 자세한 내용은 MySQL 설명서의 Slow query log contents를 참조하세요. 대화식으로 쿼리 실행을 디버깅하기 위해 세션 수준에서 `long_query_time`을 수정할 수도 있는데, 이는 `log_slow_extra`가 전역적으로 활성화된 경우 특히 유용합니다.

느린 쿼리 로깅을 활성화하고 로깅 파라미터를 설정하는 방법에 대한 자세한 내용은 MySQL 설명서의 Aurora MySQL 느린 쿼리 로그 및 일반 로그 및 The slow query log를 참조하세요.

- 감사 - 감사 로그는 데이터베이스 활동을 모니터링하고 기록합니다. Aurora MySQL에 대한 감사 로깅을 고급 감사라고 합니다. 고급 감사를 활성화하려면 특정 DB 클러스터 파라미터를 설정합니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.
- 바이너리 - 바이너리 로그(binlog)에는 테이블 생성 작업 및 테이블 데이터 변경과 같은 데이터베이스 변경 사항을 설명하는 이벤트가 포함됩니다. 또한 행 기반 로깅을 사용하지 않는 한 잠재적으로 변경을 수행했을 수 있는 문(예: 일치하는 행이 없는 DELETE)에 대한 이벤트도 포함됩니다. 또한 바이너리 로그에는 각 문이 업데이트된 데이터에 소요한 시간에 대한 정보도 포함됩니다.

  바이너리 로깅이 활성화된 상태에서 서버를 실행하면 성능이 약간 느려집니다. 그러나 복제를 설정하고 복원 작업을 수행할 수 있도록 하는 바이너리 로그의 이점은 일반적으로 이러한 사소한 성능 저하보다 큽니다.

  > **Note**
  >
  > Aurora MySQL은 복원 작업에 바이너리 로깅이 필요하지 않습니다.

  바이너리 로깅을 활성화하고 binlog 형식을 설정하는 방법에 대한 자세한 내용은 MySQL 설명서의 단일 AZ 데이터베이스의 Aurora MySQL 이진 로깅 구성 및 The binary log를 참조하세요.

오류, 일반, 느린 쿼리 및 감사 로그를 Amazon CloudWatch Logs에 게시할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 데이터베이스 로그 게시 단원을 참조하십시오.

느린 로그 파일, 일반 로그 파일, 바이너리 로그 파일을 요약하는 데 유용한 또 다른 도구는 pt-query-digest입니다.

Aurora MySQL 데이터베이스의 연결 문제 해결

워크로드를 원활하게 운영하려면 애플리케이션과 RDS DB 인스턴스 간의 안정적인 연결을 보장하는 것이 중요합니다. 그러나 네트워크 구성, 인증 문제 또는 리소스 제약과 같은 다양한 요인으로 인해 연결 문제가 발생할 수 있습니다. 이 안내서는 Aurora MySQL의 연결 문제를 해결하기 위한 포괄적인 접근 방식을 제공하는 것을 목표로 합니다.

목차

Aurora MySQL의 데이터베이스 연결 문제 식별

연결 문제의 특정 범주를 식별하면 잠재적 원인의 범위를 좁히고 문제 해결 프로세스에 도움이 될 수 있습니다. 각 범주에는 진단 및 해결을 위한 다양한 접근 방식과 기법이 필요할 수 있습니다. 데이터베이스 연결 문제는 크게 다음 범주로 분류할 수 있습니다.

연결 오류 및 예외

연결 오류 및 예외는 잘못된 연결 문자열, 인증 실패, 네트워크 중단 또는 데이터베이스 서버 문제와 같은 다양한 이유로 발생할 수 있습니다. 잘못 구성된 연결 파라미터, 잘못된 자격 증명, 네트워크 중단 또는 데이터베이스 서버 충돌 또는 재시작이 원인일 수 있습니다. 잘못 구성된 보안 그룹, 가상 프라이빗 클라우드(VPC) 설정, 네트워크 액세스 제어 목록(ACL) 및 서브넷과 연결된 라우팅 테이블도 연결 문제로 이어질 수 있습니다.

연결 한도 도달

이 문제는 데이터베이스 서버에 대한 동시 연결 수가 최대 허용 한도를 초과할 때 발생합니다. 데이터베이스 서버에는 일반적으로 클러스터 및 인스턴스 파라미터 그룹의 파라미터 max_connections로 정의된 구성 가능한 최대 연결 한도가 있습니다. 연결 한도를 적용하면 데이터베이스 서버가 기

존 연결을 효율적으로 처리하고 허용 가능한 성능을 제공하기에 충분한 리소스(예: 메모리, CPU 및 파일 핸들)를 보유하게 됩니다. 원인에는 애플리케이션의 연결 유출, 비효율적인 연결 풀링 또는 연결 요청의 예상치 못한 급증 등이 포함될 수 있습니다.

연결 시간 초과

연결 시간 초과는 클라이언트 애플리케이션이 지정된 제한 시간 내에 데이터베이스 서버와의 연결을 설정할 수 없는 경우에 발생합니다. 일반적인 원인에는 네트워크 문제, 서버 과부하, 방화벽 규칙 및 잘못 구성된 연결 설정이 포함됩니다.

유휴 연결 제한 시간

장시간 비활성 상태로 유지되는 유휴 연결은 리소스를 절약하기 위해 데이터베이스 서버에서 자동으로 닫힐 수 있습니다. 이 제한 시간은 일반적으로 `wait_timeout` 및 `interactive_timeout parameters`를 사용하여 구성할 수 있으며 애플리케이션의 연결 사용 패턴에 따라 조정해야 합니다. 연결을 장시간 유휴 상태로 두는 애플리케이션 로직 또는 부적절한 연결 관리가 원인일 수 있습니다.

기존 연결의 간헐적 연결 해제

이 오류 클래스는 클라이언트 애플리케이션과 데이터베이스 간에 설정된 연결이 활성 상태이고 사용 중임에도 불구하고 불규칙한 간격으로 예기치 않게 종료되거나 연결이 끊어지는 시나리오를 나타냅니다. 이러한 연결 해제는 간헐적으로 발생하므로 불규칙한 간격으로 발생하며 일관되게 발생하지 않습니다. 원인에는 다음이 포함될 수 있습니다.

- 재시작 또는 장애 조치와 같은 데이터베이스 서버 문제
- 부적절한 애플리케이션 연결 처리
- 로드 밸런싱 및 프록시 문제
- 네트워크 불안정성
- 연결 경로와 관련된 타사 구성 요소 또는 미들웨어 문제
- 쿼리 실행 제한 시간
- 서버 또는 클라이언트 측의 리소스 제약 조건

포괄적인 모니터링, 로깅 및 분석을 통해 근본 원인을 식별하는 것이 중요하며, 적절한 오류 처리, 연결 풀링 및 재시도 메커니즘을 구현하면 이러한 간헐적 연결 해제가 애플리케이션의 기능 및 사용자 경험에 미치는 영향을 완화하는 데 도움이 될 수 있습니다.

Aurora MySQL의 연결 문제에 대한 데이터 수집

애플리케이션, 데이터베이스, 네트워크 및 인프라 구성 요소와 관련된 포괄적인 데이터를 수집하는 것은 애플리케이션과 Aurora MySQL 데이터베이스 간의 연결 문제를 효과적으로 해결하는 데 매우 중요합니다. 관련 로그, 구성 및 진단 정보를 수집하면 연결 문제의 근본 원인을 파악하고 적절한 해결 방법을 수행하는 데 도움이 되는 가치 있는 인사이트를 얻을 수 있습니다.

보안 그룹 규칙, VPC 설정 및 라우팅 테이블과 같은 네트워크 로그 및 구성은 애플리케이션이 데이터베이스와 성공적으로 연결되지 못하게 할 수 있는 잠재적인 네트워크 관련 병목 현상 또는 잘못된 구성을 식별하는 데 필수적입니다. 이러한 네트워크 구성 요소를 분석하여 필요한 포트가 열려 있고 IP 주소가 허용되며 라우팅 구성이 올바르게 설정되었는지 확인할 수 있습니다.

타임스탬프

연결 문제가 발생할 때 정확한 타임스탬프를 기록합니다. 이를 통해 패턴을 식별하거나 문제를 다른 이벤트 또는 활동과 상호 연관시킬 수 있습니다.

DB 엔진 로그

일반 데이터베이스 로그 외에도 데이터베이스 엔진 로그(예: MySQL 오류 로그 및 느린 쿼리 로그)에서 간헐적 연결 문제와 관련이 있을 수 있는 관련 정보 또는 오류를 검토합니다. 자세한 내용은 Aurora MySQL 데이터베이스에 대한 로깅 단원을 참조하십시오.

클라이언트 애플리케이션 로그

데이터베이스에 연결하는 클라이언트 애플리케이션에서 세부 로그를 수집합니다. 애플리케이션 로그는 애플리케이션의 관점에서 연결 시도, 오류 및 관련 정보를 볼 수 있는 가시성을 제공하여 연결 문자열, 인증 자격 증명 또는 애플리케이션 수준 연결 처리와 관련된 문제를 드러낼 수 있습니다.

반면 데이터베이스 로그는 데이터베이스 측 오류, 느린 쿼리 또는 연결 문제에 기여할 수 있는 이벤트에 대한 통찰력을 제공합니다. 자세한 내용은 Aurora MySQL 데이터베이스에 대한 로깅 단원을 참조하십시오.

클라이언트 환경 변수

클라이언트 측의 환경 변수 또는 구성 설정이 프록시 설정, SSL/TLS 설정 또는 기타 관련 변수와 같은 데이터베이스 연결에 영향을 미칠 수 있는지 확인합니다.

클라이언트 라이브러리 버전

클라이언트가 데이터베이스 연결에 사용되는 모든 데이터베이스 드라이버, 라이브러리 또는 프레임워크의 최신 버전을 사용하고 있는지 확인합니다. 오래된 버전에는 알려진 문제 또는 호환성 문제가 있을 수 있습니다.

클라이언트 네트워크 캡처

연결 문제가 발생하는 동안 Wireshark 또는 `tcpdump`와 같은 도구를 사용하여 클라이언트 측에서 네트워크 캡처를 수행합니다. 이렇게 하면 클라이언트 측에서 네트워크 관련 문제 또는 이상을 식별하는 데 도움이 될 수 있습니다.

클라이언트 네트워크 토폴로지

클라이언트가 직접 연결하는 대신 데이터베이스에 연결하는 방화벽, 로드 밸런서 또는 RDS 프록시 또는 프록시 SQL과 같은 기타 구성 요소를 포함하여 클라이언트의 네트워크 토폴로지를 이해합니다.

클라이언트 운영 체제 설정

방화벽 규칙, 네트워크 어댑터 설정 및 기타 관련 설정과 같이 네트워크 연결에 영향을 미칠 수 있는 클라이언트의 운영 체제 설정을 확인합니다.

연결 풀링 구성

애플리케이션에서 연결 풀링 메커니즘을 사용하는 경우 구성 설정을 검토하고 풀 지표(예: 활성 연결, 유휴 연결 및 연결 제한 시간)를 모니터링하여 풀이 올바르게 작동하는지 확인합니다. 또한 최대 풀 크기, 최소 풀 크기 및 연결 검증 설정과 같은 풀 설정을 검토하여 올바르게 구성되었는지 확인합니다.

연결 문자열

연결 문자열에는 일반적으로 호스트 이름 또는 엔드포인트, 포트 번호, 데이터베이스 이름 및 인증 자격 증명과 같은 파라미터가 포함됩니다. 연결 문자열을 분석하면 연결 문제를 일으킬 수 있는 구성 오류 또는 잘못된 설정을 식별하는 데 도움이 될 수 있습니다. 예를 들어 호스트 이름이나 포트 번호가 올바르지 않으면 클라이언트가 데이터베이스 인스턴스에 도달하지 못할 수 있지만 잘못된 인증 자격 증명은 인증 실패 및 연결 거부로 이어질 수 있습니다. 또한 연결 문자열은 연결 풀링, 제한 시간 또는 연결 문제에 기여할 수 있는 기타 연결별 설정과 관련된 문제를 드러낼 수 있습니다. 클라이언트 애플리케이션에서 사용하는 전체 연결 문자열을 제공하면 클라이언트의 잘못된 구성을 정확히 파악하는 데 도움이 될 수 있습니다.

데이터베이스 지표

연결 문제가 발생하는 동안 CPU 사용량, 메모리 사용량 및 디스크 I/O와 같은 데이터베이스 지표를 모니터링합니다. 이를 통해 DB 인스턴스에 리소스 경합 또는 성능 문제가 있는지 확인할 수 있습니다.

DB 엔진 버전

Aurora MySQL DB 엔진 버전을 참조하세요. AWS는 알려진 문제, 보안 취약성을 해결하고 성능 향상을 도입하는 업데이트를 정기적으로 릴리스합니다. 따라서 이러한 업데이트에는 특히 연결, 성능

및 안정성과 관련된 버그 수정 및 개선 사항이 포함되어 있으므로 사용 가능한 최신 버전으로 업그레이드하는 것이 좋습니다. 데이터베이스 버전 정보를 다른 수집된 세부 정보와 함께 제공하면 지원가 연결 문제를 효과적으로 진단하고 해결하는 데 도움이 될 수 있습니다.

네트워크 지표

연결 문제가 발생하는 동안 지연 시간, 패킷 손실 및 처리량과 같은 네트워크 지표를 수집합니다. `ping`, `traceroute` 및 네트워크 모니터링 도구와 같은 도구는 이 데이터를 수집하는 데 도움이 될 수 있습니다.

소스 및 클라이언트 세부 정보

애플리케이션 서버, 로드 밸런서 또는 데이터베이스 연결을 시작하는 기타 구성 요소의 IP 주소를 확인합니다. 이는 단일 IP 주소 또는 IP 주소 범위(CIDR 표기법)일 수 있습니다. 소스가 Amazon EC2 인스턴스인 경우 인스턴스 유형, 가용 영역, 서브넷 ID 및 인스턴스와 연결된 보안 그룹, 프라이빗 IP 주소 및 퍼블릭 IP 주소와 같은 네트워크 인터페이스 세부 정보를 검토하는 것도 도움이 됩니다.

수집된 데이터를 철저히 분석하면 구성 오류, 리소스 제약, 네트워크 중단 또는 간헐적이거나 지속적인 연결 문제를 유발하는 기타 기본 문제를 식별할 수 있습니다. 이 정보를 사용하면 구성 조정, 네트워크 문제 해결 또는 애플리케이션 수준 연결 처리 처리와 같은 대상 작업을 수행할 수 있습니다.

Aurora MySQL에 대한 데이터베이스 연결 모니터링

연결 문제를 모니터링하고 해결하기 위해 다음 지표와 기능을 사용할 수 있습니다.

CloudWatch 지표

- `CPUUtilization` – DB 인스턴스에서 CPU 사용량이 높으면 쿼리 실행이 느려져 연결 시간 초과 또는 거부가 발생할 수 있습니다.
- `DatabaseConnections` – DB 인스턴스에 대한 활성 연결 수를 모니터링합니다. 구성된 최댓값에 가까운 연결 수가 많으면 잠재적 연결 문제 또는 연결 풀 소진을 나타낼 수 있습니다.
- `FreeableMemory` – 사용 가능한 메모리가 적으면 리소스 제약으로 인해 성능 문제와 연결 문제가 발생할 수 있습니다.
- `NetworkReceiveThroughput` 및 `NetworkTransmitThroughput` - 비정상적인 네트워크 처리량 급증 또는 하락은 연결 문제 또는 네트워크 병목 현상을 나타낼 수 있습니다.

성능 개선 도우미 지표

성능 개선 도우미를 사용하여 Aurora MySQL의 연결 문제를 해결하려면 다음과 같은 데이터베이스 지표를 분석합니다.

- Aborted_clients
- Aborted_connects
- 연결
- max_connections
- Threads_connected
- Threads_created
- Threads_running

이러한 지표는 연결 병목 현상을 식별하고, 네트워크 또는 인증 문제를 감지하며, 연결 풀링을 최적화하고, 효율적인 스레드 관리를 보장하는 데 도움이 될 수 있습니다. 자세한 내용은 Aurora MySQL용 성능 개선 도우미 카운터 단원을 참조하십시오.

성능 개선 도우미 기능

- 데이터베이스 로드 - 시간 경과에 따라 데이터베이스 로드를 시각화하고 연결 문제 또는 성능 저하와 상호 연관시킵니다.
- SQL 통계 - SQL 통계를 분석하여 연결 문제에 기여할 수 있는 비효율적인 쿼리 또는 데이터베이스 작업을 식별합니다.
- 상위 쿼리 - 가장 리소스 집약적인 쿼리를 식별하고 분석하여 연결 문제를 일으킬 수 있는 잠재적 성능 병목 현상 또는 장기 실행 쿼리를 식별하는 데 도움이 될 수 있습니다.

이러한 지표를 모니터링하고 성능 개선 도우미를 활용하면 연결 문제를 일으킬 수 있는 데이터베이스 인스턴스의 성능, 리소스 사용량 및 잠재적 병목 현상을 파악할 수 있습니다. 예시:

- 최대 한도에 가까운 높은 `DatabaseConnections`는 연결 풀 소진 또는 부적절한 연결 처리로 인해 연결 문제가 발생함을 나타낼 수 있습니다.
- 높은 `CPUUtilization` 또는 낮은 `FreeableMemory`은 리소스 제약을 나타낼 수 있으며, 이로 인해 쿼리 실행이 느려지고 연결 제한 시간이 초과되거나 거부될 수 있습니다.
- 상위 쿼리 및 SQL 통계를 분석하면 연결 문제에 기여할 수 있는 비효율적이거나 리소스 집약적인 쿼리를 식별하는 데 도움이 될 수 있습니다.

또한 CloudWatch Logs를 모니터링하고 경보를 설정하면 연결 문제가 에스컬레이션되기 전에 사전에 식별하고 대응하는 데 도움이 될 수 있습니다.

이러한 지표와 도구는 가치 있는 인사이트를 제공할 수 있지만 다른 문제 해결 단계와 함께 사용해야 합니다. 또한 네트워크 구성, 보안 그룹 규칙 및 애플리케이션 수준 연결 처리를 검토하여 Aurora MySQL DB 인스턴스의 연결 문제를 포괄적으로 진단하고 해결할 수 있습니다.

Aurora MySQL에 대한 추가 모니터링

CloudWatch 지표

- `AbortedClients` - 제대로 종료되지 않은 클라이언트 연결 수를 추적합니다.
- `AuroraSlowConnectionHandleCount` - 느린 연결 핸들 작업 수를 추적하여 잠재적 연결 문제 또는 성능 병목 현상을 나타냅니다.
- `AuroraSlowHandshakeCount` - 느린 핸드셰이크 작업 수를 측정하며, 이는 연결 문제를 나타내는 지표일 수도 있습니다.
- `ConnectionAttempts` - Aurora MySQL DB 인스턴스에 대한 연결 시도 횟수를 측정합니다.

글로벌 상태 변수

`Aurora_external_connection_count` - DB 인스턴스에 대한 데이터베이스 연결 수를 나타내며, 데이터베이스 상태 확인에 사용되는 RDS 서비스 연결은 제외됩니다.

이러한 지표와 글로벌 상태 변수를 모니터링하면 Amazon Aurora MySQL 인스턴스에 연결 문제를 일으킬 수 있는 연결 패턴, 오류 및 잠재적 병목 현상을 파악할 수 있습니다.

예를 들어 `AbortedClients` 또는 `AuroraSlowConnectionHandleCount`의 수가 많으면 연결 문제를 나타낼 수 있습니다.

또한 CloudWatch 경보 및 알림을 설정하면 연결 문제가 에스컬레이션되고 애플리케이션 성능에 영향을 미치기 전에 사전에 식별하고 대응하는 데 도움이 될 수 있습니다.

Aurora MySQL의 연결 오류 코드

다음은 Aurora MySQL 데이터베이스의 몇 가지 일반적인 연결 오류와 오류 코드 및 설명입니다.

오류 코드 1040: 연결이 너무 많음

이 오류는 클라이언트가 데이터베이스 서버에서 허용하는 최댓값보다 많은 연결을 설정하려고 할 때 발생합니다. 가능한 값은 다음을 포함합니다.

- 연결 풀링 구성 오류 - 연결 풀링 메커니즘을 사용하는 경우 최대 풀 크기가 너무 높게 설정되지 않았는지, 연결이 풀로 다시 제대로 릴리스되고 있는지 확인합니다.
- 데이터베이스 인스턴스 구성 - 데이터베이스 인스턴스에 허용되는 최대 연결 설정을 확인하고 필요한 경우 `max_connections` 파라미터를 설정하여 조정합니다.
- 높은 동시성 - 여러 클라이언트 또는 애플리케이션이 데이터베이스에 동시에 연결하는 경우 허용되는 최대 연결 한도에 도달할 수 있습니다.

오류 코드 1045: 사용자 '...'@'...'에 대한 액세스 거부됨(암호 사용: YES/NO)

이 오류는 데이터베이스에 연결을 시도할 때 인증 실패를 나타냅니다. 가능한 값은 다음을 포함합니다.

- 인증 플러그인 호환성 - 클라이언트에서 사용하는 인증 플러그인이 데이터베이스 서버의 인증 메커니즘과 호환되는지 확인합니다.
- 잘못된 사용자 이름 또는 암호 - 연결 문자열 또는 인증 메커니즘에서 올바른 사용자 이름과 암호가 사용되고 있는지 확인합니다.
- 사용자 권한 - 사용자에게 지정된 호스트 또는 네트워크에서 데이터베이스 인스턴스에 연결하는 데 필요한 권한이 있는지 확인합니다.

오류 코드 1049: 알 수 없는 데이터베이스 '...'

이 오류는 클라이언트가 서버에 없는 데이터베이스에 연결을 시도하고 있음을 나타냅니다. 가능한 값은 다음을 포함합니다.

- 데이터베이스가 생성되지 않음 - 지정된 데이터베이스가 데이터베이스 서버에서 생성되었는지 확인합니다.
- 잘못된 데이터베이스 이름 - 연결 문자열 또는 쿼리에 사용된 데이터베이스 이름이 정확한지 다시 확인합니다.
- 사용자 권한 - 사용자에게 지정된 데이터베이스에 액세스하는 데 필요한 권한이 있는지 확인합니다.

오류 코드 1153: 'max_allowed_packet' 바이트보다 큰 패킷을 가져옴

이 오류는 클라이언트가 데이터베이스 서버에서 허용하는 최대 패킷 크기를 초과하는 데이터를 보내거나 받으려고 할 때 발생합니다. 가능한 값은 다음을 포함합니다.

- 대규모 쿼리 또는 결과 세트 - 대량의 데이터가 포함된 쿼리를 실행하는 경우 패킷 크기 제한을 초과할 수 있습니다.
- 잘못 구성된 패킷 크기 설정 - 데이터베이스 서버의 `max_allowed_packet` 설정을 확인하고 필요한 경우 조정합니다.
- 네트워크 구성 문제 - 네트워크 구성(예: MTU 크기)이 필요한 패킷 크기를 허용하는지 확인합니다.

오류 코드 1226: '...' 사용자가 'max_user_connections' 리소스를 초과했습니다(현재 값: ...).

이 오류는 사용자가 데이터베이스 서버에서 허용하는 최대 동시 연결 수를 초과했음을 나타냅니다. 가능한 값은 다음을 포함합니다.

- 연결 풀링 구성 오류 - 연결 풀링 메커니즘을 사용하는 경우 최대 풀 크기가 사용자의 연결 한도에 비해 너무 높게 설정되지 않았는지 확인합니다.
- 데이터베이스 인스턴스 구성 - 데이터베이스 인스턴스의 `max_user_connections` 설정을 확인하고 필요한 경우 조정합니다.
- 높은 동시성 - 여러 클라이언트 또는 애플리케이션이 동일한 사용자를 사용하여 데이터베이스에 동시에 연결하는 경우 사용자별 연결 한도에 도달할 수 있습니다.

오류 코드 2003: '...'에서 MySQL 서버에 연결할 수 없음(10061)

이 오류는 일반적으로 클라이언트가 데이터베이스 서버와 TCP/IP 연결을 설정할 수 없을 때 발생합니다. 다음과 같은 다양한 문제로 인해 발생할 수 있습니다.

- 데이터베이스 인스턴스 상태 - 데이터베이스 인스턴스가 `available` 상태이고 유지 관리 또는 백업 작업을 수행하지 않는지 확인합니다.
- 방화벽 규칙 - 방화벽(운영 체제, 네트워크 또는 보안 그룹)이 지정된 포트(일반적으로 MySQL의 경우 3306)에서 연결을 차단하고 있는지 확인합니다.
- 잘못된 호스트 이름 또는 엔드포인트 - 연결 문자열에 사용된 호스트 이름 또는 엔드포인트가 올바르고 데이터베이스 인스턴스와 일치하는지 확인합니다.
- 네트워크 연결 문제 - 클라이언트 시스템이 네트워크를 통해 데이터베이스 인스턴스에 도달할 수 있는지 확인합니다. 네트워크 중단, 라우팅 문제 또는 VPC 또는 서브넷 구성 오류가 있는지 확인합니다.

오류 코드 2005: 알 수 없는 MySQL 서버 호스트 '...'(11001)

이 오류는 클라이언트가 데이터베이스 서버의 호스트 이름 또는 엔드포인트를 IP 주소로 확인할 수 없을 때 발생합니다. 가능한 값은 다음을 포함합니다.

- DNS 해결 문제 - 클라이언트 시스템이 DNS를 사용하여 호스트 이름을 올바르게 해결할 수 있는지 확인합니다. DNS 설정, DNS 캐시를 확인하고 호스트 이름 대신 IP 주소를 사용해 보세요.
- 잘못된 호스트 이름 또는 엔드포인트 - 연결 문자열에 사용된 호스트 이름 또는 엔드포인트가 정확한지 다시 확인합니다.
- 네트워크 구성 문제 - 클라이언트의 네트워크 구성(예: VPC, 서브넷 및 라우팅 테이블)이 DNS 확인 및 데이터베이스 인스턴스에 대한 연결을 허용하는지 확인합니다.

오류 코드 2026: SSL 연결 오류

이 오류는 연결 시도 중에 SSL/TLS 구성 또는 인증서 검증에 문제가 있을 때 발생합니다. 가능한 값은 다음을 포함합니다.

- 인증서 만료 - 서버에서 사용하는 SSL/TLS 인증서가 만료되어 갱신해야 하는지 확인합니다.

- 인증서 검증 문제 - 클라이언트가 서버의 SSL/TLS 인증서를 올바르게 검증할 수 있고 인증서가 신뢰할 수 있는지 확인합니다.
- 네트워크 구성 문제 - 네트워크 구성이 SSL/TLS 연결을 허용하고 SSL/TLS 핸드셰이크 프로세스를 차단하거나 방해하지 않는지 확인합니다.
- SSL/TLS 구성 불일치 - 클라이언트 및 서버의 SSL/TLS 설정(예: 암호 제품군 및 프로토콜 버전)이 호환되는지 확인합니다.

각 오류 코드에 대한 자세한 설명과 잠재적 원인을 이해하면 Aurora MySQL 데이터베이스 작업 시 연결 문제를 더 잘 해결할 수 있습니다.

Aurora MySQL에 대한 파라미터 튜닝 권장 사항

최대 연결 수

이러한 파라미터를 조정하면 허용되는 최대 연결 한도에 도달하여 발생하는 연결 문제를 방지하는 데 도움이 될 수 있습니다. 애플리케이션의 동시성 요구 사항 및 리소스 제약 조건에 따라 이러한 값이 적절하게 설정되었는지 확인합니다.

- `max_connections` - 이 파라미터는 DB 인스턴스에 허용되는 최대 동시 연결 수를 지정합니다.
- `max_user_connections` - 이 파라미터는 사용자 생성 및 수정 중에 지정할 수 있으며 특정 사용자 계정에 허용되는 최대 동시 연결 수를 설정합니다.

네트워크 버퍼 크기

이러한 값을 늘리면 네트워크 성능이 향상될 수 있습니다. 특히 대규모 데이터 전송 또는 결과 세트가 포함된 워크로드의 경우 더욱 그렇습니다. 그러나 버퍼 크기가 클수록 메모리를 더 많이 사용할 수 있으므로 주의해야 합니다.

- `net_buffer_length` - 이 파라미터는 클라이언트 연결 및 결과 버퍼의 초기 크기를 설정하여 메모리 사용량과 쿼리 성능의 균형을 맞춥니다.
- `max_allowed_packet` - 이 파라미터는 DB 인스턴스에서 전송하거나 수신할 수 있는 단일 네트워크 패킷의 최대 크기를 지정합니다.

네트워크 압축(클라이언트 측)

네트워크 압축을 활성화하면 네트워크 대역폭 사용량이 줄어들 수 있지만 클라이언트와 서버 측 모두에서 CPU 오버헤드가 증가할 수 있습니다.

- `compress` – 이 파라미터는 클라이언트/서버 통신을 위한 네트워크 압축을 활성화하거나 비활성화합니다.
- `compress_protocol` - 이 파라미터는 네트워크 통신에 사용할 압축 프로토콜을 지정합니다.

네트워크 성능 튜닝

이러한 제한 시간을 조정하면 유휴 연결을 관리하고 리소스 소진을 방지하는 데 도움이 될 수 있지만 값이 낮으면 연결이 조기에 종료될 수 있으므로 주의해야 합니다.

- `interactive_timeout` - 이 파라미터는 서버가 대화형 연결을 닫기 전에 활동을 기다리는 초 수를 지정합니다.
- `wait_timeout` - 이 파라미터는 서버가 비대화형 연결을 닫기 전에 활동을 기다리는 초 수를 정합니다.

네트워크 제한 시간 설정

이러한 제한 시간을 조정하면 느리거나 응답하지 않는 연결과 관련된 문제를 해결하는 데 도움이 될 수 있습니다. 하지만 연결이 조기에 실패할 수 있으므로 너무 낮게 설정하지 않도록 주의하세요.

- `net_read_timeout` - 이 파라미터는 읽기 작업을 종료하기 전에 연결에서 더 많은 데이터를 기다리는 초 수를 지정합니다.
- `net_write_timeout` - 이 파라미터는 쓰기 작업을 종료하기 전에 블록이 연결에 기록될 때까지 기다릴 초 수를 결정합니다.

Aurora MySQL에 대한 데이터베이스 연결 문제 해결 예제

다음 예제에서는 Aurora MySQL에 대한 데이터베이스 연결 문제를 식별하고 해결하는 방법을 보여줍니다.

예제 1: 실패한 연결 시도 문제 해결

연결 시도는 인증 실패, SSL/TLS 핸드셰이크 실패, `max_connections` 한도 도달, DB 인스턴스의 리소스 제약 등 여러 가지 이유로 실패할 수 있습니다.

성능 개선 도우미에서 또는 다음 명령을 사용하여 실패한 연결 수를 추적할 수 있습니다.

```
mysql> show global status like 'aborted_connects';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_connects | 7     |
+------------------+-------+
1 row in set (0.00 sec)
```

시간이 지남에 따라 `Aborted_connects` 수가 증가하면 애플리케이션에 간헐적인 연결 문제가 발생할 수 있습니다.

Aurora 고급 감사를 사용하여 클라이언트 연결에서 연결 및 연결 해제를 로깅할 수 있습니다. DB 클러스터 파라미터 그룹에서 다음 파라미터를 설정하여 이 작업을 수행할 수 있습니다.

- `server_audit_logging` = 1
- `server_audit_events` = CONNECT

다음은 실패한 로그인에 대한 감사 로그에서 발췌한 것입니다.

```
1728498527380921,auora-mysql-node1,user_1,172.31.49.222,147189,0,FAILED_CONNECT,,,1045
1728498527380940,auora-mysql-node1,user_1,172.31.49.222,147189,0,DISCONNECT,,,0
```

위치:

- `1728498527380921` - 실패한 로그인이 발생한 시점의 에포크 타임스탬프
- `aurora-mysql-node1` - 연결이 실패한 Aurora MySQL 클러스터 노드의 인스턴스 식별자
- `user_1` - 로그인에 실패한 데이터베이스 사용자의 이름
- `172.31.49.222` – 연결이 설정된 클라이언트의 프라이빗 IP 주소
- `147189` – 실패한 로그인의 연결 ID
- `FAILED_CONNECT` - 연결이 실패했음을 나타냄.
- `1045` - 반환 코드. 0이 아닌 값은 오류를 나타냅니다. 이 경우 `1045`는 액세스 거부에 해당합니다.

자세한 내용은 MySQL 설명서의 서버 오류 코드 및 클라이언트 오류 코드를 참조하세요.

Aurora MySQL 오류 로그에서 다음과 같은 관련 오류 메시지를 검사할 수도 있습니다.

```
2024-10-09T19:26:59.310443Z 220 [Note] [MY-010926] [Server] Access denied for user
 'user_1'@'172.31.49.222' (using password: YES) (sql_authentication.cc:1502)
```

예제 2: 비정상적인 클라이언트 연결 해제 문제 해결

성능 개선 도우미에서 또는 다음 명령을 사용하여 비정상적인 클라이언트 연결 해제 수를 추적할 수 있습니다.

```
mysql> show global status like 'aborted_clients';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
```

```
| Aborted_clients | 9     |
+-----------------+-------+
1 row in set (0.01 sec)
```

시간이 지남에 따라 Aborted_clients의 수가 증가하면 애플리케이션이 데이터베이스에 대한 연결을 올바르게 닫지 않는 것입니다. 연결이 제대로 닫히지 않으면 리소스 누출과 잠재적인 성능 문제가 발생할 수 있습니다. 연결을 불필요하게 열어 두면 메모리 및 파일 설명자와 같은 시스템 리소스를 사용할 수 있으며, 이로 인해 애플리케이션 또는 서버가 응답하지 않거나 다시 시작될 수 있습니다.

다음 쿼리를 사용하여 연결을 제대로 종료하지 않는 계정을 식별할 수 있습니다. 사용자 계정 이름, 사용자가 연결하는 호스트, 종료되지 않은 연결 수, 종료되지 않은 연결 백분율을 검색합니다.

```
SELECT
    ess.user,
    ess.host,
    (a.total_connections - a.current_connections) - ess.count_star AS not_closed,
    (((a.total_connections - a.current_connections) - ess.count_star) * 100) /
 (a.total_connections - a.current_connections) AS pct_not_closed
FROM
    performance_schema.events_statements_summary_by_account_by_event_name AS ess
    JOIN performance_schema.accounts AS a ON (ess.user = a.user AND ess.host = a.host)
WHERE
    ess.event_name = 'statement/com/quit'
    AND (a.total_connections - a.current_connections) > ess.count_star;

+----------+---------------+------------+----------------+
| user     | host          | not_closed | pct_not_closed |
+----------+---------------+------------+----------------+
| user1    | 172.31.49.222 |          1 |        33.3333 |
| user1    | 172.31.93.250 |       1024 |        12.1021 |
| user2    | 172.31.93.250 |         10 |        12.8551 |
+----------+---------------+------------+----------------+
3 rows in set (0.00 sec)
```

연결이 종료되지 않은 사용자 계정과 호스트를 식별한 후 연결을 정상적으로 종료하지 않는 코드를 계속 확인할 수 있습니다.

예를 들어 Python의 MySQL 커넥터를 사용하여 연결 객체의 close() 메서드를 사용하여 연결을 닫습니다. 다음은 데이터베이스에 대한 연결을 설정하고, 쿼리를 수행하고, 연결을 닫는 함수의 예입니다.

```
import mysql.connector
```

```
def execute_query(query):
    # Establish a connection to the database
    connection = mysql.connector.connect(
        host="your_host",
        user="your_username",
        password="your_password",
        database="your_database"
    )

    try:
        # Create a cursor object
        cursor = connection.cursor()

        # Execute the query
        cursor.execute(query)

        # Fetch and process the results
        results = cursor.fetchall()
        for row in results:
            print(row)

    finally:
        # Close the cursor and connection
        cursor.close()
        connection.close()
```

이 예제에서는 예외 발생 여부에 관계없이 연결이 닫혀 있는지 확인하기 위해 `finally` 블록에서 `connection.close()` 메서드를 호출합니다.

Aurora MySQL 데이터베이스 쿼리 성능 문제 해결

MySQL은 쿼리 계획 평가 방법에 영향을 미치는 시스템 변수, 전환 가능한 최적화, 최적화 프로그램 및 인덱스 힌트, 최적화 프로그램 비용 모델을 통해 쿼리 최적화 프로그램 제어 기능을 제공합니다. 이러한 데이터 포인트는 다양한 MySQL 환경을 비교할 때뿐만 아니라 이전 쿼리 실행 계획을 현재 실행 계획과 비교하고 언제든지 MySQL 쿼리의 전체 실행을 이해하는 데 유용할 수 있습니다.

쿼리 성능은 실행 계획, 테이블 스키마 및 크기, 통계, 리소스, 인덱스, 파라미터 구성 등 여러 요인에 따라 달라집니다. 쿼리를 튜닝하려면 병목 현상을 식별하고 실행 경로를 최적화해야 합니다.

- 쿼리의 실행 계획을 찾고 쿼리가 적절한 인덱스를 사용하고 있는지 확인하세요. EXPLAIN을 사용하고 각 계획의 세부 정보를 검토하여 쿼리를 최적화할 수 있습니다.
- Aurora MySQL 버전 3(MySQL 8.0 Community Edition과 호환)은 EXPLAIN ANALYZE 문을 사용합니다. EXPLAIN ANALYZE 문은 MySQL이 쿼리에 시간을 소비하는 위치와 그 이유를 보여주는 프로파일링 도구입니다. Aurora MySQL은 EXPLAIN ANALYZE를 사용하여 쿼리를 계획, 준비 및 실행하는 동시에 행 수를 세고 실행 계획의 다양한 시점에서 소요되는 시간을 측정합니다. 쿼리가 완료되면 EXPLAIN ANALYZE는 쿼리 결과 대신 계획과 측정값을 인쇄합니다.
- ANALYZE 문을 사용하여 스키마 통계를 최신 상태로 유지하세요. 오래된 통계 때문에 쿼리 최적화 프로그램이 잘못된 실행 계획을 선택하는 경우가 있습니다. 이 경우 테이블과 인덱스 모두의 카디널리티 추정치가 부정확하기 때문에 쿼리 성능이 저하될 수 있습니다. innodb_table_stats 테이블의 last_update 열에는 스키마 통계가 마지막으로 업데이트된 시간이 표시되며, 이는 '무효함'을 나타내는 좋은 지표입니다.
- 데이터 분포 편차와 같이 테이블 카디널리티에 고려되지 않는 다른 문제가 발생할 수 있습니다. 자세한 내용은 MySQL 설명서에서 Estimating ANALYZE TABLE complexity for InnoDB tables 및 Histogram statistics in MySQL을 참조하세요.

쿼리에 소요되는 시간 이해

쿼리에 소요되는 시간을 확인하는 방법은 다음과 같습니다.

- 프로파일링
- 성능 스키마
- 쿼리 최적화 프로그램

프로파일링

기본적으로 프로파일링은 비활성화되어 있습니다. 프로파일링을 활성화한 다음 느린 쿼리를 실행하고 해당 프로필을 검토하세요.

```
SET profiling = 1;
Run your query.
SHOW PROFILE;
```

1. 시간이 가장 많이 소요되는 단계를 식별합니다. MySQL 설명서의 General thread states에 따르면 SELECT 문의 행 읽기 및 처리는 주어진 쿼리의 수명 동안 실행 상태가 가장 긴 경우가 많습니다. EXPLAIN 문을 사용하여 MySQL이 이 쿼리를 실행하는 방식을 이해할 수 있습니다.
2. 느린 쿼리 로그를 검토하여 각 환경의 워크로드가 비슷한지 확인하기 위해 rows_examined 및 rows_sent를 평가합니다. 자세한 내용은 Aurora MySQL 데이터베이스에 대한 로깅 단원을 참조하십시오.
3. 식별된 쿼리에 속하는 테이블에 대해 다음 명령을 실행합니다.

   ```
   SHOW TABLE STATUS\G;
   ```

4. 각 환경에서 쿼리를 실행하기 전과 후에 다음 출력을 캡처합니다.

   ```
   SHOW GLOBAL STATUS;
   ```

5. 각 환경에서 다음 명령을 실행하여 이 샘플 쿼리의 성능에 영향을 미치는 다른 쿼리/세션이 있는지 확인합니다.

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   서버의 리소스가 사용량이 많으면 쿼리를 포함하여 서버의 다른 모든 작업에 영향을 미치는 경우가 있습니다. 또한 쿼리가 실행될 때 정기적으로 정보를 캡처하거나 적절한 간격으로 정보를 캡처하도록 cron 작업을 설정할 수 있습니다.

성능 스키마

성능 스키마는 성능에 미치는 영향을 최소화하면서 서버 런타임 성능에 대한 유용한 정보를 제공합니다. 이는 DB 인스턴스에 대한 스키마 정보를 제공하는 information_schema와 다릅니다. 자세한 내용은 Aurora MySQL에서 성능 개선 도우미의 성능 스키마 개요 단원을 참조하십시오.

쿼리 최적화 프로그램 트레이스

특정 [쿼리 계획을 실행하도록 선택]한 이유를 이해하려면 MySQL 쿼리 최적화 프로그램에 액세스하도록 optimizer_trace를 설정할 수 있습니다.

최적화 프로그램 트레이스를 실행하여 최적화 프로그램이 사용할 수 있는 모든 경로와 선택에 대한 광범위한 정보를 표시합니다.

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

쿼리 최적화 프로그램 설정 검토

Aurora MySQL 버전 3(MySQL 8.0 Community Edition과 호환)은 Aurora MySQL 버전 2(MySQL 5.7 Community Edition과 호환)에 비해 최적화 프로그램 관련 변경 사항이 많습니다. optimizer_switch에 대한 사용자 지정 값이 있는 경우 기본값의 차이를 검토하고 워크로드에 가장 적합한 optimizer_switch 값을 설정하는 것이 좋습니다. 또한 Aurora MySQL 버전 3에서 사용할 수 있는 옵션을 테스트하여 쿼리가 어떻게 수행되는지 검사하는 것이 좋습니다.

> **Note**
>
> Aurora MySQL 버전 3은 [innodb_stats_persistent_sample_pages] 파라미터에 커뮤니티 기본값인 20을 사용합니다.

이러한 명령을 사용하여 optimizer_switch 값을 표시할 수 있습니다.

```
SELECT @@optimizer_switch\G;
```

다음 테이블은 Aurora MySQL 버전 2와 3에 대한 optimizer_switch 기본값을 보여줍니다.

설정	Aurora MySQL 버전 2	Aurora MySQL 버전 3
batched_key_access	꺼	꺼
block_nested_loop	켜짐	켜짐
condition_fanout_filter	켜짐	켜짐
derived_condition_pushdown	–	켜짐
derived_merge	켜짐	켜짐
duplicateweedout	켜짐	켜짐
engine_condition_pushdown	켜짐	켜짐
firstmatch	켜짐	켜짐
hash_join	꺼	켜짐
hash_join_cost_based	켜짐	–
hypergraph_optimizer	–	꺼
index_condition_pushdown	켜짐	켜짐
index_merge	켜짐	켜짐
index_merge_intersection	켜짐	켜짐
index_merge_sort_union	켜짐	켜짐
index_merge_union	켜짐	켜짐
loosescan	켜짐	켜짐
materialization	켜짐	켜짐
mrr	켜짐	켜짐
mrr_cost_based	켜짐	켜짐

설정	Aurora MySQL 버전 2	Aurora MySQL 버전 3
prefer_ordering_index	켜짐	켜짐
semijoin	켜짐	켜짐
skip_scan	–	켜짐
subquery_materialization_cost_based	켜짐	켜짐
subquery_to_derived	–	꺼
use_index_extensions	켜짐	켜짐
use_invisible_indexes	–	꺼

자세한 내용은 MySQL 설명서의 Switchable optimizations (MySQL 5.7) 및 Switchable optimizations (MySQL 8.0)를 참조하세요.

Amazon Aurora MySQL 참조

이 참조에는 Aurora MySQL 파라미터, 상태 변수 및 일반 SQL 확장 또는 커뮤니티 MySQL 데이터베이스 엔진과의 차이점에 대한 정보가 포함되어 있습니다.

주제

Aurora MySQL 구성 파라미터

Amazon Aurora MySQL DB 클러스터는 다른 Amazon RDS DB 인스턴스와 마찬가지로 DB 파라미터 그룹의 파라미터를 사용하여 관리합니다. Amazon Aurora는 DB 클러스터가 다수의 DB 인스턴스로 구성되는 다른 DB 엔진들과 다릅니다. 결과적으로 Aurora MySQL DB 클러스를 관리하기 위해 사용하는 파라미터 중 일부는 전체 클러스터에 적용됩니다. 다른 파라미터는 DB 클러스터의 특정 DB 인스턴스에만 적용됩니다.

클러스터 수준 파라미터를 관리하려면 DB 클러스터 파라미터 그룹을 사용합니다. 인스턴스 수준 파라미터를 관리하려면 DB 파라미터 그룹을 사용합니다. Aurora MySQL DB 클러스터의 각 DB 인스턴스는 MySQL 데이터베이스 엔진과 호환됩니다. 그러나 MySQL 데이터베이스 엔진 파라미터 중 일부는 클러스터 수준에서 적용하며 이러한 파라미터는 DB 클러스터 파라미터그룹을 사용하여 관리합니다. Aurora DB 클러스터에 있는 인스턴스의 DB 파라미터 그룹에서는 클러스터 수준 파라미터를 찾을 수 없습니다. 클러스터 수준 파라미터의 목록은 이 단원 후반부에 나옵니다.

클러스터 수준 파라미터와 인스턴스 수준 파라미터는 모두 AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 관리할 수 있습니다. 클러스터 수준 파라미터와 인스턴스 수준 파라미터를 관리하기 위한 명령은 서로 다릅니다. 예를 들어 modify-db-cluster-parameter-group CLI 명령을 사용해 DB 클러스터 파라미터 그룹에서 클러스터 수준 파라미터를 관리할 수 있습니다. modify-db-parameter-group CLI 명령을 사용해 DB 클러스터 내 DB 인스턴스의 DB 파라미터 그룹에서 인스턴스 수준 파라미터를 관리할 수 있습니다.

클러스터 수준 파라미터와 인스턴스 수준 파라미터는 모두 콘솔에서 확인하거나 CLI 또는 RDS API를 사용해 확인할 수 있습니다. 예를 들어 describe-db-cluster-parameters AWS CLI 명령을 사용해 DB 클러스터 파라미터 그룹에서 클러스터 수준 파라미터를 확인할 수 있습니다. describe-db-parameters CLI 명령을 사용해 DB 클러스터 내 DB 인스턴스의 DB 파라미터 그룹에서 인스턴스 수준 파라미터를 확인할 수 있습니다.

> **Note**
>
> 각 기본 파라미터 그룹에는 파라미터 그룹에 있는 모든 파라미터의 기본값이 포함됩니다. 파라미터에 이 값의 "엔진 기본값"이 있는 경우, 실제 기본값은 버전별 MySQL 또는 PostgreSQL 설명서를 참조하세요.
> 달리 명시되지 않는 한, 다음 표에 나열된 파라미터는 Aurora MySQL 버전 2 및 3에 유효합니다.

DB 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오. Aurora Serverless v1 클러스터에 대한 규칙 및 제한 사항은 Aurora Serverless v1 파라미터 그룹 단원을 참조하십시오.

주제

- 클러스터 수준 파라미터
- 인스턴스 수준 파라미터
- Aurora MySQL에 적용되지 않는 MySQL 파라미터

클러스터 수준 파라미터

다음 표에는 전체 Aurora MySQL DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능	참고
`aurora_binlog_read_buffer_size`	예	이진 로그(binlog) 복제를 사용하는 클러스터에만 영향을 줍니다. binlog 복제에 대한 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하세요. Aurora MySQL 버전 3에서 제거되었습니다.

파라미터 이름	수정 가능	참고
aurora_binlog_replication_max_yield_seconds	예	이진 로그(binlog) 복제를 사용하는 클러스터에만 영향을 줍니다. binlog 복제에 대한 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.
aurora_binlog_replication_sec_index_parallel_workers	예	보조 인덱스가 두 개 이상인 대규모 테이블의 트랜잭션을 복제할 때 보조 인덱스 변경 사항을 적용하는 데 사용할 수 있는 총 병렬 스레드 수를 설정합니다. 기본적으로 파라미터는 0(비활성화됨)로 설정됩니다. 이 파라미터는 Aurora MySQL 버전 306 이상에서 사용할 수 있습니다. 자세한 내용은 Aurora MySQL의 이진수 로그 복제 최적화 단원을 참조하십시오.
aurora_binlog_use_large_read_buffer	예	이진 로그(binlog) 복제를 사용하는 클러스터에만 영향을 줍니다. binlog 복제에 대한 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하세요. Aurora MySQL 버전 3에서 제거되었습니다.
aurora_disable_hash_join	예	Aurora MySQL 버전 2.09 이상에서 해시 조인 최적화를 비활성화하려면 이 파라미터를 ON으로 설정하세요. 버전 3에는 지원되지 않습니다. 자세한 내용은 Amazon Aurora MySQL용 병렬 쿼리 단원을 참조하십시오.

파라미터 이름	수정 가능	참고
aurora_enable_replica_log_compression	예	자세한 내용은 Amazon Aurora MySQL 복제를 위한 성능 고려 사항 섹션을 참조하세요. Aurora Global Database의 일부로 포함된 클러스터에는 적용되지 않습니다. Aurora MySQL 버전 3에서 제거되었습니다.
aurora_enable_repl_bin_log_filtering	예	자세한 내용은 Amazon Aurora MySQL 복제를 위한 성능 고려 사항 섹션을 참조하세요. Aurora Global Database의 일부로 포함된 클러스터에는 적용되지 않습니다. Aurora MySQL 버전 3에서 제거되었습니다.
aurora_enable_staggered_replica_restart	예	이 설정은 Aurora MySQL 버전 3에서 사용할 수 있지만 사용되지는 않습니다.
aurora_enable_zdr	예	이 설정은 Aurora MySQL 2.10 이상에서 기본적으로 켜집니다. 자세한 내용은 Amazon Aurora MySQL의 제로 다운타임 다시 시작(ZDR) 섹션을 참조하세요.
aurora_enhanced_binlog	예	Aurora MySQL 버전 3.03.1 이상에서 향상된 binlog를 켜려면 이 파라미터의 값을 1로 설정하세요. 자세한 내용은 Aurora MySQL에 대해 향상된 binlog 설정 단원을 참조하십시오.
aurora_jemalloc_background_thread	예	이 파라미터를 사용하여 백그라운드 스레드가 메모리 유지 관리 작업을 수행할 수 있도록 합니다. 허용되는 값은 `0`(비활성화) 및 1(활성화)입니다. 기본값은 `0`입니다. 이 파라미터는 Aurora MySQL 버전 3.05 이상에 적용됩니다.

파라미터 이름	수정 가능	참고
aurora_jemalloc_dirty_decay_ms	예	이 파라미터를 사용하여 비워진 메모리를 일정 시간(밀리초) 동안 유지할 수 있습니다. 메모리를 유지하면 더 빠르게 재사용할 수 있습니다. 허용 값은 0~18446744073709551615 입니다. 기본값(0)은 모든 메모리를 사용 가능한 메모리로 운영 체제에 반환합니다. 이 파라미터는 Aurora MySQL 버전 3.05 이상에 적용됩니다.
aurora_jemalloc_tcache_enabled	예	이 파라미터를 사용하면 메모리 아레나를 우회하여 스레드 로컬 캐시에서 작은 메모리 요청(최대 32KiB)을 처리할 수 있습니다. 허용되는 값은 0(비활성화) 및 1(활성화)입니다. 기본값은 1입니다. 이 파라미터는 Aurora MySQL 버전 3.05 이상에 적용됩니다.
aurora_load_from_s3_role	예	자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 단원을 참조하십시오. 현재 Aurora MySQL 버전 3에서는 사용할 수 없습니다. aws_default_s3_role 를 사용합니다.

파라미터 이름	수정 가능	참고
aurora_mask_password_hashes_type	예	이 설정은 Aurora MySQL 2.11 이상에서는 기본적으로 켜집니다. 이 설정을 사용하면 느린 쿼리 로그 및 감사 로그에서 Aurora MySQL 암호 해시를 마스킹할 수 있습니다. 허용되는 값은 0 및 1(기본값)입니다. 1로 설정하면 암호가 <secret>으로 기록됩니다. 0으로 설정하면 암호가 해시(#) 값으로 기록됩니다.
aurora_select_into_s3_role	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 단원을 참조하십시오. 현재 Aurora MySQL 버전 3에서는 사용할 수 없습니다. aws_default_s3_role 를 사용합니다.
authentication_kerberos_caseins_cmp	예	authentication_kerberos 플러그인의 대소문자를 구분하지 않는 사용자 이름 비교를 제어합니다. 대소문자를 구분하지 않고 비교하려면 true로 설정합니다. 기본적으로 대소문자를 구분하는 비교가 사용됩니다(false). 자세한 내용은 Aurora MySQL에 Kerberos 인증 사용 단원을 참조하십시오. 이 파라미터는 Aurora MySQL 버전 3.03 이상에서 사용할 수 있습니다.
auto_increment_increment	예	
auto_increment_offset	예	

파라미터 이름	수정 가능	참고
aws_default_lambda_role	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 섹션을 참조하세요.
aws_default_s3_role	예	DB 클러스터에서 LOAD DATA FROM S3, LOAD XML FROM S3 또는 SELECT INTO OUTFILE S3 문을 호출하는 데 사용됩니다. Aurora MySQL 버전 2에서 이 파라미터에 지정된 IAM 역할은 적절한 문에 대해 aurora_load_from_s3_role 또는 aurora_select_into_s3_role 에 IAM 역할이 지정되지 않은 경우에 사용됩니다. Aurora MySQL 버전 3에서는 이 파라미터에 지정된 IAM 역할이 항상 사용됩니다. 자세한 내용은 IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 단원을 참조하십시오.
binlog_backup	예	Aurora MySQL 버전 3.03.1 이상에서 향상된 binlog를 켜려면 이 파라미터의 값을 0으로 설정하세요. 향상된 binlog를 사용하는 경우에만 이 파라미터를 해제할 수 있습니다. 자세한 내용은 Aurora MySQL에 대해 향상된 binlog 설정 단원을 참조하십시오.

파라미터 이름	수정 가능	참고
binlog_checksum	예	이 파라미터가 설정되지 않은 경우 AWS CLI 및 RDS API는 None의 값을 보고합니다. 이 경우 Aurora MySQL에서는 엔진 기본값 (CRC32) 을 사용합니다. 이 설정은 체크섬을 비활성화하는 NONE의 명시적 설정과 다릅니다.
binlog-do-db	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_format	예	자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.
binlog_group_commit_sync_delay	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_group_commit_sync_no_delay_count	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog-ignore-db	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_replication_globaldb	예	Aurora MySQL 버전 3.03.1 이상에서 향상된 binlog를 켜려면 이 파라미터의 값을 0으로 설정하세요. 향상된 binlog를 사용하는 경우에만 이 파라미터를 해제할 수 있습니다. 자세한 내용은 Aurora MySQL에 대해 향상된 binlog 설정 단원을 참조하십시오.
binlog_row_image	아니요	
binlog_row_metadata	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
binlog_row_value_options	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_rows_query_log_events	예	
binlog_transaction_compression	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_transaction_compression_level_zstd	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_transaction_dependency_history_size	예	이 파라미터는 메모리에 보관되고 지정된 행을 마지막으로 수정한 트랜잭션을 조회하는 데 사용되는 행 해시 수의 상한을 설정합니다. 해시 수가 이 상한에 도달하면 기록이 삭제됩니다. 이 파라미터는 Aurora MySQL 버전 2.12 이상 및 버전 3에 적용됩니다.
binlog_transaction_dependency_tracking	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
character-set-client-handshake	예	
character_set_client	예	
character_set_connection	예	
character_set_database	예	
character_set_filesystem	예	
character_set_results	예	
character_set_server	예	

파라미터 이름	수정 가능	참고
collation_connection	예	
collation_server	예	
completion_type	예	
default_storage_engine	아니요	Aurora MySQL 클러스터는 모든 데이터에 InnoDB 스토리지 엔진을 사용합니다.
enforce_gtid_consistency	가끔	Aurora MySQL 버전 2 이상에서 수정 가능.
event_scheduler	예	이벤트 스케줄러의 상태를 나타냅니다. Aurora MySQL 버전 3에서는 클러스터 수준에서만 수정할 수 있습니다.
gtid-mode	가끔	Aurora MySQL 버전 2 이상에서 수정 가능.
information_schema_stats_expiry	예	MySQL 데이터베이스 서버가 스토리지 엔진에서 데이터를 가져와 캐시에 있는 데이터를 교체하는 데 걸리는 시간(초)입니다. 허용 값은 `0~31536000`입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

<table>
<tr><th>파라미터 이름</th><th>수정 가능</th><th>참고</th></tr>
<tr><td><code>init_connect</code></td><td>예</td><td>연결된 각 클라이언트에 대해 서버에서 실행할 명령입니다. 연결 실패를 방지하려면 설정에 큰따옴표(“)를 사용해야 합니다. 예를 들면 다음과 같습니다.
<pre>SET optimizer_switch="hash_join
=off"</pre>
Aurora MySQL 버전 3에서는 <code>CONNECTION_ADMIN</code> 권한이 있는 사용자에게 이 파라미터가 적용되지 않습니다. Aurora 마스터 사용자도 마찬가지입니다. 자세한 내용은 역할 기반 권한 모델 단원을 참조하십시오.</td></tr>
<tr><td><code>innodb_adaptive_hash_index</code></td><td>예</td><td>이 파라미터는 Aurora MySQL 버전 2와 3의 DB 클러스터 수준에서 수정할 수 있습니다.

Adaptive Hash Index는 리더 DB 인스턴스에서 지원되지 않습니다.</td></tr>
</table>

파라미터 이름	수정 가능	참고
innodb_aurora_instant_alter_column_allowed	예	INSTANT 알고리즘이 글로벌 수준의 ALTER COLUMN 작업에 사용할 수 있는지를 제어합니다. 허용되는 값은 다음과 같습니다. • 0 - INSTANT 알고리즘이 ALTER COLUMN 작업에 허용되지 않습니다(OFF). 다른 알고리즘으로 되돌립니다. • 1 - INSTANT 알고리즘이 ALTER COLUMN 작업에 허용됩니다(ON). 이것이 기본값입니다. 자세한 내용은 MySQL 설명서의 [열 작업](#) 섹션을 참조하세요. 이 파라미터는 Aurora MySQL 버전 3.05 이상에 적용됩니다.
innodb_autoinc_lock_mode	예	
innodb_checksums	아니요	Aurora MySQL 버전 3에서 제거되었습니다.
innodb_cmp_per_index_enabled	예	
innodb_commit_concurrency	예	
innodb_data_home_dir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.

파라미터 이름	수정 가능	참고
innodb_deadlock_detect	예	이 옵션은 Aurora MySQL 버전 2.11 이상 및 버전 3에서 고착 감지를 비활성화하는 데 사용됩니다. 동시성이 높은 시스템에서 수많은 스레드가 동일한 잠금을 기다릴 때 고착 감지로 인해 속도가 느려질 수 있습니다. 이 파라미터에 대한 자세한 내용은 MySQL 설명서를 참조하세요.
innodb_default_row_format	예	이 파라미터는 InnoDB 테이블(사용자가 만든 InnoDB 임시 테이블 포함)의 기본 행 형식을 정의합니다. Aurora MySQL 버전 2와 3에 적용됩니다. 값은 DYNAMIC, COMPACT 또는 REDUNDANT. 일 수 있습니다.
innodb_file_per_table	예	이 파라미터는 테이블 스토리지가 구성되는 방식에 영향을 미칩니다. 자세한 내용은 스토리지 조정 단원을 참조하십시오.
innodb_flush_log_at_trx_commit	예	기본값인 1을 그대로 사용하는 것이 좋습니다. Aurora MySQL 버전 3에서 이 파라미터를 1이 아닌 값으로 설정하려면 먼저 innodb_trx_commit_allow_data_loss 의 값을 1로 변경해야 합니다. 자세한 내용은 로그 버퍼를 플러시하는 빈도 구성 단원을 참조하십시오.
innodb_ft_max_token_size	예	
innodb_ft_min_token_size	예	

파라미터 이름	수정 가능	참고
innodb_ft_num_word_optimize	예	
innodb_ft_sort_pll_degree	예	
innodb_online_alter_log_max_size	예	
innodb_optimize_fulltext_only	예	
innodb_page_size	아니요	
innodb_print_all_deadlocks	예	이 파라미터를 켜져 있으면 모든 InnoDB 교착 상태에 대한 정보가 Aurora MySQL 오류 로그에 기록됩니다. 자세한 내용은 Aurora MySQL 교착 상태 최소화 및 문제 해결 단원을 참조하십시오.
innodb_purge_batch_size	예	
innodb_purge_threads	예	
innodb_rollback_on_timeout	예	
innodb_rollback_segments	예	
innodb_spin_wait_delay	예	
innodb_strict_mode	예	
innodb_support_xa	예	Aurora MySQL 버전 3에서 제거되었습니다.
innodb_sync_array_size	예	
innodb_sync_spin_loops	예	

파라미터 이름	수정 가능	참고
innodb_stats_include_delete_marked	예	이 파라미터를 활성화하면 InnoDB는 영구 최적화 프로그램 통계를 계산할 때 삭제 표시된 레코드를 포함합니다. 이 파라미터는 Aurora MySQL 버전 2.12 이상 및 버전 3에 적용됩니다.
innodb_table_locks	예	
innodb_trx_commit_allow_data_loss	예	Aurora MySQL 버전 3에서 innodb_flush_log_at_trx_commit 값을 변경할 수 있도록 이 파라미터를 1로 설정합니다. innodb_trx_commit_allow_data_loss 의 기본값은 0입니다. 자세한 내용은 로그 버퍼를 플러시하는 빈도 구성 단원을 참조하십시오.
innodb_undo_directory	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
internal_tmp_disk_storage_engine	예	내부 임시 테이블에 사용되는 인메모리 스토리지 엔진을 제어합니다. 허용되는 값은 INNODB 및 MYISAM입니다. 이 파라미터는 Aurora MySQL 버전 2에 적용됩니다.
internal_tmp_mem_storage_engine	예	내부 임시 테이블에 사용되는 인메모리 스토리지 엔진을 제어합니다. 허용되는 값은 MEMORY 및 TempTable 입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
key_buffer_size	예	MyISAM 테이블의 키 캐시입니다. 자세한 내용은 keycache->cache_lock 뮤텍스 관련 설명을 참조하세요.
lc_time_names	예	
log_error_suppression_list	예	MySQL 오류 로그에서 로깅되지 않은 오류 코드 목록을 지정합니다. 이렇게 하면 중요하지 않은 특정 오류 조건을 무시하여 오류 로그를 깔끔하게 유지할 수 있습니다. 자세한 내용은 MySQL 설명서의 log_error_suppression_list를 참조하세요. 이 파라미터는 Aurora MySQL 버전 3.03 이상에 적용됩니다.
low_priority_updates	예	INSERT, UPDATE, DELETE 및 LOCK TABLE WRITE 작업은 보류 중인 SELECT 작업이 없을 때까지 대기합니다. 이 파라미터는 테이블 수준 잠금(MyISAM, MEMORY, MERGE)만 사용하는 스토리지 엔진에만 영향을 줍니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
lower_case_table_names	예(Aurora MySQL 버전 2) 클러스터 생성 시에만(Aurora MySQL 버전 3)	Aurora MySQL 버전 2.10 이상 2.x 버전에서는 이 설정을 변경하고 작성자 인스턴스를 재부팅한 후 모든 리더 인스턴스를 재부팅해야 합니다. 자세한 내용은 읽기 가용성을 포함하여 Aurora 클러스터 재부팅 섹션을 참조하세요. Aurora MySQL 버전 3에서 이 파라미터의 값은 클러스터가 생성될 때 영구적으로 설정됩니다. 이 옵션에 기본값이 아닌 값을 사용하는 경우 업그레이드하기 전에 Aurora MySQL 버전 3 사용자 지정 파라미터를 설정하고 버전 3 클러스터를 생성하는 스냅샷 복원 작업 중에 파라미터 그룹을 지정합니다. Aurora MySQL 기반 Aurora Global Database 사용 시 lower_case_table_names 파라미터가 활성화되어 있는 경우 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수 없습니다. 사용할 수 있는 메서드에 대한 자세한 내용은 메이저 버전 업그레이드 섹션을 참조하세요.
master-info-repository	예	Aurora MySQL 버전 3에서 제거되었습니다.
master_verify_checksum	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 source_verify_checksum 을 사용합니다.

파라미터 이름	수정 가능	참고
max_delayed_threads	예	INSERT DELAYED 명령문을 처리할 최대 스레드 수를 설정합니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
max_error_count	예	표시용으로 저장할 오류, 경고 및 참고 메시지의 최대 수입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
max_execution_time	예	SELECT 문의 실행 제한 시간(밀리초)입니다. 값은 0~18446744073709551615 가 될 수 있습니다. 0으로 설정하면 제한 시간이 없습니다. 자세한 내용은 MySQL 설명서의 [max_execution_time](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_max_execution_time)을 참조하세요.
min_examined_row_limit	예	이 파라미터를 사용하면 지정한 개수보다 적은 수의 행을 검사하는 쿼리가 로깅되지 않도록 할 수 있습니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
partial_revokes	아니요	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
preload_buffer_size	예	인덱스를 미리 로드할 때 할당되는 버퍼의 크기입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
query_cache_type	예	Aurora MySQL 버전 3에서 제거되었습니다.

파라미터 이름	수정 가능	참고
`read_only`	예	이 파라미터를 켜면 서버는 복제본 스레드에서 수행한 업데이트를 제외한 어떤 업데이트도 허용하지 않습니다. Aurora MySQL 버전 2에 유효한 값은 다음과 같습니다. • `0` – `OFF` • `1` – `ON` • `{TrueIfReplica}` - 읽기 전용 복제본의 경우 `ON`. 이것이 기본값입니다. • `{TrueIfClusterReplica}` - 리전 간 읽기 전용 복제본, Aurora 글로벌 데이터베이스의 보조 클러스터, 블루/그린 배포 등의 복제본 클러스터의 경우 `ON`. Aurora MySQL 버전 3에 유효한 값은 다음과 같습니다. • `0` – `OFF`. 기본값입니다. • `1` – `ON` • `{TrueIfClusterReplica}` - 리전 간 읽기 전용 복제본, Aurora 글로벌 데이터베이스의 보조 클러스터, 블루/그린 배포 등의 복제본 클러스터의 경우 `ON`. Aurora MySQL 버전 3에서는 `CONNECTION_ADMIN` 권한이 있는 사용자에게 이 파라미터가 적용되지 않습니다. Aurora 마스터 사용자도 마찬가지

파라미터 이름	수정 가능	참고
		입니다. 자세한 내용은 역할 기반 권한 모델 단원을 참조하십시오.
`relay-log-space-limit`	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
`replica_parallel_type`	예	이 파라미터를 사용하면 이미 준비 단계에 있는 모든 커밋되지 않은 스레드의 복제본에서 일관성을 침해하지 않고 병렬 실행을 수행할 수 있습니다. Aurora MySQL 버전 3에 적용됩니다. Aurora MySQL 버전 3.03.* 이하에서는 기본값이 DATABASE입니다. Aurora MySQL 버전 3.04 이상에서는 기본값이 LOGICAL_CLOCK입니다.
`replica_preserve_commit_order`	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
`replica_transaction_retries`	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
`replica_type_conversions`	예	이 파라미터는 복제본에 사용되는 유형 변환을 결정합니다. 허용되는 값은 `ALL_LOSSY` , `ALL_NON_LOSSY` , `ALL_SIGNED` , `ALL_UNSIGNED` 입니다. 자세한 내용은 MySQL 설명서의 소스 및 복제본에서 서로 다른 테이블 정의를 사용하는 복제를 참조하세요. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
`replicate-do-db`	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
replicate-do-table	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
replicate-ignore-db	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
replicate-ignore-table	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
replicate-wild-do-table	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
replicate-wild-ignore-table	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
require_secure_transport	예	이 파라미터는 Aurora MySQL 버전 2와 3에 적용됩니다. 자세한 내용은 Aurora MySQL DB 클러스터에 대한 TLS 연결 단원을 참조하십시오.
rpl_read_size	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
server_audit_cw_upload	아니요	이 파라미터는 Aurora MySQL에서 더 이상 사용되지 않습니다. server_audit_logs_upload 를 사용합니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
server_audit_events	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.

파라미터 이름	수정 가능	참고
server_audit_excl_users	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.
server_audit_incl_users	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.
server_audit_logging	예	로그를 Amazon CloudWatch Logs에 업로드하는 방법에 관한 지침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
server_audit_logs_upload	예	고급 감사를 활성화하고 이 파라미터를 1로 설정하여 CloudWatch Logs에 감사 로그를 게시할 수 있습니다. server_audit_logs_upload 파라미터의 기본값은 0입니다. 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
server_id	아니요	
skip-character-set-client-handshake	예	
skip_name_resolve	아니요	
slave-skip-errors	예	MySQL 5.7 호환성을 갖춘 Aurora MySQL 버전 2 클러스터에만 적용됩니다.
source_verify_checksum	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
sync_frm	예	Aurora MySQL 버전 3에서 제거되었습니다.
thread_cache_size	예	캐시할 스레드 수입니다. 이 파라미터는 Aurora MySQL 버전 2와 3에 적용됩니다.
time_zone	예	기본적으로 Aurora DB 클러스터의 시간대는 협정 세계시(UTC)입니다. 대신 DB 클러스터의 인스턴스 시간대를 애플리케이션의 현지 시간대로 설정할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터의 현지 시간대 단원을 참조하십시오.
tls_version	예	자세한 내용은 Aurora MySQL에 지원되는 TLS 버전 섹션을 참조하세요.

인스턴스 수준 파라미터

다음 표에는 Aurora MySQL DB 클러스터의 특정 DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능	참고
activate_all_roles_on_login	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
allow-suspicious-udfs	아니요	
aurora_disable_hash_join	예	Aurora MySQL 버전 2.09 이상에서 해시 조인 최적화를 비활성화하려면 이 파라미터를 ON으로 설정하세요. 버전 3에는 지원되지 않습니다. 자세한 내용은

파라미터 이름	수정 가능	참고
		Amazon Aurora MySQL용 병렬 쿼리 단원을 참조하십시오.
aurora_lab_mode	예	자세한 내용은 Amazon Aurora MySQL 랩 모드 단원을 참조하십시오. Aurora MySQL 버전 3에서 제거되었습니다.
aurora_oom_response	예	이 파라미터는 Aurora MySQL 버전 2 및 3에서 지원됩니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 단원을 참조하십시오.
aurora_parallel_query	예	Aurora MySQL 버전 2.09 이상에서 병렬 쿼리를 켜려면 ON으로 설정합니다. 이전 aurora_pq 파라미터는 이러한 버전에서 사용되지 않습니다. 자세한 내용은 Amazon Aurora MySQL용 병렬 쿼리 섹션을 참조하세요.
aurora_pq	예	2.09 이전 Aurora MySQL 버전에서 특정 DB 인스턴스에 대한 병렬 쿼리를 해제하려면 OFF로 설정합니다. 버전 2.09 이상에서는 aurora_parallel_query 를 사용하여 병렬 쿼리를 설정하거나 해제합니다. 자세한 내용은 Amazon Aurora MySQL용 병렬 쿼리 단원을 참조하십시오.

파라미터 이름	수정 가능	참고
aurora_read_replica_read_committed	예	Aurora 복제본에 대해 READ COMMITTED 격리 수준을 활성화하고 격리 동작을 변경하여 장기 실행 쿼리 중 삭제 지연 시간을 줄입니다. 이 설정은 동작 변경과 이러한 변경이 쿼리 결과에 미치는 영향을 이해한 경우에만 활성화하십시오. 예를 들어 이 설정에서는 MySQL 기본값보다 덜 엄격한 격리를 사용합니다. 이 설정이 활성화된 경우 쿼리 실행 중 Aurora에서 테이블 데이터를 재편하므로 장기 실행 쿼리에서는 동일 행의 사본을 두 개 이상 반환할 수 있습니다. 자세한 내용은 Aurora MySQL 격리 수준 단원을 참조하십시오.
aurora_tmptable_enable_per_table_limit	예	tmp_table_size 파라미터가 Aurora MySQL 버전 3.04 이상의 TempTable 스토리지 엔진에서 생성된 인 메모리 임시 테이블의 최대 크기를 제어하는지를 결정합니다. 자세한 내용은 내부 인 메모리 임시 테이블의 크기 제한 단원을 참조하십시오.
aurora_use_vector_instructions	예	이 파라미터가 활성화되면 Aurora MySQL은 최신 CPU에서 제공하는 최적화된 벡터 처리 명령을 사용하여 I/O 집약적인 워크로드의 성능을 개선합니다. 이 설정은 Aurora MySQL 3.05 이상에서는 기본적으로 활성화됩니다.
autocommit	예	
automatic_sp_privileges	예	

파라미터 이름	수정 가능	참고
back_log	예	
basedir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
binlog_cache_size	예	
binlog_max_flush_queue_time	예	
binlog_order_commits	예	
binlog_stmt_cache_size	예	
binlog_transaction_compression	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
binlog_transaction_compression_level_zstd	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
bulk_insert_buffer_size	예	
concurrent_insert	예	
connect_timeout	예	
core-file	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
datadir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
default_authentication_plugin	아니요	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.

파라미터 이름	수정 가능	참고
default_time_zone	아니요	
default_tmp_storage_engine	예	임시 테이블의 기본 스토리지 엔진입니다.
default_week_format	예	
delay_key_write	예	
delayed_insert_limit	예	
delayed_insert_timeout	예	
delayed_queue_size	예	
div_precision_increment	예	
end_markers_in_json	예	
eq_range_index_dive_limit	예	
event_scheduler	가끔	이벤트 스케줄러의 상태를 나타냅니다. Aurora MySQL 버전 3에서는 클러스터 수준에서만 수정할 수 있습니다.
explicit_defaults_for_timestamp	예	
flush	아니요	
flush_time	예	
ft_boolean_syntax	아니요	
ft_max_word_len	예	
ft_min_word_len	예	

파라미터 이름	수정 가능	참고
ft_query_expansion_limit	예	
ft_stopword_file	예	
general_log	예	로그를 CloudWatch Logs에 업로드하는 것에 관한 지침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
general_log_file	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
group_concat_max_len	예	
host_cache_size	예	
init_connect	예	연결된 각 클라이언트에 대해 서버에서 실행할 명령입니다. 연결 실패를 방지하려면 설정에 큰따옴표(“)를 사용해야 합니다. 예를 들면 다음과 같습니다. `SET optimizer_switch="hash_join =off"` Aurora MySQL 버전 3에서는 Aurora 마스터 사용자를 비롯하여 CONNECTION_ADMIN 권한이 있는 사용자에게 이 파라미터가 적용되지 않습니다. 자세한 내용은 역할 기반 권한 모델 단원을 참조하십시오.

파라미터 이름	수정 가능	참고
innodb_adaptive_hash_index	예	이 파라미터는 Aurora MySQL 버전 2의 DB 인스턴스 수준에서 수정할 수 있습니다. Aurora MySQL 버전 3에서는 DB 클러스터 수준에서만 수정할 수 있습니다. Adaptive Hash Index는 리더 DB 인스턴스에서 지원되지 않습니다.
innodb_adaptive_max_sleep_delay	예	Aurora에 대해 innodb_thread_concurrency 는 항상 0이므로 이 파라미터를 수정해도 효과가 없습니다.
innodb_aurora_max_partitions_for_range	예	영구 통계를 사용할 수 없는 경우 이 파라미터를 사용하여 파티션으로 분할된 테이블의 행 수 추정 성능을 개선할 수 있습니다. 0~8,192 범위의 값으로 설정할 수 있으며, 이 값에 따라 행 수를 추정하는 동안 검사할 파티션 수가 결정됩니다. 기본값은 0이며, 이 값은 기본 MySQL 동작과 일치하게 모든 파티션을 사용하여 추정합니다. 이 파라미터는 Aurora MySQL 버전 3.03.1 이상에서 사용할 수 있습니다.
innodb_autoextend_increment	예	
innodb_buffer_pool_dump_at_shutdown	아니요	
innodb_buffer_pool_dump_now	아니요	

파라미터 이름	수정 가능	참고
innodb_buffer_pool_filename	아니요	
innodb_buffer_pool_load_abort	아니요	
innodb_buffer_pool_load_at_startup	아니요	
innodb_buffer_pool_load_now	아니요	
innodb_buffer_pool_size	예	기본값은 공식으로 표시됩니다. 수식에서 DBInstanceClassMemory 값을 계산하는 방법에 대한 자세한 내용은 DB 파라미터 수식 변수 섹션을 참조하세요.
innodb_change_buffer_max_size	아니요	Aurora MySQL은 InnoDB 변경 버퍼를 전혀 사용하지 않습니다.
innodb_compression_failure_threshold_pct	예	
innodb_compression_level	예	
innodb_compression_pad_pct_max	예	
innodb_concurrency_tickets	예	Aurora에 대해 innodb_thread_concurrency 는 항상 0이므로 이 파라미터를 수정해도 효과가 없습니다.

파라미터 이름	수정 가능	참고
innodb_deadlock_detect	예	이 옵션은 Aurora MySQL 버전 2.11 이상 및 버전 3에서 고착 감지를 비활성화하는 데 사용됩니다. 동시성이 높은 시스템에서 수많은 스레드가 동일한 잠금을 기다릴 때 고착 감지로 인해 속도가 느려질 수 있습니다. 이 파라미터에 대한 자세한 내용은 MySQL 설명서를 참조하세요.
innodb_file_format	예	Aurora MySQL 버전 3에서 제거되었습니다.
innodb_flushing_avg_loops	아니요	
innodb_force_load_corrupted	아니요	
innodb_ft_aux_table	예	
innodb_ft_cache_size	예	
innodb_ft_enable_stopword	예	
innodb_ft_server_stopword_table	예	
innodb_ft_user_stopword_table	예	
innodb_large_prefix	예	Aurora MySQL 버전 3에서 제거되었습니다.
innodb_lock_wait_timeout	예	
innodb_log_compressed_pages	아니요	

파라미터 이름	수정 가능	참고
innodb_lru_scan_depth	예	
innodb_max_purge_lag	예	
innodb_max_purge_lag_delay	예	
innodb_monitor_disable	예	
innodb_monitor_enable	예	
innodb_monitor_reset	예	
innodb_monitor_reset_all	예	
innodb_old_blocks_pct	예	
innodb_old_blocks_time	예	
innodb_open_files	예	
innodb_print_all_deadlocks	예	이 파라미터를 켜져 있으면 모든 InnoDB 교착 상태에 대한 정보가 Aurora MySQL 오류 로그에 기록됩니다. 자세한 내용은 Aurora MySQL 교착 상태 최소화 및 문제 해결 단원을 참조하십시오.
innodb_random_read_ahead	예	
innodb_read_ahead_threshold	예	
innodb_read_io_threads	아니요	

파라미터 이름	수정 가능	참고
innodb_read_only	아니요	Aurora MySQL은 클러스터 유형을 기반으로 읽기 전용 및 읽기-쓰기 상태의 DB 인스턴스를 관리합니다. 예를 들어 프로비저닝된 클러스터에는 읽기-쓰기 DB 인스턴스(기본 인스턴스) 하나가 있으며, 클러스터에 있는 기타 인스턴스는 읽기 전용(Aurora 복제본)입니다.
innodb_replication_delay	예	
innodb_sort_buffer_size	예	
innodb_stats_auto_recalc	예	
innodb_stats_method	예	
innodb_stats_on_metadata	예	
innodb_stats_persistent	예	
innodb_stats_persistent_sample_pages	예	
innodb_stats_transient_sample_pages	예	
innodb_thread_concurrency	아니요	
innodb_thread_sleep_delay	예	Aurora에 대해 innodb_thread_concurrency 는 항상 0이므로 이 파라미터를 수정해도 효과가 없습니다.
interactive_timeout	예	Aurora는 interactive_timeout 및 wait_timeout 의 최소값을 평가합니다. 그런 다음, 해당 최소값을 시간 초과로 사용하여 대화형 및 비대화형 모든 유휴 세션을 종료합니다.

파라미터 이름	수정 가능	참고
internal_tmp_disk_storage_engine	예	내부 임시 테이블에 사용되는 인메모리 스토리지 엔진을 제어합니다. 허용되는 값은 INNODB 및 MYISAM입니다. 이 파라미터는 Aurora MySQL 버전 2에 적용됩니다.
internal_tmp_mem_storage_engine	예	내부 임시 테이블에 사용되는 인메모리 스토리지 엔진을 제어합니다. 허용되는 값은 MEMORY 및 TempTable 입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
join_buffer_size	예	
keep_files_on_create	예	
key_buffer_size	예	MyISAM 테이블의 키 캐시입니다. 자세한 내용은 keycache->cache_lock 뮤텍스 관련 설명을 참조하세요.
key_cache_age_threshold	예	
key_cache_block_size	예	
key_cache_division_limit	예	
local_infile	예	
lock_wait_timeout	예	

파라미터 이름	수정 가능	참고
log-bin	아니요	binlog_format 을 STATEMENT , MIXED 또는 ROW로 설정하면 log-bin이 자동으로 ON으로 설정됩니다. binlog_format 을 OFF로 설정하면 log-bin이 자동으로 OFF로 설정됩니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 섹션을 참조하세요.
log_bin_trust_function_creators	예	
log_bin_use_v1_row_events	예	Aurora MySQL 버전 3에서 제거되었습니다.
log_error	아니요	
log_error_suppression_list	예	MySQL 오류 로그에서 로깅되지 않은 오류 코드 목록을 지정합니다. 이렇게 하면 중요하지 않은 특정 오류 조건을 무시하여 오류 로그를 깔끔하게 유지할 수 있습니다. 자세한 내용은 MySQL 설명서의 log_error_suppression_list를 참조하세요. 이 파라미터는 Aurora MySQL 버전 3.03 이상에 적용됩니다.
log_output	예	
log_queries_not_using_indexes	예	

파라미터 이름	수정 가능	참고
log_slave_updates	아니요	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 log_replica_updates 을 사용합니다.
log_replica_updates	아니요	Aurora MySQL 버전 3
log_throttle_queries_not_using_indexes	예	
log_warnings	예	Aurora MySQL 버전 3에서 제거되었습니다.
long_query_time	예	
low_priority_updates	예	INSERT, UPDATE, DELETE 및 LOCK TABLE WRITE 작업은 보류 중인 SELECT 작업이 없을 때까지 대기합니다. 이 파라미터는 테이블 수준 잠금(MyISAM, MEMORY, MERGE)만 사용하는 스토리지 엔진에만 영향을 줍니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
max_allowed_packet	예	
max_binlog_cache_size	예	
max_binlog_size	아니요	
max_binlog_stmt_cache_size	예	
max_connect_errors	예	

파라미터 이름	수정 가능	참고
max_connections	예	기본값은 공식으로 표시됩니다. 수식에서 DBInstanceClassMemory 값을 계산하는 방법에 대한 자세한 내용은 DB 파라미터 수식 변수 섹션을 참조하세요. 인스턴스 클래스에 따른 기본값은 Aurora MySQL DB 인스턴스에 대한 최대 연결 섹션을 참조하세요.
max_delayed_threads	예	INSERT DELAYED 명령문을 처리할 최대 스레드 수를 설정합니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
max_error_count	예	표시용으로 저장할 오류, 경고 및 참고 메시지의 최대 수입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
max_execution_time	예	SELECT 문의 실행 제한 시간(밀리초)입니다. 값은 0~18446744073709551615 가 될 수 있습니다. 0으로 설정하면 제한 시간이 없습니다. 자세한 내용은 MySQL 설명서의 max_execution_time을 참조하세요.
max_heap_table_size	예	
max_insert_delayed_threads	예	
max_join_size	예	
max_length_for_sort_data	예	Aurora MySQL 버전 3에서 제거되었습니다.

파라미터 이름	수정 가능	참고
max_prepared_stmt_count	예	
max_seeks_for_key	예	
max_sort_length	예	
max_sp_recursion_depth	예	
max_tmp_tables	예	Aurora MySQL 버전 3에서 제거되었습니다.
max_user_connections	예	
max_write_lock_count	예	
metadata_locks_cache_size	예	Aurora MySQL 버전 3에서 제거되었습니다.
min_examined_row_limit	예	이 파라미터를 사용하면 지정한 개수보다 적은 수의 행을 검사하는 쿼리가 로깅되지 않도록 할 수 있습니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
myisam_data_pointer_size	예	
myisam_max_sort_file_size	예	
myisam_mmap_size	예	
myisam_sort_buffer_size	예	
myisam_stats_method	예	
myisam_use_mmap	예	
net_buffer_length	예	

파라미터 이름	수정 가능	참고
net_read_timeout	예	
net_retry_count	예	
net_write_timeout	예	
old-style-user-limits	예	
old_passwords	예	Aurora MySQL 버전 3에서 제거되었습니다.
optimizer_prune_level	예	
optimizer_search_depth	예	
optimizer_switch	예	이 스위치를 사용하는 Aurora MySQL 기능에 대한 자세한 내용은 Amazon Aurora MySQL 모범 사례 단원을 참조하십시오.
optimizer_trace	예	
optimizer_trace_features	예	
optimizer_trace_limit	예	
optimizer_trace_max_mem_size	예	
optimizer_trace_offset	예	
performance-schema-consumer-events-waits-current	예	
performance-schema-instrument	예	
performance_schema	예	

파라미터 이름	수정 가능	참고
performance_schema_accounts_size	예	
performance_schema_consumer_global_instrumentation	예	
performance_schema_consumer_thread_instrumentation	예	
performance_schema_consumer_events_stages_current	예	
performance_schema_consumer_events_stages_history	예	
performance_schema_consumer_events_stages_history_long	예	
performance_schema_consumer_events_statements_current	예	
performance_schema_consumer_events_statements_history	예	
performance_schema_consumer_events_statements_history_long	예	
performance_schema_consumer_events_waits_history	예	
performance_schema_consumer_events_waits_history_long	예	
performance_schema_consumer_statements_digest	예	

파라미터 이름	수정 가능	참고
performance_schema_digests_size	예	
performance_schema_events_stages_history_long_size	예	
performance_schema_events_stages_history_size	예	
performance_schema_events_statements_history_long_size	예	
performance_schema_events_statements_history_size	예	
performance_schema_events_transactions_history_long_size	예	
performance_schema_events_transactions_history_size	예	
performance_schema_events_waits_history_long_size	예	
performance_schema_events_waits_history_size	예	
performance_schema_hosts_size	예	
performance_schema_max_cond_classes	예	
performance_schema_max_cond_instances	예	

파라미터 이름	수정 가능	참고
performance_schema_max_digest_length	예	
performance_schema_max_file_classes	예	
performance_schema_max_file_handles	예	
performance_schema_max_file_instances	예	
performance_schema_max_index_stat	예	
performance_schema_max_memory_classes	예	
performance_schema_max_metadata_locks	예	
performance_schema_max_mutex_classes	예	
performance_schema_max_mutex_instances	예	
performance_schema_max_prepared_statements_instances	예	
performance_schema_max_program_instances	예	
performance_schema_max_rwlock_classes	예	
performance_schema_max_rwlock_instances	예	

파라미터 이름	수정 가능	참고
performance_schema_max_sock et_classes	예	
performance_schema_max_sock et_instances	예	
performance_schema_max_sql_ text_length	예	
performance_schema_max_stag e_classes	예	
performance_schema_max_stat ement_classes	예	
performance_schema_max_stat ement_stack	예	
performance_schema_max_tabl e_handles	예	
performance_schema_max_tabl e_instances	예	
performance_schema_max_tabl e_lock_stat	예	
performance_schema_max_thre ad_classes	예	
performance_schema_max_thre ad_instances	예	
performance_schema_session_ connect_attrs_size	예	
performance_schema_setup_ac tors_size	예	

파라미터 이름	수정 가능	참고
performance_schema_setup_objects_size	예	
performance_schema_show_processlist	예	이 파라미터는 어떤 SHOW PROCESSLIST 구현을 사용할지를 결정합니다. • 기본 구현은 글로벌 뮤텍스를 보유하면서 스레드 관리자 내에서 활성 스레드를 반복합니다. 이로 인해 특히 사용량이 많은 시스템에서 성능이 저하될 수 있습니다. • 대안인 SHOW PROCESSLIST 구현은 성능 스키마 processlist 테이블을 기반으로 합니다. 이 구현은 스레드 관리자가 아닌 성능 스키마에서 활성 스레드 데이터를 쿼리하며 뮤텍스가 필요하지 않습니다. 이 파라미터는 Aurora MySQL 버전 2.12 이상 및 버전 3에 적용됩니다.
performance_schema_users_size	예	
pid_file	아니요	
plugin_dir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
port	아니요	Aurora MySQL은 연결 속성을 관리하여 클러스터에 속한 모든 DB 인스턴스에 일관된 설정을 적용합니다.

파라미터 이름	수정 가능	참고
preload_buffer_size	예	인덱스를 미리 로드할 때 할당되는 버퍼의 크기입니다. 이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
profiling_history_size	예	
query_alloc_block_size	예	
query_cache_limit	예	Aurora MySQL 버전 3에서 제거되었습니다.
query_cache_min_res_unit	예	Aurora MySQL 버전 3에서 제거되었습니다.
query_cache_size	예	기본값은 공식으로 표시됩니다. 수식에서 DBInstanceClassMemory 값을 계산하는 방법에 대한 자세한 내용은 DB 파라미터 수식 변수 섹션을 참조하세요. Aurora MySQL 버전 3에서 제거되었습니다.
query_cache_type	예	Aurora MySQL 버전 3에서 제거되었습니다.
query_cache_wlock_invalidate	예	Aurora MySQL 버전 3에서 제거되었습니다.
query_prealloc_size	예	
range_alloc_block_size	예	
read_buffer_size	예	

파라미터 이름	수정 가능	참고
read_only	예	이 파라미터를 켜면 서버는 복제본 스레드에서 수행한 업데이트를 제외한 어떤 업데이트도 허용하지 않습니다. Aurora MySQL 버전 2에 유효한 값은 다음과 같습니다. • 0 – OFF • 1 – ON • {TrueIfReplica} - 읽기 전용 복제본의 경우 ON. 이것이 기본값입니다. • {TrueIfClusterReplica} - 리전 간 읽기 전용 복제본, Aurora 글로벌 데이터베이스의 보조 클러스터, 블루/그린 배포 등의 복제본 클러스터 인스턴스의 경우 ON. 장애 조치 시 read_only 파라미터가 새 라이터 인스턴스에 적용되도록 하려면 Aurora MySQL 버전 2의 DB 클러스터 파라미터 그룹을 사용하는 것이 좋습니다. **Note** Aurora MySQL은 모든 리더에 대해 innodb_read_only 를 1로 설정하므로 리더 인스턴스는 항상 읽기 전용입니다. 따라서 read_only 는 리더 인스턴스에서 중복됩니다.

파라미터 이름	수정 가능	참고
		Aurora MySQL 버전 3에서 인스턴스 수준에서 제거되었습니다.
`read_rnd_buffer_size`	예	
`relay-log`	아니요	
`relay_log_info_repository`	예	Aurora MySQL 버전 3에서 제거되었습니다.
`relay_log_recovery`	아니요	
`replica_checkpoint_group`	예	Aurora MySQL 버전 3
`replica_checkpoint_period`	예	Aurora MySQL 버전 3
`replica_parallel_workers`	예	Aurora MySQL 버전 3
`replica_pending_jobs_size_max`	예	Aurora MySQL 버전 3
`replica_skip_errors`	예	Aurora MySQL 버전 3
`replica_sql_verify_checksum`	예	Aurora MySQL 버전 3
`safe-user-create`	예	
`secure_auth`	예	이 파라미터는 Aurora MySQL 버전 2에서 항상 활성화되어 있습니다. 비활성화하려고 하면 오류가 발생합니다. Aurora MySQL 버전 3에서 제거되었습니다.
`secure_file_priv`	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.

파라미터 이름	수정 가능	참고
show_create_table_verbosity	예	이 변수를 활성화하면 기본 형식인지와 상관없이 SHOW_CREATE_TABLE이 ROW_FORMAT 을 표시합니다. 이 파라미터는 Aurora MySQL 버전 2.12 이상 및 버전 3에 적용됩니다.
skip-slave-start	아니요	
skip_external_locking	아니요	
skip_show_database	예	
slave_checkpoint_group	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 replica_checkpoint_group 을 사용합니다.
slave_checkpoint_period	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 replica_checkpoint_period 을 사용합니다.
slave_parallel_workers	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 replica_parallel_workers 을 사용합니다.
slave_pending_jobs_size_max	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 replica_pending_jobs_size_max 을 사용합니다.
slave_sql_verify_checksum	예	Aurora MySQL 버전 2 Aurora MySQL 버전 3에서 replica_sql_verify_checksum 을 사용합니다.
slow_launch_time	예	

파라미터 이름	수정 가능	참고
slow_query_log	예	로그를 CloudWatch Logs에 업로드하는 것에 관한 지침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 단원을 참조하십시오.
slow_query_log_file	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
socket	아니요	
sort_buffer_size	예	
sql_mode	예	
sql_select_limit	예	
stored_program_cache	예	
sync_binlog	아니요	
sync_master_info	예	
sync_source_info	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다.
sync_relay_log	예	Aurora MySQL 버전 3에서 제거되었습니다.
sync_relay_log_info	예	
sysdate-is-now	예	
table_cache_element_entry_ttl	아니요	

파라미터 이름	수정 가능	참고
table_definition_cache	예	기본값은 공식으로 표시됩니다. 수식에서 DBInstanceClassMemory 값을 계산하는 방법에 대한 자세한 내용은 DB 파라미터 수식 변수 섹션을 참조하세요.
table_open_cache	예	기본값은 공식으로 표시됩니다. 수식에서 DBInstanceClassMemory 값을 계산하는 방법에 대한 자세한 내용은 DB 파라미터 수식 변수 섹션을 참조하세요.
table_open_cache_instances	예	
temp-pool	예	Aurora MySQL 버전 3에서 제거되었습니다.
temptable_max_mmap	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다. 세부 정보는 Aurora MySQL 버전 3의 새로운 임시 테이블 동작을 참조하세요.
temptable_max_ram	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다. 세부 정보는 Aurora MySQL 버전 3의 새로운 임시 테이블 동작을 참조하세요.
temptable_use_mmap	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다. 세부 정보는 Aurora MySQL 버전 3의 새로운 임시 테이블 동작을 참조하세요.
thread_cache_size	예	캐시할 스레드 수입니다. 이 파라미터는 Aurora MySQL 버전 2와 3에 적용됩니다.
thread_handling	아니요	
thread_stack	예	

파라미터 이름	수정 가능	참고
timed_mutexes	예	
tmp_table_size	예	Aurora MySQL 버전 3의 MEMORY 스토리지 엔진이 생성한 내부 인 메모리 임시 테이블의 최대 크기를 정의합니다. Aurora MySQL 버전 3.04 이상에서는 aurora_tmptable_enable_per_table_limit 이 ON일 때 TempTable 스토리지 엔진이 생성한 내부 인 메모리 임시 테이블의 최대 크기를 정의합니다. 자세한 내용은 내부 인 메모리 임시 테이블의 크기 제한 단원을 참조하십시오.
tmpdir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하므로 사용자가 직접 파일 시스템에 액세스하지 않습니다.
transaction_alloc_block_size	예	
transaction_isolation	예	이 파라미터는 Aurora MySQL 버전 3에 적용됩니다. tx_isolation 을 대체합니다.
transaction_prealloc_size	예	
tx_isolation	예	Aurora MySQL 버전 3에서 제거되었습니다. transaction_isolation 으로 대체됩니다.
updatable_views_with_limit	예	
validate-password	아니요	

파라미터 이름	수정 가능	참고
validate_password_dictionary_file	아니요	
validate_password_length	아니요	
validate_password_mixed_case_count	아니요	
validate_password_number_count	아니요	
validate_password_policy	아니요	
validate_password_special_char_count	아니요	
wait_timeout	예	Aurora는 interactive_timeout 및 wait_timeout 의 최소값을 평가합니다. 그런 다음, 해당 최소값을 시간 초과로 사용하여 대화형 및 비대화형 모든 유휴 세션을 종료합니다.

Aurora MySQL에 적용되지 않는 MySQL 파라미터

Aurora MySQL과 MySQL 간의 아키텍처 차이 때문에 일부 MySQL 파라미터는 Aurora MySQL에 적용되지 않습니다.

다음 MySQL 파라미터는 Aurora MySQL에 적용되지 않습니다. 단, 이 목록이 전부는 아닙니다.

- activate_all_roles_on_login - 이 파라미터는 Aurora MySQL 버전 2에 적용되지 않습니다. Aurora MySQL 버전 3에서 사용할 수 있습니다.
- big_tables
- bind_address
- character_sets_dir
- innodb_adaptive_flushing
- innodb_adaptive_flushing_lwm

- innodb_buffer_pool_chunk_size
- innodb_buffer_pool_instances
- innodb_change_buffering
- innodb_checksum_algorithm
- innodb_data_file_path
- innodb_dedicated_server
- innodb_doublewrite
- innodb_flush_log_at_timeout - 이 파라미터는 Aurora MySQL에 적용되지 않습니다. 자세한 내용은 로그 버퍼를 플러시하는 빈도 구성 단원을 참조하십시오.
- innodb_flush_method
- innodb_flush_neighbors
- innodb_io_capacity
- innodb_io_capacity_max
- innodb_log_buffer_size
- innodb_log_file_size
- innodb_log_files_in_group
- innodb_log_spin_cpu_abs_lwm
- innodb_log_spin_cpu_pct_hwm
- innodb_log_writer_threads
- innodb_max_dirty_pages_pct
- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Aurora MySQL 글로벌 상태 변수

Aurora MySQL에는 커뮤니티 MySQL의 상태 변수와 Aurora에 고유한 변수가 포함되어 있습니다. 이러한 변수를 검사하여 데이터베이스 엔진 내에서 어떤 일이 일어나고 있는지 알아볼 수 있습니다. 커뮤니티 MySQL의 상태 변수에 대한 자세한 내용은 커뮤니티 MySQL 8.0 설명서의 서버 상태 변수를 참조하세요.

다음과 같은 명령문을 사용하여 Aurora MySQL 글로벌 상태 변수의 현재 값을 찾을 수 있습니다.

```
show global status like '%aurora%';
```

다음 테이블에는 Aurora MySQL이 사용하는 글로벌 상태 변수가 설명되어 있습니다.

명칭	설명
AuroraDb_commits	마지막 재시작 이후 총 커밋 수입니다.
AuroraDb_commit_latency	마지막 재시작 이후 집계된 커밋 지연 시간입니다.
AuroraDb_ddl_stmt_duration	마지막 재시작 이후 집계된 DDL 지연 시간입니다.
AuroraDb_select_stmt_duration	마지막 재시작 이후 집계된 SELECT 명령문 지연 시간입니다.
AuroraDb_insert_stmt_duration	마지막 재시작 이후 집계된 INSERT 명령문 지연 시간입니다.
AuroraDb_update_stmt_duration	마지막 재시작 이후 집계된 UPDATE 명령문 지연 시간입니다.
AuroraDb_delete_stmt_duration	마지막 재시작 이후 집계된 DELETE 명령문 지연 시간입니다.
Aurora_binlog_io_cache_allocated	Binlog I/O 캐시에 할당된 바이트 수입니다.
Aurora_binlog_io_cache_read_requests	Binlog I/O 캐시에 전송된 읽기 요청 수입니다.

명칭	설명
Aurora_binlog_io_cache_reads	Binlog I/O 캐시에서 제공된 읽기 요청 수입니다.
Aurora_enhanced_binlog	이 DB 인스턴스에 대해 향상된 binlog를 활성화했는지 아니면 비활성화했는지를 나타냅니다. 자세한 내용은 Aurora MySQL에 대해 향상된 binlog 설정 단원을 참조하십시오.
Aurora_external_connection_count	DB 인스턴스에 대한 데이터베이스 연결 수입니다. 데이터베이스 상태 확인에 사용되는 RDS 서비스 연결은 제외됩니다.
Aurora_fast_insert_cache_hits	캐싱된 커서가 성공적으로 검색 및 확인되면 증가하는 카운터입니다. 고속 삽입 캐시에 대한 자세한 내용은 Amazon Aurora MySQL 성능 개선 사항 섹션을 참조하세요.
Aurora_fast_insert_cache_misses	캐싱된 커서가 더 이상 유효하지 않고 Aurora가 정상적인 인덱스 순회를 수행하면 증가하는 카운터입니다. 고속 삽입 캐시에 대한 자세한 내용은 Amazon Aurora MySQL 성능 개선 사항 섹션을 참조하세요.
Aurora_fts_cache_memory_used	InnoDB 전체 텍스트 검색 시스템이 사용 중인 메모리 양(바이트)입니다. 이 변수는 Aurora MySQL 버전 3.07 이상에 적용됩니다.
Aurora_fwd_master_dml_stmt_count	이 라이터 DB 인스턴스로 전달된 총 DML 명령문 수입니다. 이 변수는 Aurora MySQL 버전 2에 적용됩니다.
Aurora_fwd_master_dml_stmt_duration	이 라이터 DB 인스턴스로 전달된 DML 명령문의 총 기간입니다. 이 변수는 Aurora MySQL 버전 2에 적용됩니다.
Aurora_fwd_master_errors_rpc_timeout	전달된 연결이 라이터에서 설정되지 못한 횟수입니다.

명칭	설명
Aurora_fwd_master_errors_session_limit	전달된 쿼리 중 라이터의 session full로 인해 거부된 쿼리 수입니다.
Aurora_fwd_master_errors_session_timeout	라이터의 시간 초과로 인해 전달 세션이 종료된 횟수입니다.
Aurora_fwd_master_open_sessions	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. 이 변수는 Aurora MySQL 버전 2에 적용됩니다.
Aurora_fwd_master_select_stmt_count	이 라이터 DB 인스턴스로 전달된 총 SELECT 명령문 수입니다. 이 변수는 Aurora MySQL 버전 2에 적용됩니다.
Aurora_fwd_master_select_stmt_duration	이 라이터 DB 인스턴스로 전달된 SELECT 명령문의 총 기간입니다. 이 변수는 Aurora MySQL 버전 2에 적용됩니다.
Aurora_fwd_writer_dml_stmt_count	이 라이터 DB 인스턴스로 전달된 총 DML 명령문 수입니다. 이 변수는 Aurora MySQL 버전 3에 적용됩니다.
Aurora_fwd_writer_dml_stmt_duration	이 라이터 DB 인스턴스로 전달된 DML 명령문의 총 기간입니다. 이 변수는 Aurora MySQL 버전 3에 적용됩니다.
Aurora_fwd_writer_errors_rpc_timeout	전달된 연결이 라이터에서 설정되지 못한 횟수입니다.
Aurora_fwd_writer_errors_session_limit	전달된 쿼리 중 라이터의 session full로 인해 거부된 쿼리 수입니다.
Aurora_fwd_writer_errors_session_timeout	라이터의 시간 초과로 인해 전달 세션이 종료된 횟수입니다.

명칭	설명
Aurora_fwd_writer_open_sessions	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. 이 변수는 Aurora MySQL 버전 3에 적용됩니다.
Aurora_fwd_writer_select_stmt_count	이 라이터 DB 인스턴스로 전달된 총 SELECT 명령문 수입니다. 이 변수는 Aurora MySQL 버전 3에 적용됩니다.
Aurora_fwd_writer_select_stmt_duration	이 라이터 DB 인스턴스로 전달된 SELECT 명령문의 총 기간입니다. 이 변수는 Aurora MySQL 버전 3에 적용됩니다.
Aurora_lockmgr_buffer_pool_memory_used	Aurora MySQL 잠금 관리자가 사용하는 버퍼 풀 메모리의 양(바이트)입니다.
Aurora_lockmgr_memory_used	Aurora MySQL 잠금 관리자가 사용하는 메모리의 양(바이트)입니다.
Aurora_ml_actual_request_cnt	DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에 전송하는 총 요청 수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
Aurora_ml_actual_response_cnt	DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 응답 횟수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
Aurora_ml_cache_hit_cnt	DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 내부 캐시 히트 횟수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.

명칭	설명
Aurora_ml_logical_request_cnt	마지막 상태 재설정 이후 DB 인스턴스가 Aurora 기계 학습 서비스로 전송되어야 한다고 평가한 논리적 요청의 수입니다. 배치 사용 여부에 따라 이 값은 Aurora_ml_actual_request_cnt 보다 높을 수 있습니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
Aurora_ml_logical_response_cnt	DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 응답 횟수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
Aurora_ml_retry_request_cnt	마지막 상태 재설정 이후 DB 인스턴스가 Aurora 기계 학습 서비스로 전송한 재시도 요청의 수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
Aurora_ml_single_request_cnt	DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 비일괄 모드로 평가되는 Aurora 기계 학습 함수의 총 개수입니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
aurora_oom_avoidance_recovery_state	이 DB 인스턴스에 대해 Aurora 메모리 부족(OOM) 회피 복구가 ACTIVE 상태인지 INACTIVE 상태인지를 나타냅니다.

명칭	설명
aurora_oom_reserved_mem_enter_kb	Aurora의 OOM 처리 메커니즘에서 RESERVED 상태로 진입하는 임계값을 나타냅니다. 서버의 사용 가능한 메모리가 이 임계값 아래로 떨어지면aurora_oom_status 가 RESERVED로 변경되어 서버의 메모리 사용량이 임계 수준에 가까워지고 있음을 나타냅니다.
aurora_oom_reserved_mem_exit_kb	Aurora의 OOM 처리 메커니즘에서 RESERVED 상태를 종료하는 임계값을 나타냅니다. 서버의 사용 가능한 메모리가 이 임계값 이상으로 올라가면 aurora_oom_status 는 NORMAL로 되돌아가 서버가 충분한 메모리 리소스를 갖춘 보다 안정적인 상태로 돌아갔음을 나타냅니다.
aurora_oom_status	이 DB 인스턴스의 현재 OOM 상태를 나타냅니다. 값이 NORMAL인 경우 메모리 리소스가 충분함을 나타냅니다. 값이 RESERVED로 변경되면 서버의 사용 가능한 메모리가 부족함을 나타냅니다. aurora_oom_response 파라미터 구성에 따라 조치가 취해집니다. 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 단원을 참조하십시오.
Aurora_pq_bytes_returned	병렬 쿼리 동안 헤드 노드에 전송된 튜플 데이터 구조를 위한 바이트 수입니다. Aurora_pq_pages_pushed_down 과 비교하기 위해 16,384로 나눕니다.

명칭	설명
`Aurora_pq_max_concurrent_requests`	이 Aurora DB 인스턴스에서 동시에 실행될 수 있는 병렬 쿼리 세션의 최대 수입니다. 이 수는 AWS DB 인스턴스 클래스에 따라 결정되는 고정된 수입니다.
`Aurora_pq_pages_pushed_down`	병렬 쿼리가 헤드 노드로 네트워크 전송을 회피한 데이터 페이지의 수입니다(각각 16 KiB의 고정 크기).
`Aurora_pq_request_attempted`	요청된 병렬 쿼리 세션의 수입니다. 이 값은 하위 쿼리 및 조인과 같은 SQL 구조에 따라, 쿼리당 두 개 이상의 세션을 나타낼 수 있습니다.
`Aurora_pq_request_executed`	병렬 쿼리 세션의 수가 성공적으로 실행됩니다.
`Aurora_pq_request_failed`	클라이언트에 오류를 반환한 병렬 쿼리 세션의 수입니다. 일부 경우에 병렬 쿼리를 위한 요청이 실패할 수도 있습니다(예를 들어, 스토리지 계층의 문제로 인해). 이러한 경우에는 실패한 쿼리 부분이 비병렬 쿼리 메커니즘을 사용하여 다시 시도됩니다. 다시 시도된 쿼리 또한 실패하는 경우, 오류가 클라이언트에 반환되고 이 카운터가 증가합니다.
`Aurora_pq_request_in_progress`	병렬 쿼리 세션의 수가 현재 진행 중입니다. 이 수는 전체 Aurora DB 클러스터가 아니라 연결되어 있는 특정 Aurora DB 인스턴스에 적용됩니다. DB 인스턴스가 동시성 한도에 근접했는지 확인하려면, 이 값을 `Aurora_pq_max_concurrent_requests` 와 비교합니다.
`Aurora_pq_request_not_chosen`	병렬 쿼리가 쿼리를 충족시키기 위해 선택되지 않은 횟수입니다. 이 값은 여러 개의 다른 더 세분화된 카운터의 합계입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.

명칭	설명
Aurora_pq_request_not_chosen_below_min_rows	테이블에 있는 행의 수로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.
Aurora_pq_request_not_chosen_column_bit	예상 열 목록에서 지원되지 않는 데이터 형식으로 인해 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_column_geometry	테이블에 GEOMETRY 데이터 형식의 열이 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 제한 사항이 적용되지 않는 Aurora MySQL 버전에 대한 자세한 정보는 [Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드](#) 섹션을 참조하세요.
Aurora_pq_request_not_chosen_column_lob	테이블에 LOB 데이터 형식의 열이 있거나 VARCHAR 선언된 길이로 인해 외부에 저장된 열이 있으므로에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 제한 사항이 제거된 Aurora MySQL 버전에 대한 자세한 정보는 [Aurora MySQL 버전 3에 병렬 쿼리 클러스터 업그레이드](#) 섹션을 참조하세요.
Aurora_pq_request_not_chosen_column_virtual	테이블에 가상 열이 포함되어 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_custom_charset	테이블에 사용자 지정 문자 집합이 있는 열이 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_fast_ddl	테이블이 현재 빠른 DDL ALTER 문에 의해 변경되고 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.

명칭	설명
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	테이블 데이터 중 95% 미만이 버퍼 풀에 있는 경우에도, 버퍼링되지 않은 테이블 데이터가 병렬 쿼리가 가치 있을 만큼 충분하지 않았기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다.
Aurora_pq_request_not_chosen_full_text_index	테이블에 전체 텍스트 인덱스가 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_high_buffer_pool_pct	테이블 데이터 중 많은 양이(현재, 95% 이상) 이미 버퍼 풀에 있었기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다. 이러한 경우, 옵티마이저는 버퍼 풀에서 데이터 읽기가 더 효율적이라고 결정합니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.
Aurora_pq_request_not_chosen_index_hint	쿼리에 인덱스 힌트가 포함되어 있기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_innodb_table_format	테이블이 지원되지 않는 InnoDB 행 형식을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. Aurora 병렬 쿼리는 COMPACT, REDUNDANT 및 DYNAMIC 행 형식에만 적용됩니다.
Aurora_pq_request_not_chosen_long_trx	장기 실행 트랜잭션 내부에서 쿼리가 시작되는 중이어서 비병렬 쿼리 처리 경로를 사용한 병렬 쿼리 요청의 수입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.
Aurora_pq_request_not_chosen_no_where_clause	쿼리에 WHERE 절이 포함되어 있지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.

명칭	설명
Aurora_pq_request_not_chosen_range_scan	쿼리가 인덱스에서 범위 스캔을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_row_length_too_long	모든 열의 총 결합 길이가 너무 길기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_small_table	행의 수 및 평균 행 길이에 따라 결정된 테이블의 전체 크기로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 쿼리가 실제로 수행되지 않더라도 EXPLAIN 문을 통해 이 카운터가 증가할 수 있습니다.
Aurora_pq_request_not_chosen_temporary_table	쿼리가 지원되지 않는 MyISAM 또는 memory 테이블 형식을 사용하는 임시 테이블을 참조하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_tx_isolation	쿼리가 지원되지 않는 트랜잭션 격리 수준을 사용하기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 읽기 전용 DB 인스턴스에서는 병렬 쿼리가 REPEATABLE READ 및 READ COMMITTED 격리 수준에만 적용됩니다.
Aurora_pq_request_not_chosen_update_delete_stmts	쿼리가 UPDATE 또는 DELETE 문의 일부이기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다.
Aurora_pq_request_not_chosen_unsupported_access	WHERE 절이 병렬 쿼리를 위한 기준에 부합되지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 결과는 쿼리에 데이터 집약적인 스캔이 필요하지 않은 경우이거나 쿼리가 DELETE 또는 UPDATE 문인 경우에 발생할 수 있습니다.

명칭	설명
Aurora_pq_request_not_chosen_unsupported_storage_type	Aurora MySQL DB 클러스터가 지원되는 Aurora 클러스터 스토리지 구성을 사용하지 않기 때문에 비병렬 쿼리 처리 경로를 사용하는 병렬 쿼리 요청 수입니다. 자세한 내용은 제한 사항 단원을 참조하십시오. 이 파라미터는 Aurora MySQL 버전 3.04 이상에 적용됩니다.
Aurora_pq_request_throttled	동시 병렬 쿼리의 최대 수가 이미 특정 Aurora DB 인스턴스에서 실행 중이기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다.
Aurora_repl_bytes_received	마지막 재시작 이후 Aurora MySQL 리더 데이터베이스 인스턴스에 복제된 바이트 수입니다. 자세한 내용은 Amazon Aurora MySQL을 사용한 복제 단원을 참조하십시오.
Aurora_reserved_mem_exceeded_incidents	마지막 재시작 이후 엔진이 예약된 메모리 제한을 초과한 횟수입니다. aurora_oom_response 가 구성된 경우 이 임계값은 메모리 부족(OOM) 방지 활동이 트리거되는 시기를 정의합니다. Aurora MySQL OOM 응답에 대한 자세한 내용은 Aurora MySQL 데이터베이스의 메모리 부족 문제 해결 섹션을 참조하세요.
Aurora_thread_pool_thread_count	Aurora 스레드 풀의 현재 스레드 수입니다. Aurora MySQL의 스레드 풀에 대한 자세한 내용은 스레드 풀 섹션을 참조하세요.

명칭	설명
Aurora_tmz_version	DB 클러스터에서 사용하는 시간대 정보의 현재 버전을 나타냅니다. 값은 인터넷 할당 번호 관리 기관(IANA) 형식인 YYYYsuffix 형식을 따릅니다(예: 2022a 및 2023c). 이 파라미터는 Aurora MySQL 버전 2.12 이상 및 버전 3.04 이상에 적용됩니다.
Aurora_zdr_oom_threshold	잠재적인 메모리 관련 문제에서 복구하기 위해 가동 중지 시간 제로 재시작(ZDR)을 시작하기 위한 Aurora DB 인스턴스의 메모리 임계값을 킬로바이트(KB)로 나타냅니다.
server_aurora_das_running	이 DB 인스턴스에서 데이터베이스 활동 스트림(DAS)을 활성화했는지 아니면 비활성했는지를 나타냅니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 단원을 참조하십시오.

Aurora MySQL에 적용되지 않는 MySQL 상태 변수

Aurora MySQL과 MySQL 간의 아키텍처 차이 때문에 일부 MySQL 상태 변수는 Aurora MySQL에 적용되지 않습니다.

다음 MySQL 상태 변수는 Aurora MySQL에 적용되지 않습니다. 단, 이 목록이 전부는 아닙니다.

- innodb_buffer_pool_bytes_dirty
- innodb_buffer_pool_pages_dirty
- innodb_buffer_pool_pages_flushed

Aurora MySQL 버전 3은 Aurora MySQL 버전 2에 있던 다음 상태 변수를 제거합니다.

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used

- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats
- Slave_retried_transactions
- Slave_running
- Time_since_zero_connections

이러한 MySQL 상태 변수는 Aurora MySQL 버전 2에서 사용할 수 있지만 Aurora MySQL 버전 3에서는 사용할 수 없습니다.

- Innodb_redo_log_enabled

- `Innodb_undo_tablespaces_total`
- `Innodb_undo_tablespaces_implicit`
- `Innodb_undo_tablespaces_explicit`
- `Innodb_undo_tablespaces_active`

Aurora MySQL 대기 이벤트

다음은 Aurora MySQL의 일반 대기 이벤트입니다.

> **Note**
>
> 대기 이벤트를 사용하여 Aurora MySQL 성능을 튜닝하는 방법에 대한 자세한 내용은 대기 이벤트로 Aurora MySQL 튜닝을 참조하세요.
> MySQL 대기 이벤트에 사용되는 이름 지정 규칙에 대한 자세한 내용은 MySQL 설명서의 성능 스키마 장비 이름 지정 규칙을 참조하세요.

cpu

실행할 준비가 된 활성 연결 수는 vCPU 수보다 일관되게 많습니다. 자세한 내용은 cpu 단원을 참조하십시오.

io/aurora_redo_log_flush

세션은 Aurora 스토리지에 데이터를 유지하고 있습니다. 일반적으로 이 대기 이벤트는 Aurora MySQL의 쓰기 I/O 작업을 위한 것입니다. 자세한 내용은 io/aurora_redo_log_flush 단원을 참조하십시오.

io/aurora_respond_to_client

쿼리 처리가 완료되었으며 2.10.2 이상 2.10 버전, 2.09.3 이상 2.09 버전, 2.07.7 이상 2.07 버전과 같은 Aurora MySQL 버전에 대한 결과가 애플리케이션 클라이언트에 반환됩니다. DB 인스턴스 클래스의 네트워크 대역폭을 반환되는 결과 집합의 크기와 비교합니다. 또한 클라이언트 측 응답 시간도 확인합니다. 클라이언트가 응답하지 않고 TCP 패킷을 처리할 수 없는 경우 패킷이 삭제되고 TCP가 재전송될 수 있습니다. 이러한 상황은 네트워크 대역폭에 부정적인 영향을 미칩니다. 2.10.2, 2.09.3 및 2.07.7 이전 버전에서는 이 대기 이벤트에 유휴 시간이 잘못 포함됩니다. 이 대기 시간이 눈에 띄게 표시되는 경우 데이터베이스를 튜닝하는 법을 알아보려면 io/aurora_respond_to_client 섹션을 참조하세요.

io/file/csv/data

스레드는 쉼표로 구분된 값(CSV) 형식으로 테이블에 작성합니다. CSV 테이블 사용을 확인하세요. 일반적으로 이 이벤트는 테이블에서 `log_output`을 설정하는 중에 발생합니다.

io/file/sql/binlog

스레드는 디스크에 기록 중인 이진 로그(binlog) 파일에서 대기 중입니다.

io/redo_log_flush

세션은 Aurora 스토리지에 데이터를 유지하고 있습니다. 일반적으로 이 대기 이벤트는 Aurora MySQL의 쓰기 I/O 작업을 위한 것입니다. 자세한 내용은 io/redo_log_flush 단원을 참조하십시오.

io/socket/sql/client_connection

`mysqld` 프로그램은 들어오는 새 클라이언트 연결을 처리하기 위해 스레드를 만드는 중입니다. 자세한 내용은 io/socket/sql/client_connection 단원을 참조하십시오.

io/table/sql/handler

엔진이 테이블에 대한 액세스를 기다리고 있습니다. 이 이벤트는 데이터가 버퍼 풀에 캐시되는지 디스크에 액세스되는지 관계없이 발생합니다. 자세한 내용은 io/table/sql/handler 단원을 참조하십시오.

lock/table/sql/handler

이 대기 이벤트는 테이블 잠금 대기 이벤트 핸들러입니다. 성능 스키마의 atom 및 molecule 이벤트에 대한 자세한 내용은 MySQL 설명서의 성능 스키마 Atom 및 Molecule 이벤트를 참조하세요.

synch/cond/innodb/row_lock_wait

여러 데이터 조작 언어(DML) 문이 동시에 동일한 데이터베이스 행에 액세스하고 있습니다. 자세한 내용은 synch/cond/innodb/row_lock_wait 단원을 참조하십시오.

synch/cond/innodb/row_lock_wait_cond

여러 DML 문이 동시에 동일한 데이터베이스 행에 액세스하고 있습니다. 자세한 내용은 synch/cond/innodb/row_lock_wait_cond 단원을 참조하십시오.

synch/cond/sql/MDL_context::COND_wait_status

스레드가 테이블 메타데이터 잠금을 기다리는 중입니다. 엔진이 이러한 유형의 잠금을 사용하여 데이터베이스 스키마에 대한 동시 액세스를 관리하고 데이터 일관성을 보장합니다. 자세한 내용은 MySQL 설명서의 잠금 작업 최적화를 참조하세요. 이 이벤트가 눈에 띄게 표시되는 경우 데이터베이스를 튜닝하는 법을 알아보려면 synch/cond/sql/MDL_context::COND_wait_status 섹션을 참조하세요.

synch/cond/sql/MYSQL_BIN_LOG::COND_done

이진 로깅을 설정했습니다. 높은 커밋 처리량, 많은 커밋 트랜잭션 또는 binlog를 읽는 복제본이 있을 수 있습니다. 다중 행 문을 사용하거나 명령문을 하나의 트랜잭션에 번들링하는 것이 좋습니다. Aurora에서는 이진 로그 복제 대신 글로벌 데이터베이스를 사용하거나 `aurora_binlog_*` 파라미터를 사용합니다.

synch/mutex/innodb/aurora_lock_thread_slot_futex

여러 DML 문이 동시에 동일한 데이터베이스 행에 액세스하고 있습니다. 자세한 내용은 synch/mutex/innodb/aurora_lock_thread_slot_futex 단원을 참조하십시오.

synch/mutex/innodb/buf_pool_mutex

버퍼 풀은 작업 중인 데이터 세트를 보유하기에 충분하지 않습니다. 또는 워크로드가 특정 테이블의 페이지에 액세스하여 버퍼 풀에서 경합이 발생합니다. 자세한 내용은 synch/mutex/innodb/buf_pool_mutex 단원을 참조하십시오.

synch/mutex/innodb/fil_system_mutex

프로세스가 테이블스페이스 메모리 캐시에 대한 액세스를 기다리고 있습니다. 자세한 내용은 synch/mutex/innodb/fil_system_mutex 단원을 참조하십시오.

synch/mutex/innodb/trx_sys_mutex

작업은 일관되거나 제어된 방식으로 InnoDB의 트랜잭션 ID를 확인, 업데이트, 삭제 또는 추가하는 것입니다. 이러한 작업에는 성능 스키마 계측을 통해 추적되는 `trx_sys` 뮤텍스 호출이 필요합니다. 작업에는 데이터베이스가 시작하거나 종료할 때 트랜잭션 시스템 관리, 롤백, 실행 취소 정리, 행 읽기 액세스 및 버퍼 풀 로드 등이 포함됩니다. 많은 수의 트랜잭션을 통한 데이터베이스 로드가 많으면 이 대기 이벤트가 자주 표시됩니다. 자세한 내용은 synch/mutex/innodb/trx_sys_mutex 단원을 참조하십시오.

synch/mutex/mysys/KEY_CACHE::cache_lock

`keycache->cache_lock` 뮤텍스는 MyISAM 테이블의 키 캐시에 대한 액세스를 제어합니다. Aurora MySQL에서는 영구 데이터를 저장하는 데는 MyISAM 테이블을 사용할 수 없지만 내부 임시 테이블을 저장하는 데는 사용됩니다. 특정 상황에서는 임시 테이블이 더 이상 메모리에 들어가지 않을 때 디스크에 기록되므로 `created_tmp_tables` 또는 `created_tmp_disk_tables` 상태 카운터를 확인하는 것이 좋습니다.

synch/mutex/sql/FILE_AS_TABLE::LOCK_offsets

엔진은 테이블 메타데이터 파일을 열거나 생성할 때 이 뮤텍스를 가져옵니다. 이 대기 이벤트가 과도한 빈도로 발생하면 생성되거나 열려 있는 테이블 수가 급증합니다.

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

엔진은 열린 테이블을 추적하는 내부 구조에서 `reset_size`, `detach_contents` 또는 `add_contents`와 같은 작업을 수행하는 동안 이 뮤텍스를 가져옵니다. 뮤텍스는 목록 내용에 대한 액세스를 동기화합니다. 이 대기 이벤트가 높은 빈도로 발생하면 이전에 액세스한 테이블 집합이 갑자기 변경되었음을 나타냅니다. 엔진은 새 테이블에 액세스하거나 이전에 액세스한 테이블과 관련된 컨텍스트를 제거해야 합니다.

synch/mutex/sql/LOCK_open

세션에서 열리는 테이블 수가 테이블 정의 캐시 또는 테이블 열기 캐시의 크기를 초과합니다. 이러한 캐시의 크기를 늘립니다. 자세한 내용은 MySQL에서 테이블을 열고 닫는 방법을 참조하세요.

synch/mutex/sql/LOCK_table_cache

세션에서 열리는 테이블 수가 테이블 정의 캐시 또는 테이블 열기 캐시의 크기를 초과합니다. 이러한 캐시의 크기를 늘립니다. 자세한 내용은 MySQL에서 테이블을 열고 닫는 방법을 참조하세요.

synch/mutex/sql/LOG

이 대기 이벤트에서는 로그 잠금을 대기 중인 스레드가 있습니다. 예를 들어, 스레드가 느린 쿼리 로그에 쓰기 위해 잠금을 대기 중일 수 있습니다.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

이 대기 이벤트에서는 바이너리 로그에 커밋할 목적으로 잠금 획득을 대기 중인 스레드가 있습니다. 변경률이 매우 높은 데이터베이스에서 바이너리 로깅 경합이 발생할 수 있습니다. MySQL의 버전에 따라 바이너리 로그의 일관성과 내구성을 보호하기 위해 사용 중인 특정 잠금이 있습니다. RDS for MySQL에서는 복제 및 자동 백업 과정에 바이너리 로그가 사용됩니다. Aurora MySQL에서는 기본 복제 또는 백업에 바이너리 로그가 사용되지 않습니다. 바이너리 로그는 기본적으로 비활성화되지만, 활성화한 후 외부 복제 또는 변경 데이터 캡처에 사용할 수 있습니다. 자세한 내용은 MySQL 설명서의 바이너리 로그를 참조하십시오.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_dump_thread_metrics_collection

이진 로깅이 설정되어 있으면 엔진은 활성 덤프 스레드 지표를 엔진 오류 로그와 내부 작업 맵에 인쇄할 때 이 뮤텍스를 가져옵니다.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_inactive_binlogs_map

이진 로깅이 설정되어 있으면 엔진이 최신 파일 뒤에 있는 binlog 파일 목록을 통해 추가, 삭제 또는 검색할 때 이 뮤텍스를 가져옵니다.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_io_cache

이진 로깅이 설정되어 있으면 Aurora binlog IO 캐시 작업(캐시 정보 할당, 크기 조정, 해제, 쓰기, 읽기, 제거 및 액세스) 중에 엔진이 이 뮤텍스를 획득합니다. 이 이벤트가 자주 발생하면 엔진은 binlog 이벤트가 저장되는 캐시에 액세스합니다. 대기 시간을 줄이려면 커밋을 줄이세요. 여러 명령문을 단일 트랜잭션으로 그룹화해 보세요.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

이진 로깅을 설정했습니다. 높은 커밋 처리량, 많은 커밋 트랜잭션 또는 binlog를 읽는 복제본이 있을 수 있습니다. 다중 행 문을 사용하거나 명령문을 하나의 트랜잭션에 번들링하는 것이 좋습니다. Aurora에서는 이진 로그 복제 대신 글로벌 데이터베이스를 사용하거나 `aurora_binlog_*` 파라미터를 사용합니다.

synch/mutex/sql/SERVER_THREAD::LOCK_sync

파일 쓰기를 위해 스레드를 스케줄링, 처리 또는 시작하는 동안 `SERVER_THREAD::LOCK_sync` 뮤텍스를 가져옵니다. 이 대기 이벤트가 과도하게 발생하면 데이터베이스에서 쓰기 작업이 증가했다는 것을 나타냅니다.

synch/mutex/sql/TABLESPACES:lock

엔진이 다음과 같은 테이블스페이스 작업(생성, 삭제, 자르기 및 확장) 중에 `TABLESPACES:lock` 뮤텍스를 가져옵니다. 이 대기 이벤트가 과도하게 발생하면 테이블스페이스 작업의 빈도가 높다는 것을 나타냅니다. 예를 들어 대량의 데이터를 데이터베이스에 로드하는 것입니다.

synch/rwlock/innodb/dict

이 대기 이벤트에서는 InnoDB 데이터 딕셔너리에 보관된 rwlock을 대기 중인 스레드가 있습니다.

synch/rwlock/innodb/dict_operation_lock

이 대기 이벤트에서는 InnoDB 데이터 딕셔너리 작업에 대한 잠금을 보유한 스레드가 있습니다.

synch/rwlock/innodb/dict sys RW lock

데이터 정의 언어(DDL) 코드에서 많은 수의 동시 데이터 제어 언어(DCL) 문이 동시에 트리거됩니다. 정기적인 애플리케이션 작업 중에 DDL에 대한 애플리케이션의 종속성을 줄입니다.

synch/rwlock/innodb/index_tree_rw_lock

많은 수의 비슷한 데이터 조작 언어(DML) 문이 동시에 동일한 데이터베이스 객체에 액세스하고 있습니다. 다중 행 문을 사용해 보세요. 또한 서로 다른 데이터베이스 객체에 워크로드를 분산하세요. 예를 들어 파티셔닝을 구현합니다.

synch/sxlock/innodb/dict_operation_lock

데이터 정의 언어(DDL) 코드에서 많은 수의 동시 데이터 제어 언어(DCL) 문이 동시에 트리거됩니다. 정기적인 애플리케이션 작업 중에 DDL에 대한 애플리케이션의 종속성을 줄입니다.

synch/sxlock/innodb/dict_sys_lock

데이터 정의 언어(DDL) 코드에서 많은 수의 동시 데이터 제어 언어(DCL) 문이 동시에 트리거됩니다. 정기적인 애플리케이션 작업 중에 DDL에 대한 애플리케이션의 종속성을 줄입니다.

synch/sxlock/innodb/hash_table_locks

세션이 버퍼 풀에서 페이지를 찾을 수 없습니다. 엔진은 파일을 읽거나 버퍼 풀의 LRU(least-recently used) 목록을 수정해야 합니다. 버퍼 캐시 크기를 늘리고 관련 쿼리에 대한 액세스 경로를 개선하는 것이 좋습니다.

synch/sxlock/innodb/index_tree_rw_lock

많은 수의 비슷한 데이터 조작 언어(DML) 문이 동시에 동일한 데이터베이스 객체에 액세스하고 있습니다. 다중 행 문을 사용해 보세요. 또한 서로 다른 데이터베이스 객체에 워크로드를 분산하세요. 예를 들어 파티셔닝을 구현합니다.

동기화 대기 이벤트 문제 해결에 대한 자세한 내용은 [MySQL DB 인스턴스가 Performance Insights에서 SYNCH 대기 이벤트를 기다리는 많은 활성 세션 수를 표시하는 이유는 무엇인가요?]를 참조하세요.

Aurora MySQL 스레드 상태

다음은 Aurora MySQL의 일반 스레드 상태입니다.

권한 확인

스레드는 서버에 명령문을 실행하는 데 필요한 권한이 있는지 확인합니다.

쿼리에 대한 쿼리 캐시 확인

서버는 현재 쿼리가 쿼리 캐시에 있는지 확인합니다.

정리

이 상태는 작업이 완료되었지만 클라이언트에 의해 닫히지 않은 연결의 최종 상태입니다. 가장 좋은 해결책은 코드에서 연결을 명시적으로 닫는 것입니다. 또는 파라미터 그룹에서 `wait_timeout`에 대해 더 낮은 값을 설정할 수 있습니다.

테이블 닫기

스레드가 변경된 테이블 데이터를 디스크로 플러시하고 사용된 테이블을 닫습니다. 빠른 작업이 아닌 경우 인스턴스 클래스 네트워크 대역폭에 대해 네트워크 대역폭 사용 지표를 확인합니다. 또한 `table_open_cache` 및 `table_definition_cache`의 파라미터 값이 엔진이 테이블을 자주 열고 닫을 필요가 없도록 충분한 테이블을 동시에 열 수 있는지 확인합니다. 이러한 파라미터는 인스턴스의 메모리 사용에 영향을 줍니다.

HEAP을 MyISAM으로 변환

쿼리는 임시 테이블을 인메모리에서 온디스크로 변환하고 있습니다. 쿼리 처리의 중간 단계에서 MySQL에서 만든 임시 테이블이 메모리에 비해 너무 커지기 때문에 이러한 변환이 필요합니다. `tmp_table_size` 및 `max_heap_table_size`의 값을 확인합니다. 이후 버전에서는 이 스레드 상태 이름은 `converting HEAP to ondisk`입니다.

HEAP을 온디스크로 변환

스레드는 내부 임시 테이블을 인메모리 테이블에서 온디스크 테이블로 변환합니다.

임시 테이블에 복사

스레드가 `ALTER TABLE` 문을 처리 중입니다. 이 상태는 새 구조의 테이블이 작성된 후 행이 이 테이블로 복사되기 전에 발생합니다. 이 상태의 스레드의 경우 성능 스키마를 사용하여 복사 작업의 진행률에 대한 정보를 얻을 수 있습니다.

정렬 인덱스 생성

Aurora MySQL은 기존 인덱스를 사용하여 쿼리의 `ORDER BY` 또는 `GROUP BY` 절을 충족시킬 수 없기 때문에 정렬을 수행합니다. 자세한 내용은 [정렬 인덱스 생성](#) 단원을 참조하십시오.

테이블 생성

스레드가 영구 또는 임시 테이블을 생성 중입니다.

지연된 커밋 확인 완료

Aurora MySQL의 비동기 커밋이 승인을 받아 완료되었습니다.

지연된 커밋 확인 시작

Aurora MySQL 스레드는 비동기 커밋 프로세스를 시작했지만 승인을 기다리고 있습니다. 이 시간은 일반적으로 트랜잭션의 진정한 커밋 시간입니다.

지연된 전송 확인 완료

응답이 클라이언트에 전송되는 동안 연결된 Aurora MySQL 작업자 스레드를 해제할 수 있습니다. 스레드는 다른 작업을 시작할 수 있습니다. `delayed send ok` 상태는 클라이언트에 대한 비동기 승인이 완료되었음을 의미합니다.

지연된 전송 확인 시작

Aurora MySQL 작업자 스레드가 클라이언트에 비동기적으로 응답을 보냈으므로 이제 다른 연결에 대해 자유롭게 작업할 수 있습니다. 트랜잭션이 아직 승인되지 않은 비동기 커밋 프로세스를 시작했습니다.

executing

스레드가 명령문을 실행하기 시작했습니다.

freeing items

스레드가 명령을 실행했습니다. 이 상태에서 수행되는 항목의 일부 해제에는 쿼리 캐시가 포함됩니다. 이 상태는 일반적으로 정리됩니다.

init

이 상태는 `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT` 또는 `UPDATE` 문의 초기화 전에 발생합니다. 이 상태의 작업에는 이진 로그 또는 InnoDB 로그의 플러시 및 쿼리 캐시의 일부 정리가 포함됩니다.

소스가 모든 binlog를 복제본으로 전송했습니다. 더 많은 업데이트를 기다리고 있습니다.

주 노드가 복제의 일부를 완료했습니다. 스레드는 이진 로그(binlog)에 쓸 수 있도록 더 많은 쿼리가 실행되기를 기다리고 있습니다.

테이블 열기

스레드가 테이블을 열려고 합니다. 이 작업은 `ALTER TABLE` 또는 `LOCK TABLE` 문이 `table_open_cache` 값을 완료해야 하거나 초과하지 않는 한 빠르게 수행됩니다.

최적화

서버가 쿼리에 대해 초기 최적화를 수행 중입니다.

준비 중

이 상태는 쿼리 최적화 중에 발생합니다.

쿼리 종료

이 상태는 쿼리 처리 후 항목 해제 상태 전에 발생합니다.

중복 제거

Aurora MySQL은 쿼리의 초기 단계에서 `DISTINCT` 작업을 최적화 할 수 없습니다. Aurora MySQL은 결과를 클라이언트에 보내기 전에 중복된 행을 모두 제거해야 합니다.

업데이트를 위해 행 검색

스레드는 업데이트하기 전에 일치하는 모든 행을 찾습니다. 이 단계는 `UPDATE`가 엔진이 행을 찾는 데 사용하는 인덱스를 변경하는 경우 필요합니다.

binlog 이벤트를 슬레이브로 보내기

스레드는 이진 로그에서 이벤트를 읽은 다음 복제본으로 보냅니다.

캐시된 결과를 클라이언트로 보내기

서버가 쿼리 캐시에서 쿼리 결과를 가져와 클라이언트로 보냅니다.

데이터 전송

스레드가 `SELECT` 문에 대한 행을 읽고 처리하고 있지만 아직 클라이언트로 데이터 전송을 시작하지 않았습니다. 이 프로세스는 쿼리를 충족하는 데 필요한 결과가 포함된 페이지를 식별하는 것입니다. 자세한 내용은 데이터 전송 단원을 참조하십시오.

클라이언트로 전송

서버가 클라이언트에 패킷을 쓰고 있습니다. 이전 MySQL 버전에서 이 대기 이벤트에 `writing to net`의 레이블이 지정되었습니다.

starting

이 단계는 명령문 실행 시작의 첫 번째 단계입니다.

statistics

서버는 쿼리 실행 계획을 개발하기 위해 통계를 계산하고 있습니다. 스레드가 오랫동안 이 상태인 경우 서버가 다른 작업을 수행하는 동안 디스크 바인딩될 수 있습니다.

쿼리 캐시에 결과 저장

서버가 쿼리 캐시에 쿼리 결과를 저장하고 있습니다.

system lock

스레드가 `mysql_lock_tables`를 호출했지만 호출 이후 스레드 상태가 업데이트되지 않았습니다. 이 일반적인 상태는 여러 가지 이유로 발생합니다.

update

스레드가 테이블 업데이트를 시작하기 위해 준비 중입니다.

updating

스레드가 행을 검색하고 업데이트하고 있습니다.

user lock

스레드가 GET_LOCK 호출을 실행했습니다. 스레드는 자문 잠금을 요청하고 기다리고 있거나 자문 잠금을 요청할 계획입니다.

추가 업데이트 대기 중

주 노드가 복제의 일부를 완료했습니다. 스레드는 이진 로그(binlog)에 쓸 수 있도록 더 많은 쿼리가 실행되기를 기다리고 있습니다.

스키마 메타데이터 잠금 대기 중

메타데이터 잠금을 기다리는 것입니다.

저장된 함수 메타데이터 잠금 대기 중

메타데이터 잠금을 기다리는 것입니다.

저장된 프로시저 메타데이터 잠금 대기 중

메타데이터 잠금을 기다리는 것입니다.

테이블 플러시 대기 중

스레드는 FLUSH TABLES를 실행 중이며 모든 스레드에서 해당 테이블을 닫을 때까지 대기하고 있습니다. 또는 스레드는 테이블의 기본 구조가 변경되었다는 알림을 받았으므로 테이블을 다시 열어 새 구조를 가져와야 합니다. 테이블을 다시 열려면 스레드는 다른 모든 스레드에서 테이블을 닫을 때까지 기다려야 합니다. 이 알림은 다른 스레드가 테이블에서 FLUSH TABLES, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE 또는 OPTIMIZE TABLE 문 중 하나를 사용한 경우 발생합니다.

테이블 수준 잠금 대기 중

한 세션은 테이블에서 잠금을 유지하는 반면 다른 세션은 동일한 테이블에서 동일한 잠금을 가져오려고 시도합니다.

테이블 메타데이터 잠금 대기 중

Aurora MySQL은 메타데이터 잠금을 사용하여 데이터베이스 객체에 대한 동시 액세스를 관리하고 데이터 일관성을 보장합니다. 이 대기 이벤트에서 한 세션은 테이블에서 메타데이터 잠금을 유지하는 반면 다른 세션은 동일한 테이블에서 동일한 잠금을 가져오려고 시도합니다. 성능 스키마가 사용 설정되면 이 스레드 상태는 synch/cond/sql/MDL_context::COND_wait_status 대기 이벤트로 보고됩니다.

net에 쓰기

서버가 네트워크에 패킷을 쓰고 있습니다. MySQL 이후 버전에서 이 대기 이벤트에 `Sending to client`의 레이블이 지정됩니다.

Aurora MySQL 격리 수준

Aurora MySQL 클러스터의 DB 인스턴스가 격리의 데이터베이스 속성을 구현하는 방식을 알아봅니다. 이 주제에서는 Aurora MySQL 기본 동작이 엄격한 일관성과 높은 성능 사이에서 어떻게 균형을 이루는지 설명합니다. 이 정보를 사용하여 워크로드의 특성에 따라 언제 기본 설정을 변경할지 결정할 수 있습니다.

라이터 인스턴스에 사용 가능한 격리 수준

Aurora MySQL DB 클러스터의 기본 인스턴스에서 격리 수준 `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED` 및 `SERIALIZABLE`을 사용할 수 있습니다. 이러한 격리 수준은 RDS for MySQL과 마찬가지로 Aurora MySQL에서 동일하게 작동합니다.

리더 인스턴스의 REPEATABLE READ 격리 수준

기본적으로, 읽기 전용 Aurora 복제본으로 구성된 Aurora MySQL DB 인스턴스에서는 항상 `REPEATABLE READ` 격리 수준을 사용합니다. 이 DB 인스턴스에서는 `SET TRANSACTION ISOLATION LEVEL` 문은 모두 무시하고 계속해서 `REPEATABLE READ` 격리 수준을 사용합니다.

리더 인스턴스의 READ COMMITTED 격리 수준

애플리케이션에 기본 인스턴스의 쓰기 집약적인 워크로드와 Aurora 복제본의 장기 실행 쿼리가 포함된 경우 상당한 제거 지연 시간이 발생할 수 있습니다. 제거 지연 시간은 내부 가비지 수집이 장기 실행 쿼리로 차단되는 경우 발생합니다. 사용자가 겪는 증상은 `history list length` 명령의 출력에서 `SHOW ENGINE INNODB STATUS`의 값이 커지는 것입니다. CloudWatch의 `RollbackSegmentHistoryListLength` 지표를 사용하여 이 값을 모니터링할 수 있습니다. 상당한 제거 지연은 보조 인덱스의 효율성과 전반적인 쿼리 성능을 저하하며 스토리지 공간 낭비로 이어질 수 있습니다.

이러한 문제가 발생하면 Aurora MySQL 세션 수준 구성 설정인 `aurora_read_replica_read_committed`를 Aurora 복제본에서 `READ COMMITTED` 격리 수준을 사용하도록 설정할 수 있습니다. 이 설정을 적용하면 장기 실행 쿼리를 테이블을 수정하는 트랜잭션과 동시에 수행하여 발생할 수 있는 속도 저하 및 공간 낭비를 줄이는 데 도움이 됩니다.

이 설정을 사용하기 전에 READ COMMITTED 격리의 특정 Aurora MySQL 동작을 이해하는 것이 좋습니다. Aurora 복제본 READ COMMITTED 동작은 ANSI SQL 표준을 준수합니다. 그러나 격리는 사용자에게 익숙한 일반적인 MySQL READ COMMITTED 동작보다 덜 엄격합니다. 따라서 Aurora MySQL 기본 인스턴스 또는 RDS for MySQL의 READ COMMITTED에서 실행한 동일 쿼리에 대한 결과와는 다른 쿼리 결과가 Aurora MySQL 읽기 전용 복제본의 READ COMMITTED에서 나올 수 있습니다. 매우 큰 데이터베이스를 스캔하는 종합 보고서와 같은 사례에서는 aurora_read_replica_read_committed 설정을 사용하는 것을 고려해 볼 수 있습니다. 반면에, 정밀도 및 반복성이 중요하고 결과 집합이 작은 짧은 쿼리에는 이 설정을 사용하지 않는 것이 좋습니다.

쓰기 전달 기능을 사용하는 Aurora Global Database의 보조 클러스터 내에 있는 세션에는 READ COMMITTED 격리 수준을 사용할 수 없습니다. 쓰기 전달에 대한 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용](#) 단원을 참조하십시오.

리더에 대해 READ COMMITTED 사용

Aurora 복제본에 대해 READ COMMITTED 격리 수준을 사용하려면 aurora_read_replica_read_committed 구성 설정을 ON으로 설정합니다. 특정 Aurora 복제본에 연결된 상태에서 이 설정을 세션 수준에서 사용합니다. 이렇게 하려면 다음 SQL 명령을 실행합니다.

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

이 구성 설정을 일시적으로 사용하여 대화형 일회성 쿼리를 수행할 수 있습니다. 또한 다른 애플리케이션에 대한 기본 설정을 변경하지 않은 상태에서 READ COMMITTED 격리 수준에서 이점을 얻는 보고 또는 데이터 분석 애플리케이션을 실행할 수 있습니다.

aurora_read_replica_read_committed 설정이 활성화된 경우 SET TRANSACTION ISOLATION LEVEL 명령을 사용하여 적절한 트랜잭션에 격리 수준을 지정하세요.

```
set transaction isolation level read committed;
```

Aurora 복제본에서 READ COMMITTED 동작의 차이

aurora_read_replica_read_committed 설정을 통해 Aurora 복제본에 대해 READ COMMITTED 격리 수준을 사용할 수 있게 할 수 있습니다. 이때 일관성 동작은 장기 실행 트랜잭션에 최적화됩니다. Aurora 복제본의 READ COMMITTED 격리 수준은 Aurora 기본 인스턴스보다 격리가 덜 엄격합니다. 이러한 이유로 쿼리에서 일관성 없는 특정 유형의 결과가 반환될 가능성을 수용할 수 있음을 사용자가 알고 있는 Aurora 복제본에서만 이 설정을 활성화해야 합니다.

aurora_read_replica_read_committed 설정이 켜져 있으면 쿼리 시 특정한 종류의 읽기 이상이 발생할 수 있습니다. 두 가지 종류의 이상이 애플리케이션 코드를 이해하고 애플리케이션 코드에서 처리하는 데 특히 중요합니다. 반복 불가능한 읽기는 쿼리가 실행 중일 때 다른 트랜잭션이 커밋되면 발생합니다. 장기 실행 쿼리에는 종료 시 반환되는 것과 다른 데이터가 쿼리 시작 시점에 반환될 수 있습니다. 가상 읽기는 쿼리가 실행 중일 때 다른 트랜잭션으로 인해 기존 행이 재편되고 쿼리에서 하나 이상의 행을 두 번 읽으면 발생합니다.

가상 읽기의 결과로 쿼리에서 행 수의 일관성이 결여될 수 있습니다. 또한 반복 불가능한 읽기로 인해 쿼리에서 불완전하거나 일관성 없는 결과를 반환할 수 있습니다. 예를 들어 INSERT 또는 DELETE와 같은 SQL 문에서 동시에 수정하는 테이블을 조인 연산자가 참조한다고 가정합시다. 이 경우 조인 쿼리에서는 테이블 하나에서 행 하나를 읽지만 다른 테이블에서 이에 상응하는 행은 읽지 않을 수 있습니다.

ANSI SQL 표준에서는 READ COMMITTED 격리 수준에 대해 이 두 가지 동작을 모두 허용합니다. 그러나 그러한 동작은 READ COMMITTED의 일반적인 MySQL 구현과 다릅니다. 따라서 aurora_read_replica_read_committed 설정을 활성화하기 전에 모든 기존 SQL 코드를 확인하여 이 코드가 더 느슨한 일관성 모델에서 예상대로 작동하는지 확인하십시오.

이 설정이 활성화되어 있는 상태에서 행 수 및 기타 결과는 READ COMMITTED 격리 수준에서 강한 일관성을 보이지 않을 수 있습니다. 따라서 대량 데이터를 집계하고 절대적인 정밀성이 필요하지 않은 분석 쿼리를 실행 중일 때만 일반적으로 이 설정을 활성화합니다. 이러한 종류의 장기 실행 쿼리와 함께 쓰기 집약적인 워크로드가 없는 경우에는 aurora_read_replica_read_committed 설정이 필요 없을 수 있습니다. 장기 실행 쿼리와 쓰기 집약적인 워크로드의 조합이 없으면 기록 목록 길이와 관련된 문제를 겪지 않을 가능성이 높습니다.

Example Aurora 복제본에서 READ COMMITTED에 대해 격리 동작을 보여주는 쿼리

다음 예시는 트랜잭션에서 연결된 테이블을 동시에 수정하는 경우 Aurora 복제본의 READ COMMITTED 쿼리에서 반복 불가능한 결과를 반환하는 방식을 보여줍니다. 쿼리 시작 전에 BIG_TABLE 테이블에는 1백만 개의 행이 포함되어 있습니다. 다른 데이터 조작 언어(DML) 문은 실행 중에 행을 추가, 제거 또는 변경합니다.

READ COMMITTED 격리 수준에서 Aurora 기본 인스턴스에 대한 쿼리를 통해 예측 가능한 결과가 산출됩니다. 그러나 모든 장기 실행 쿼리의 수명에 대해 일관성 있는 읽기 뷰를 유지하는 오버헤드로 인해 나중에 높은 가비지 수집 비용이 발생할 수 있습니다.

READ COMMITTED 격리 수준에서 Aurora 복제본에 대한 쿼리는 이러한 가비지 수집 오버헤드를 최소화도록 최적화됩니다. 하지만 쿼리가 실행 중일 때 커밋되는 트랜잭션에서 추가, 제거 또는 재편하는 행을 쿼리에서 가져오는지 여부에 따라 결과가 달라질 수 있다는 단점이 있습니다. 쿼리는 이러한 행을

고려하도록 허용될 뿐 반드시 고려해야 하는 것은 아닙니다. 데모용으로 쿼리에서는 `COUNT(*)` 함수를 사용해 테이블의 행 수만 확인합니다.

시간	Aurora 기본 인스턴스의 DML 문	READ COMMITTED로 Aurora 기본 인스턴스에 대해 쿼리	READ COMMITTED로 Aurora 복제본에 대해 쿼리
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1: `SELECT COUNT(*) FROM big_table;`	Q2: `SELECT COUNT(*) FROM big_table;`
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		
T4		Q1이 지금 완료되면 결과는 1,000,000입니다.	Q2가 지금 완료되면 결과는 1,000,000 또는 1,000,001입니다.
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		
T6		Q1이 지금 완료되면 결과는 1,000,000입니다.	Q2가 지금 완료되면 결과는 1,000,000 또는 1,000,001 또는 999,999 또는 999,998입니다.

시간	Aurora 기본 인스턴스의 DML 문	READ COMMITTED로 Aurora 기본 인스턴스에 대해 쿼리	READ COMMITTED로 Aurora 복제본에 대해 쿼리
T7	`UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;`		
T8		Q1이 지금 완료되면 결과는 1,000,000입니다.	Q2가 지금 완료되면 결과는 1,000,000 또는 1,000,001 또는 999,999이거나 이보다 더 높은 수 일 수도 있습니다.
T9		Q3: `SELECT COUNT(*) FROM big_table;`	Q4: `SELECT COUNT(*) FROM big_table;`
T10		Q3이 지금 완료되면 결과는 999,999입니다.	Q4가 지금 완료되면 결과는 999,999입니다.
T11		Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`

시간	Aurora 기본 인스턴스의 DML 문	READ COMMITTED로 Aurora 기본 인스턴스에 대해 쿼리	READ COMMITTED로 Aurora 복제본에 대해 쿼리
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		Q5가 지금 완료되면 결과는 0입니다.	Q6이 지금 완료되면 결과는 0 또는 1입니다.

다른 트랜잭션에서 DML 문을 수행하고 커밋하기 전에 쿼리가 빠르게 완료되면 결과는 예측 가능하며 기본 인스턴스와 Aurora 복제본 간에 동일합니다. 첫 번째 쿼리부터 시작하여 동작의 차이점을 자세히 살펴보겠습니다.

기본 인스턴스의 `READ COMMITTED`에서는 `REPEATABLE READ` 격리 수준과 유사한 강력한 일관성 모델을 사용하므로 Q1에 대한 결과는 예측 가능성이 매우 높습니다.

Q2에 대한 결과는 쿼리가 실행 중일 때 어떤 트랜잭션이 커밋되는가에 따라 달라질 수 있습니다. 예를 들어 쿼리가 실행 중일 때 다른 트랜잭션에서 DML 문을 수행하고 커밋한다고 가정합시다. 이 경우 격리 수준이 `READ COMMITTED`인 Aurora 복제본에 대한 쿼리에서는 이러한 변경 사항을 고려할 수도 하지 않을 수도 있습니다. 행 수는 `REPEATABLE READ` 격리 수준과 마찬가지로 예측할 수 없습니다. 또한 행 수를 기본 인스턴스 또는 RDS for MySQL 인스턴스의 `READ COMMITTED` 격리 수준에서 실행 중인 쿼리와 같은 수준으로 예측할 수는 없습니다.

T7의 `UPDATE` 문은 실제로 테이블의 행 수를 변경하지 않습니다. 그러나 이 문은 변수 길이 열의 길이를 변경함으로써 행이 내부적으로 재편되게 할 수 있습니다. 장기 실행 `READ COMMITTED` 트랜잭션에서는 행의 이전 버전이 생길 수 있고, 나중에 동일 쿼리 내에서는 동일 행의 새로운 버전이 생길 수 있습니다. 또한 쿼리는 행의 이전 버전과 새 버전을 모두 건너뛸 수 있으므로 행의 수가 예상과 다를 수 있습니다.

Q5 및 Q6의 결과는 서로 같거나 약간 다를 수 있습니다. `READ COMMITTED`에서 Aurora 복제본에 대한 쿼리 Q6에서는 쿼리가 실행 중일 때 커밋된 새 행이 반환될 수 있지만 반드시 반환되어야 하는 것은 아닙니다. 또한 테이블 하나에서 해당 행을 반환할 수 있지만 다른 테이블에서는 반환하지 않을 수 있습니다. 조인 쿼리가 두 테이블에서 일치하는 행을 찾지 못하는 경우 0이라는 수를 반환합니다. 이 쿼리가 `PARENT_TABLE` 및 `CHILD_TABLE` 모두에서 새 행을 찾지 못하면 쿼리는 1이라는 수를 반환합니다. 장기 실행 쿼리에서는 조인된 테이블에서의 조회는 간격이 크게 벌어진 여러 시점에 발생할 수 있습니다.

> **Note**
>
> 동작의 이러한 차이는 트랜잭션이 커밋되는 시점과 쿼리에서 기본 테이블 행을 처리하는 시점에 따라 달라집니다. 따라서 몇 분 또는 몇 시간이 걸리고 OLTP 트랜잭션을 동시에 처리하는 Aurora 클러스터에서 실행되는 보고서 쿼리에서 이러한 차이가 반환될 가능성이 높습니다. 이것은 Aurora 복제본의 `READ COMMITTED` 격리 수준에서 가장 큰 이익을 얻는 혼합 워크로드 유형입니다.

Aurora MySQL 힌트

Aurora MySQL 쿼리와 함께 SQL 힌트를 사용하여 성능을 미세 조정할 수 있습니다. 또한 힌트를 사용하여 예기치 않은 조건으로 인해 중요한 쿼리에 대한 실행 계획이 변경되는 것을 방지할 수 있습니다.

> **Tip**
>
> 힌트가 쿼리에 미치는 영향을 확인하려면 `EXPLAIN` 문에서 생성된 쿼리 계획을 검토합니다. 힌트가 있는 쿼리 계획과 없는 쿼리 계획을 비교합니다.

Aurora MySQL 버전 3에서는 MySQL Community Edition 8.0에서 사용 가능한 모든 힌트를 사용할 수 있습니다. 이러한 힌트에 대한 자세한 내용은 MySQL 참조 설명서의 [최적화 프로그램 힌트]를 참조하세요.

다음 힌트는 Aurora MySQL 버전 2에서 사용할 수 있습니다. 이러한 힌트는 Aurora MySQL 버전 2의 해시 조인 기능을 사용하는 쿼리, 특히 병렬 쿼리 최적화를 사용하는 쿼리에 적용됩니다.

PQ, NO_PQ

최적화 프로그램이 테이블별 또는 쿼리별로 병렬 쿼리를 사용하도록 할지 여부를 지정합니다.

PQ는 최적화 프로그램이 지정된 테이블 또는 전체 쿼리(블록)에 대해 병렬 쿼리를 사용하도록 합니다. NO_PQ는 최적화 프로그램이 지정된 테이블 또는 전체 쿼리(블록)에 대해 병렬 쿼리를 사용하지 못하도록 합니다.

이 힌트는 Aurora MySQL 버전 2.11 이상에서 사용할 수 있습니다. 다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

> **Note**
>
> 테이블 이름을 지정하면 최적화 프로그램은 이러한 선별된 테이블에만 PQ/NO_PQ 힌트를 적용합니다. 테이블 이름을 지정하지 않으면 쿼리 블록의 영향을 받는 모든 테이블에 PQ/NO_PQ 힌트가 적용됩니다.

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN, NO_HASH_JOIN

쿼리에 해시 조인 최적화 방법을 사용할지 여부를 선택할 수 있는 병렬 쿼리 최적화 프로그램의 기능을 설정하거나 해제합니다. HASH_JOIN을 사용하면 해당 메커니즘이 더 효율적인 경우 최적화 프로그램이 해시 조인을 사용할 수 있습니다. NO_HASH_JOIN은 최적화 프로그램이 쿼리에 해시 조인을 사용하지 못하게 합니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다. Aurora MySQL 버전 3에서는 아무런 효과가 없습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING, NO_HASH_JOIN_PROBING

해시 조인 쿼리에서 조인의 프로브 측에 지정된 테이블을 사용할지를 지정합니다. 쿼리는 프로브 테이블의 전체 내용을 읽는 대신 빌드 테이블의 열 값이 프로브 테이블에 있는지를 테스트합니다. `HASH_JOIN_PROBING` 및 `HASH_JOIN_BUILDING`을 사용하여 쿼리 텍스트 내에서 테이블을 재정렬하지 않고 해시 조인 쿼리가 처리되는 방법을 지정할 수 있습니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다. Aurora MySQL 버전 3에서는 아무런 효과가 없습니다.

다음 예제에서는 이 힌트를 사용하는 방법을 보여 줍니다. `HASH_JOIN_PROBING` 테이블에 `T2` 힌트를 지정하면 `NO_HASH_JOIN_PROBING` 테이블에 `T1`을 지정하는 것과 같은 효과가 있습니다.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING, NO_HASH_JOIN_BUILDING

해시 조인 쿼리에서 조인의 빌드 측에 지정된 테이블을 사용할지를 지정합니다. 쿼리는 이 테이블의 모든 행을 처리하여 다른 테이블과 상호 참조할 열 값 목록을 작성합니다. `HASH_JOIN_PROBING` 및 `HASH_JOIN_BUILDING`을 사용하여 쿼리 텍스트 내에서 테이블을 재정렬하지 않고 해시 조인 쿼리가 처리되는 방법을 지정할 수 있습니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다. Aurora MySQL 버전 3에서는 아무런 효과가 없습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다. `HASH_JOIN_BUILDING` 테이블에 `T2` 힌트를 지정하면 `NO_HASH_JOIN_BUILDING` 테이블에 `T1`을 지정하는 것과 같은 효과가 있습니다.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

쿼리의 테이블이 쿼리에 나열된 순서에 따라 조인되도록 지정합니다. 이는 세 개 이상의 테이블을 포함하는 쿼리에 유용합니다. 이는 MySQL STRAIGHT_JOIN 힌트를 대체하기 위한 것이며, MySQL JOIN_FIXED_ORDER 힌트에 해당합니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

쿼리의 테이블에 대한 조인 순서를 지정합니다. 이는 세 개 이상의 테이블을 포함하는 쿼리에 유용합니다. 이는 MySQL JOIN_ORDER 힌트에 해당합니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

조인 순서에서 먼저 넣을 테이블을 지정합니다. 이는 세 개 이상의 테이블을 포함하는 쿼리에 유용합니다. 이는 MySQL JOIN_PREFIX 힌트에 해당합니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

조인 순서에 마지막으로 넣을 테이블을 지정합니다. 이는 세 개 이상의 테이블을 포함하는 쿼리에 유용합니다. 이는 MySQL JOIN_SUFFIX 힌트에 해당합니다. 이 힌트는 Aurora MySQL 버전 2.08 이상에서 사용할 수 있습니다.

다음 예시에서는 이 힌트를 사용하는 방법을 보여줍니다.

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

해시 조인 쿼리 사용에 대한 자세한 내용은 해시 조인을 사용하여 대규모 Aurora MySQL 조인 쿼리 최적화 단원을 참조하십시오.

Aurora MySQL 저장 프로시저 참조

기본 제공 저장 프로시저를 호출하여 Aurora MySQL DB 클러스터를 관리할 수 있습니다.

주제

글로벌 상태 기록 수집 및 유지

Amazon RDS는 시간에 따른 상태 변수 값의 스냅샷을 생성하고 이를 마지막 스냅샷 이후의 변경 사항과 함께 테이블에 쓰는 일련의 프로시저를 제공합니다. 이 인프라를 전역적 상태 이력이라고 합니다. 자세한 내용은 전역적 상태 이력 관리를 참조하세요.

다음 저장 프로시저는 전역적 상태 이력을 수집하고 유지하는 방법을 관리합니다.

주제

mysql.rds_collect_global_status_history

전역적 상태 이력에 대해 요청 시 스냅샷을 생성합니다.

구문

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

전역적 상태 이력에서 생성한 스냅샷을 비활성화합니다.

구문

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

`mysql.global_status_history` 테이블 회전을 비활성화합니다.

구문

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

전역적 상태 이력을 활성화하여 `rds_set_gsh_collector`로 지정한 간격에 따라 기본 스냅샷을 생성합니다.

구문

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

`mysql.global_status_history` 테이블의 내용이 `rds_set_gsh_rotation`에서 설정한 주기에 따라 `mysql.global_status_history_old`로 회전되도록 회전을 활성화합니다.

구문

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

필요에 따라 `mysql.global_status_history` 테이블의 내용을 `mysql.global_status_history_old`로 로테이션합니다.

구문

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

전역적 상태 이력에서 생성하는 스냅샷 간격(분)을 지정합니다.

구문

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

파라미터

intervalPeriod

스냅샷 주기(분)입니다. 기본값은 5입니다.

mysql.rds_set_gsh_rotation

`mysql.global_status_history` 테이블의 로테이션 주기(일)을 지정합니다.

구문

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

파라미터

intervalPeriod

테이블 로테이션 주기(일)입니다. 기본값은 7입니다.

이진수 로그(binlog) 복제 구성, 시작 및 중지

Aurora MySQL 클러스터에 있는 기본 인스턴스에 연결된 상태에서 다음과 같은 저장 프로시저를 호출할 수 있습니다. 이 프로시저에서는 트랜잭션이 외부 데이터베이스에서 Aurora MySQL로, 또는 Aurora MySQL에서 외부 데이터베이스로 복제되는 방식을 제어합니다.

주제

mysql.rds_disable_session_binlog(Aurora MySQL 버전 2)

`sql_log_bin` 변수를 `OFF`로 설정하여 현재 세션에 대한 이진 로깅을 끕니다.

구문

```
CALL mysql.rds_disable_session_binlog;
```

파라미터

없음

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

Aurora의 경우, 이 프로시저는 Aurora MySQL 버전 2.12 이상과 MySQL 5.7 호환 버전에서 지원됩니다.

> **Note**
>
> Aurora MySQL 버전 3에서는 `SESSION_VARIABLES_ADMIN` 권한이 있는 경우 다음 명령을 사용하여 현재 세션에 대한 바이너리 로깅을 비활성화할 수 있습니다.
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog(Aurora MySQL 버전 2)

`sql_log_bin` 변수를 ON으로 설정하여 현재 세션에 대한 이진 로깅을 켭니다.

구문

```
CALL mysql.rds_enable_session_binlog;
```

파라미터

없음

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

Aurora의 경우, 이 프로시저는 Aurora MySQL 버전 2.12 이상과 MySQL 5.7 호환 버전에서 지원됩니다.

> **Note**
>
> Aurora MySQL 버전 3에서는 `SESSION_VARIABLES_ADMIN` 권한이 있는 경우 다음 명령을 사용하여 현재 세션에 대한 바이너리 로깅을 활성화할 수 있습니다.
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_import_binlog_ssl_material

인증 기관(CA) 인증서, 클라이언트 인증서, 클라이언트 키를 Aurora MySQL DB 클러스터로 가져옵니다. SSL 통신과 암호화된 복제에 필요한 정보입니다.

> **Note**
>
> 현재 이 프로시저는 Aurora MySQL 버전 2 중 2.09.2, 2.10.0, 2.10.1, 2.11.0 및 버전 3 중 3.01.1 이상에서 지원됩니다.

구문

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

파라미터

`ssl_material`

MySQL 클라이언트를 위한 다음 .pem 형식 파일 내용이 포함된 JSON 페이로드.

- "ssl_ca":"*`Certificate authority certificate`*"
- "ssl_cert":"*`Client certificate`*"
- "ssl_key":"*`Client key`*"

사용 노트

이 프로시저를 실행하기 전에 암호화된 복제를 준비합니다.

- 외부 MySQL 소스 데이터베이스 인스턴스에서 SSL을 활성화하지 않았고 클라이언트 키와 클라이언트 인증서가 준비되지 않았다면, MySQL 데이터베이스 서버의 SSL을 활성화하고 필요한 클라이언트 키와 클라이언트 인증서를 생성합니다.
- 외부 소스 데이터베이스 인스턴스에 SSL이 활성화되어 있으면 Aurora MySQL DB 클러스터에 클라이언트 키와 인증서를 제공합니다. 인증서와 키가 없다면, Aurora MySQL DB 클러스터에 대해 새 키와 인증서를 생성합니다. 클라이언트 인증서에 서명하려면, 외부 MySQL 소스 데이터베이스 인스턴스에 SSL을 구성할 때 사용하는 인증 기관(CA) 키가 있어야 합니다.

자세한 내용은 MySQL 설명서의 [openssl을 사용하여 SSL 인증서 및 키 생성]을 참조하십시오.

> **Important**
>
> 암호화된 복제를 준비한 후, SSL 연결을 사용해 이 프로시저를 실행합니다. 클라이언트 키를 보안성이 낮은 연결을 이용해 전송할 수 없습니다.

이 프로시저는 외부 MySQL 데이터베이스의 SSL 정보를 Aurora MySQL DB 클러스터로 가져옵니다. SSL 정보는 Aurora MySQL DB 클러스터에 대한 SSL 정보가 포함되어 있는 .pem 형식의 파일입니다. 암호화 복제 중, Aurora MySQL DB 클러스터는 MySQL 데이터베이스 서버의 클라이언트 역할을 합니다. Aurora MySQL 클라이언트 인증서와 키는 .pem 형식의 파일입니다.

이들 파일에서 올바른 JSON 페이로드의 `ssl_material` 파라미터로 정보를 복사할 수 있습니다. 암호화된 복제를 지원하려면 Aurora MySQL DB 클러스터로 이 SSL 정보를 가져옵니다.

JSON 페이로드는 다음 형식이어야 합니다.

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

예시

다음은 Aurora MySQL에 SSL 정보를 가져오는 예시입니다. .pem 형식 파일은 본문 코드의 길이가 일반적으로 예제의 본문 코드 길이보다 깁니다.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log(Aurora MySQL 버전 2)

소스 데이터베이스 인스턴스 로그 위치를 소스 데이터베이스 인스턴스에서 다음 이진 로그의 시작으로 변경합니다. 읽기 전용 복제본에 대한 복제 I/O 오류 1236을 수신 중일 경우에만 이 프로시저를 사용하십시오.

구문

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

파라미터

curr_master_log

현재 마스터 로그 파일의 인덱스입니다. 예를 들어, 현재 파일의 이름이 mysql-bin-changelog.012345이면 인덱스는 12345입니다. 현재 마스터 로그 파일 이름을 확인하려면 SHOW REPLICA STATUS 명령을 실행하고 Master_Log_File 필드를 봅니다.

사용 노트

마스터 사용자는 mysql.rds_next_master_log 프로시저를 실행해야 합니다.

> **Warning**
>
> 복제 원본인 다중 AZ DB 인스턴스의 장애 조치 후 복제가 실패하고 mysql.rds_next_master_log의 Last_IO_Errno필드에서 I/O 오류 1236을 보고하는 경우에만 SHOW REPLICA STATUS를 호출합니다.
>
> 장애 조치 이벤트가 발생하기 전에 원본 인스턴스의 트랜잭션이 디스크의 바이너리 로그에 기록되지 않은 경우 mysql.rds_next_master_log를 호출하면 읽기 전용 복제본에서 데이터가 손실될 수 있습니다.

예시

Aurora MySQL 읽기 전용 복제본에서 복제가 실패한다고 가정해 보겠습니다. 읽기 전용 복제본에서 SHOW REPLICA STATUS\G를 실행하면 다음 결과가 반환됩니다.

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
```

```
               Replicate_Do_DB:
           Replicate_Ignore_DB:
            Replicate_Do_Table:
        Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
   Replicate_Wild_Ignore_Table:
                    Last_Errno: 0
                    Last_Error:
                  Skip_Counter: 0
           Exec_Source_Log_Pos: 30223232
               Relay_Log_Space: 5248928866
               Until_Condition: None
                Until_Log_File:
                 Until_Log_Pos: 0
            Source_SSL_Allowed: No
            Source_SSL_CA_File:
            Source_SSL_CA_Path:
               Source_SSL_Cert:
             Source_SSL_Cipher:
                Source_SSL_Key:
         Seconds_Behind_Master: NULL
 Source_SSL_Verify_Server_Cert: No
                 Last_IO_Errno: 1236
                 Last_IO_Error: Got fatal error 1236 from master when reading data from
  binary log: 'Client requested master to start replication from impossible position;
  the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
  '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
 rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
                Last_SQL_Errno: 0
                Last_SQL_Error:
   Replicate_Ignore_Server_Ids:
              Source_Server_Id: 67285976
```

`Last_IO_Errno` 필드가 인스턴스에서 I/O 오류 1236을 수신하고 있음을 보여 줍니다. `Master_Log_File` 필드는 파일 이름이 `mysql-bin-changelog.012345`임을 보여 줍니다. 이는 로그 파일 인덱스가 `12345`임을 의미합니다. 오류를 해결하려면 다음 파라미터와 함께 `mysql.rds_next_master_log`를 호출합니다.

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log(Aurora MySQL 버전 3)

소스 데이터베이스 인스턴스 로그 위치를 소스 데이터베이스 인스턴스에서 다음 이진 로그의 시작으로 변경합니다. 읽기 전용 복제본에 대한 복제 I/O 오류 1236을 수신 중일 경우에만 이 프로시저를 사용하십시오.

구문

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

파라미터

curr_source_log

현재 소스 로그 파일의 인덱스입니다. 예를 들어, 현재 파일의 이름이 mysql-bin-changelog.012345이면 인덱스는 12345입니다. 현재 소스 로그 파일 이름을 확인하려면 SHOW REPLICA STATUS 명령을 실행하고 Source_Log_File 필드를 봅니다.

사용 노트

관리 사용자는 mysql.rds_next_source_log 프로시저를 실행해야 합니다.

> **Warning**
> 복제 원본인 다중 AZ DB 인스턴스의 장애 조치 후 복제가 실패하고 mysql.rds_next_source_log의 Last_IO_Errno필드에서 I/O 오류 1236을 보고하는 경우에만 SHOW REPLICA STATUS를 호출합니다.
> 장애 조치 이벤트가 발생하기 전에 원본 인스턴스의 트랜잭션이 디스크의 바이너리 로그에 기록되지 않은 경우 mysql.rds_next_source_log를 호출하면 읽기 전용 복제본에서 데이터가 손실될 수 있습니다. 소스 인스턴스 파라미터 innodb_support_xa 및 sync_binlog를 1로 설정하여 이런 상황이 발생할 가능성을 줄일 수 있으나, 성능이 저하될 수 있습니다.

예시

Aurora MySQL 읽기 전용 복제본에서 복제가 실패한다고 가정해 보겠습니다. 읽기 전용 복제본에서 SHOW REPLICA STATUS\G를 실행하면 다음 결과가 반환됩니다.

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
```

```
  Replicate_Ignore_Server_Ids:
               Source_Server_Id: 67285976
```

`Last_IO_Errno` 필드가 인스턴스에서 I/O 오류 1236을 수신하고 있음을 보여 줍니다. `Source_Log_File` 필드는 파일 이름이 `mysql-bin-changelog.012345`임을 보여 줍니다. 이는 로그 파일 인덱스가 `12345`임을 의미합니다. 오류를 해결하려면 다음 파라미터와 함께 `mysql.rds_next_source_log`를 호출합니다.

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

SSL 통신과 암호화된 복제용 인증 기관(CA) 인증서, 클라이언트 인증서, 클라이언트 키를 제거합니다. [mysql.rds_import_binlog_ssl_material](#)를 사용하여, 이 정보를 가져옵니다.

구문

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master(Aurora MySQL 버전 2)

이제 Aurora MySQL DB 인스턴스가 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되지 않도록 다시 구성합니다.

> **Important**
>
> 이 프로시저를 실행하려면 `autocommit`를 활성화해야 합니다. 활성화하려면 `autocommit` 파라미터를 `1`로 설정합니다. 파라미터 수정에 대한 자세한 정보는 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정](#) 단원을 참조하십시오.

구문

```
CALL mysql.rds_reset_external_master;
```

사용 노트

마스터 사용자는 `mysql.rds_reset_external_master` 프로시저를 실행해야 합니다. 이 프로시저는 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본으로 제거되도록 MySQL DB 인스턴스에서 실행해야 합니다.

> **Note**
>
> 이러한 저장 프로시저는 주로 Amazon RDS 외부에서 실행되는 MySQL 인스턴스를 사용하여 복제할 수 있도록 하기 위한 것입니다. 가능하면 Aurora 복제본을 사용하여 Aurora MySQL DB 클러스터 내에서 복제를 관리하는 것이 좋습니다. Aurora MySQL DB 클러스터의 복제 관리에 대한 자세한 내용은 [Aurora 복제본 사용](#) 단원을 참조하세요.

복제를 사용하여 Aurora MySQL 외부에서 실행 중인 MySQL 인스턴스에서 데이터를 가져오는 방법에 대한 자세한 내용은 [Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제](#) 섹션을 참조하세요.

mysql.rds_reset_external_source(Aurora MySQL 버전 3)

이제 Aurora MySQL DB 인스턴스가 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되지 않도록 다시 구성합니다.

> **Important**
>
> 이 프로시저를 실행하려면 `autocommit`를 활성화해야 합니다. 활성화하려면 `autocommit` 파라미터를 1로 설정합니다. 파라미터 수정에 대한 자세한 정보는 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정](#) 단원을 참조하십시오.

구문

```
CALL mysql.rds_reset_external_source;
```

사용 노트

관리 사용자는 `mysql.rds_reset_external_source` 프로시저를 실행해야 합니다. 이 프로시저는 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본으로 제거되도록 MySQL DB 인스턴스에서 실행해야 합니다.

> **Note**
>
> 이러한 저장 프로시저는 주로 Amazon RDS 외부에서 실행되는 MySQL 인스턴스를 사용하여 복제할 수 있도록 하기 위한 것입니다. 가능하면 Aurora 복제본을 사용하여 Aurora MySQL DB 클러스터 내에서 복제를 관리하는 것이 좋습니다. Aurora MySQL DB 클러스터의 복제 관리에 대한 자세한 내용은 Aurora 복제본 사용 단원을 참조하세요.

mysql.rds_set_binlog_source_ssl(Aurora MySQL 버전 3)

Binlog 복제를 위한 `SOURCE_SSL` 암호화를 활성화합니다. 자세한 내용은 MySQL 설명서의 CHANGE REPLICATION SOURCE TO statement를 참조하세요.

구문

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

파라미터

mode

`SOURCE_SSL` 암호화가 활성화되었는지를 나타내는 값입니다.

- `0` - `SOURCE_SSL` 암호화가 비활성화되었습니다. 기본값은 `0`입니다.
- `1` - `SOURCE_SSL` 암호화가 활성화었습니다. SSL 또는 TLS를 사용하여 암호화를 구성할 수 있습니다.

사용 노트

이 프로시저는 Aurora MySQL 버전 3.06 이상에서 지원됩니다.

mysql.rds_set_external_master(Aurora MySQL 버전 2)

Aurora MySQL DB 인스턴스가 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되도록 구성합니다.

`mysql.rds_set_external_master` 프로시저는 사용되지 않으며 향후 릴리스에서 삭제됩니다. 대신 `mysql.rds_set_external_source`을 사용하세요.

> **Important**
>
> 이 프로시저를 실행하려면 `autocommit`를 활성화해야 합니다. 활성화하려면 `autocommit` 파라미터를 1로 설정합니다. 파라미터 수정에 대한 자세한 정보는 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정](#) 단원을 참조하십시오.

구문

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

파라미터

host_name

소스 데이터베이스 인스턴스가 될 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 호스트 이름 또는 IP 주소입니다.

host_port

소스 데이터베이스 인스턴스로 구성될 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 사용하는 포트입니다. 네트워크 구성에 포트 번호를 변환하는 SSH 포트 복제가 포함되는 경우 SSH(Secure Shell)에 의해 공개되는 포트 이름을 지정하십시오.

replication_user_name

Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 보유한 사용자의 ID입니다. 외부 인스턴스를 사용한 복제에만 사용되는 계정을 제공하는 것이 좋습니다.

replication_user_password

`replication_user_name`에 지정된 사용자 ID의 암호입니다.

mysql_binary_log_file_name

복제 정보를 포함하는 소스 데이터베이스 인스턴스의 이진 로그 이름입니다.

mysql_binary_log_file_location

복제 시 복제 정보를 읽기 시작하는 `mysql_binary_log_file_name` 이진수 로그 내 위치입니다.

소스 데이터베이스 인스턴스의 `SHOW MASTER STATUS`를 실행하여 binlog 파일 이름 및 위치를 확인할 수 있습니다.

ssl_encryption

복제 연결에 보안 소켓 계층(SSL) 암호화를 사용할지 여부를 지정하는 값입니다. 1은 SSL 암호화 사용, 0은 암호화 사용 안 함입니다. 기본값은 0입니다.

> **Note**
>
> `MASTER_SSL_VERIFY_SERVER_CERT` 옵션은 지원되지 않습니다. 이 옵션은 0으로 설정되어 있는데, 이는 연결이 암호화되었지만 인증서는 확인되지 않았음을 의미합니다.

사용 노트

마스터 사용자는 `mysql.rds_set_external_master` 프로시저를 실행해야 합니다. 이 프로시저는 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본으로 구성되도록 MySQL DB 인스턴스에서 실행해야 합니다.

`mysql.rds_set_external_master`을 실행하기 전에 소스 데이터베이스 인스턴스가 되도록 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스를 구성해야 합니다. Amazon RDS 외부에서 실행하는 MySQL 인스턴스에 연결하려면 MySQL의 외부 인스턴스에서 `replication_user_name` 및 `replication_user_password` 권한이 있는 복제 사용자를 나타내는 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 값을 지정해야 합니다.

MySQL의 외부 인스턴스를 소스 데이터베이스 인스턴스로 구성하려면

1. 선택한 MySQL 클라이언트를 사용하여 MySQL의 외부 인스턴스에 연결하고 복제에 사용될 사용자 계정을 생성합니다. 다음은 예입니다.

   MySQL 5.7

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

MySQL 8.0

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
 'password';
```

> **Note**
> 보안 모범 사례로 여기에 표시된 프롬프트 이외의 암호를 지정하는 것이 좋습니다.

2. MySQL의 외부 인스턴스에서 복제 사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다. 다음 예제에서는 도메인의 'repl_user' 사용자에게 모든 데이터베이스에 대한 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다.

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

암호화된 복제를 사용하려면 SSL 연결을 사용하도록 소스 데이터베이스 인스턴스를 구성합니다. 또한 mysql.rds_import_binlog_ssl_material 프로시저를 사용하여 인증 기관(CA) 인증서, 클라이언트 인증서, 클라이언트 키를 DB 인스턴스나 DB 클러스터로 가져옵니다.

> **Note**
> 이러한 저장 프로시저는 주로 Amazon RDS 외부에서 실행되는 MySQL 인스턴스를 사용하여 복제할 수 있도록 하기 위한 것입니다. 가능하면 Aurora 복제본을 사용하여 Aurora MySQL DB 클러스터 내에서 복제를 관리하는 것이 좋습니다. Aurora MySQL DB 클러스터의 복제 관리에 대한 자세한 내용은 Aurora 복제본 사용 단원을 참조하세요.

mysql.rds_set_external_master을 호출하여 Amazon RDS DB 인스턴스를 읽기 전용 복제본으로 구성한 후 읽기 전용 복제본에서 mysql.rds_start_replication을 호출하여 복제 프로세스를 시작할 수 있습니다. mysql.rds_reset_external_master(Aurora MySQL 버전 2)를 호출하여 읽기 전용 복제본 구성을 제거할 수 있습니다.

mysql.rds_set_external_master를 호출하면 Amazon RDS는 set master의 시간, 사용자 및 작업을 mysql.rds_history 및 mysql.rds_replication_status 테이블에 기록합니다.

예시

MySQL DB 인스턴스에서 다음 예제를 실행하면 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되도록 DB 인스턴스가 구성됩니다.

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source(Aurora MySQL 버전 3)

Aurora MySQL DB 인스턴스가 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되도록 구성합니다.

> **Important**
>
> 이 프로시저를 실행하려면 autocommit를 활성화해야 합니다. 활성화하려면 autocommit 파라미터를 1로 설정합니다. 파라미터 수정에 대한 자세한 정보는 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하십시오.

구문

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
```

```
   , replication_user_password
   , mysql_binary_log_file_name
   , mysql_binary_log_file_location
   , ssl_encryption
);
```

파라미터

host_name

소스 데이터베이스 인스턴스가 될 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 호스트 이름 또는 IP 주소입니다.

host_port

소스 데이터베이스 인스턴스로 구성될 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 사용하는 포트입니다. 네트워크 구성에 포트 번호를 변환하는 SSH 포트 복제가 포함되는 경우 SSH(Secure Shell)에 의해 공개되는 포트 이름을 지정하십시오.

replication_user_name

Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 보유한 사용자의 ID입니다. 외부 인스턴스를 사용한 복제에만 사용되는 계정을 제공하는 것이 좋습니다.

replication_user_password

`replication_user_name`에 지정된 사용자 ID의 암호입니다.

mysql_binary_log_file_name

복제 정보를 포함하는 소스 데이터베이스 인스턴스의 이진 로그 이름입니다.

mysql_binary_log_file_location

복제 시 복제 정보를 읽기 시작하는 `mysql_binary_log_file_name` 이진수 로그 내 위치입니다.

소스 데이터베이스 인스턴스의 `SHOW MASTER STATUS`를 실행하여 binlog 파일 이름 및 위치를 확인할 수 있습니다.

ssl_encryption

복제 연결에 보안 소켓 계층(SSL) 암호화를 사용할지 여부를 지정하는 값입니다. 1은 SSL 암호화 사용, 0은 암호화 사용 안 함입니다. 기본값은 0입니다.

> **Note**
>
> 이 옵션을 활성화하려면 mysql.rds_import_binlog_ssl_material을 사용하여 사용자 지정 SSL 인증서를 가져와야 합니다. 사용자 지정 SSL 인증서를 가져오지 않은 경우 이 파라미터를 0으로 설정하고 mysql.rds_set_binlog_source_ssl(Aurora MySQL 버전 3)을 사용하여 binlog 복제를 위한 SSL을 활성화하세요.
> `SOURCE_SSL_VERIFY_SERVER_CERT` 옵션은 지원되지 않습니다. 이 옵션은 0으로 설정되어 있는데, 이는 연결이 암호화되었지만 인증서는 확인되지 않았음을 의미합니다.

사용 노트

관리 사용자는 `mysql.rds_set_external_source` 프로시저를 실행해야 합니다. 이 프로시저는 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본으로 구성되도록 Aurora MySQL DB 인스턴스에서 실행해야 합니다.

`mysql.rds_set_external_source`을 실행하기 전에 소스 데이터베이스 인스턴스가 되도록 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스를 구성해야 합니다. Amazon RDS 외부에서 실행하는 MySQL 인스턴스에 연결하려면 MySQL의 외부 인스턴스에서 `replication_user_name` 및 `replication_user_password` 권한이 있는 복제 사용자를 나타내는 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 값을 지정해야 합니다.

MySQL의 외부 인스턴스를 소스 데이터베이스 인스턴스로 구성하려면

1. 선택한 MySQL 클라이언트를 사용하여 MySQL의 외부 인스턴스에 연결하고 복제에 사용될 사용자 계정을 생성합니다. 다음은 예입니다.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> 보안 모범 사례로 여기에 표시된 프롬프트 이외의 암호를 지정하는 것이 좋습니다.

2. MySQL의 외부 인스턴스에서 복제 사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다. 다음 예제에서는 도메인의 'repl_user' 사용자에게 모든 데이터베이스에 대한 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

암호화된 복제를 사용하려면 SSL 연결을 사용하도록 소스 데이터베이스 인스턴스를 구성합니다. 또한 mysql.rds_import_binlog_ssl_material 프로시저를 사용하여 인증 기관(CA) 인증서, 클라이언트 인증서, 클라이언트 키를 DB 인스턴스나 DB 클러스터로 가져옵니다.

> **Note**
>
> 이러한 저장 프로시저는 주로 Amazon RDS 외부에서 실행되는 MySQL 인스턴스를 사용하여 복제할 수 있도록 하기 위한 것입니다. 가능하면 Aurora 복제본을 사용하여 Aurora MySQL DB 클러스터 내에서 복제를 관리하는 것이 좋습니다. Aurora MySQL DB 클러스터의 복제 관리에 대한 자세한 내용은 Aurora 복제본 사용 단원을 참조하세요.

`mysql.rds_set_external_source`를 직접 호출하여 Aurora MySQL DB 인스턴스를 읽기 전용 복제본으로 구성한 후 읽기 전용 복제본에서 mysql.rds_start_replication을 직접 호출하여 복제 프로세스를 시작할 수 있습니다. mysql.rds_reset_external_source(Aurora MySQL 버전 3)를 호출하여 읽기 전용 복제본 구성을 제거할 수 있습니다.

`mysql.rds_set_external_source`를 호출하면 Amazon RDS는 `set master`의 시간, 사용자 및 작업을 `mysql.rds_history` 및 `mysql.rds_replication_status` 테이블에 기록합니다.

예시

Aurora MySQL DB 인스턴스에서 실행할 경우 다음 예시는 Amazon RDS 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본이 되도록 DB 인스턴스를 구성합니다.

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position(Aurora MySQL 버전 2)

Aurora MySQL 기본 인스턴스가 외부 MySQL 인스턴스에서 수신되는 복제를 수락하도록 구성하십시오. 또한 이 절차에서는 전역 트랜잭션 식별자(GTID)를 기반으로 한 복제를 구성합니다.

Aurora MySQL은 지연된 복제를 지원하지 않기 때문에 이 프로시저는 지연된 복제를 구성하지 않습니다.

구문

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

파라미터

host_name

복제 소스가 될 Aurora의 외부에서 실행 중인 MySQL 인스턴스의 호스트 이름 또는 IP 주소입니다.

host_port

복제 소스로 구성될 Aurora 외부에서 실행 중인 MySQL 인스턴스에서 사용하는 포트입니다. 네트워크 구성에 포트 번호를 변환하는 SSH 포트 복제가 포함되는 경우 SSH(Secure Shell)에 의해 공개되는 포트 이름을 지정하세요.

replication_user_name

Aurora 외부에서 실행하는 MySQL 인스턴스에서 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 보유한 사용자의 ID입니다. 외부 인스턴스를 사용한 복제에만 사용되는 계정을 제공하는 것이 좋습니다.

replication_user_password

`replication_user_name`에 지정된 사용자 ID의 암호입니다.

ssl_encryption

이 옵션은 현재 구현되지 않았습니다. 기본값은 0입니다.

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

마스터 사용자는 `mysql.rds_set_external_master_with_auto_position` 프로시저를 실행해야 합니다. 마스터 사용자는 복제 대상의 역할을 하는 Aurora MySQL DB 클러스터의 기본 인스턴스에

서 이 절차를 실행합니다. 이것은 외부 MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터의 복제 대상이 될 수 있습니다.

이 프로시저는 Aurora MySQL 버전 2에서 지원됩니다. Aurora MySQL 버전 3의 경우 대신 mysql.rds_set_external_source_with_auto_position(Aurora MySQL 버전 3) 프로시저를 사용하세요.

`mysql.rds_set_external_master_with_auto_position`을 실행하기 전에 외부 MySQL DB 인스턴스가 복제 소스가 되도록 구성합니다. 외부 MySQL 인스턴스에 연결하려면 `replication_user_name` 및 `replication_user_password`의 값을 지정해야 합니다. 이러한 값은 외부 MySQL의 외부 인스턴스에 대해 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 가진 복제 사용자를 나타내야 합니다.

외부 MySQL 인스턴스를 복제 소스로 구성하려면

1. 선택한 MySQL 클라이언트를 사용하여 외부 MySQL 인스턴스에 연결하고 복제에 사용할 사용자 계정을 생성합니다. 다음은 예제입니다.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. 외부 MySQL 인스턴스의 경우 복제 사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다. 다음 예제에서는 도메인의 `REPLICATION CLIENT` 사용자에게 모든 데이터베이스에 대한 `REPLICATION SLAVE` 및 `'repl_user'` 권한을 부여합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

`mysql.rds_set_external_master_with_auto_position`을 호출하면 Amazon RDS는 특정 정보를 기록합니다. 이 정보는 `"set master"` 및 `mysql.rds_history` 테이블에 있는 `mysql.rds_replication_status`의 시간, 사용자 및 작업입니다.

문제의 원인으로 알려진 특정 GTID 기반 트랜잭션을 건너뛰려면 mysql.rds_skip_transaction_with_gtid(Aurora MySQL 버전 2 및 3) 저장 프로시저를 사용할 수 있습니다. GTID 기반 복제 작업에 대한 자세한 내용은 GTID 기반 복제 사용 단원을 참조하세요.

예시

Aurora 기본 인스턴스에서 다음 예제를 실행하면 Aurora 클러스터가 Aurora 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본의 역할을 하도록 구성됩니다.

```
call mysql.rds_set_external_master_with_auto_position(
```

```
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position(Aurora MySQL 버전 3)

Aurora MySQL 기본 인스턴스가 외부 MySQL 인스턴스에서 수신되는 복제를 수락하도록 구성하십시오. 또한 이 절차에서는 전역 트랜잭션 식별자(GTID)를 기반으로 한 복제를 구성합니다.

구문

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

파라미터

host_name

복제 소스가 될 Aurora의 외부에서 실행 중인 MySQL 인스턴스의 호스트 이름 또는 IP 주소입니다.

host_port

복제 소스로 구성될 Aurora 외부에서 실행 중인 MySQL 인스턴스에서 사용하는 포트입니다. 네트워크 구성에 포트 번호를 변환하는 SSH 포트 복제가 포함되는 경우 SSH(Secure Shell)에 의해 공개되는 포트 이름을 지정하세요.

replication_user_name

Aurora 외부에서 실행하는 MySQL 인스턴스에서 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 보유한 사용자의 ID입니다. 외부 인스턴스를 사용한 복제에만 사용되는 계정을 제공하는 것이 좋습니다.

replication_user_password

`replication_user_name`에 지정된 사용자 ID의 암호입니다.

ssl_encryption

이 옵션은 현재 구현되지 않았습니다. 기본값은 0입니다.

> **Note**
>
> Binlog를 복제하려면 [mysql.rds_set_binlog_source_ssl(Aurora MySQL 버전 3)](#)을 사용하여 SSL을 활성화합니다.

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

관리 사용자는 `mysql.rds_set_external_source_with_auto_position` 프로시저를 실행해야 합니다. 관리 사용자는 복제 대상의 역할을 하는 Aurora MySQL DB 클러스터의 기본 인스턴스에서 이 프로시저를 실행합니다. 이것은 외부 MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터의 복제 대상이 될 수 있습니다.

이 프로시저는 Aurora MySQL 버전 3에서 지원됩니다. Aurora MySQL은 지연된 복제를 지원하지 않기 때문에 이 프로시저는 지연된 복제를 구성하지 않습니다.

`mysql.rds_set_external_source_with_auto_position`을 실행하기 전에 외부 MySQL DB 인스턴스가 복제 소스가 되도록 구성합니다. 외부 MySQL 인스턴스에 연결하려면 `replication_user_name` 및 `replication_user_password`의 값을 지정해야 합니다. 이러한 값은 외부 MySQL의 외부 인스턴스에 대해 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 가진 복제 사용자를 나타내야 합니다.

외부 MySQL 인스턴스를 복제 소스로 구성하려면

1. 선택한 MySQL 클라이언트를 사용하여 외부 MySQL 인스턴스에 연결하고 복제에 사용할 사용자 계정을 생성합니다. 다음은 예제입니다.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. 외부 MySQL 인스턴스의 경우 복제 사용자에게 `REPLICATION CLIENT` 및 `REPLICATION SLAVE` 권한을 부여합니다. 다음 예제에서는 도메인의 `REPLICATION CLIENT` 사용자에게 모든 데이터베이스에 대한 `REPLICATION SLAVE` 및 `'repl_user'` 권한을 부여합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

`mysql.rds_set_external_source_with_auto_position`을 호출하면 Amazon RDS는 특정 정보를 기록합니다. 이 정보는 `"set master"` 및 `mysql.rds_history` 테이블에 있는 `mysql.rds_replication_status`의 시간, 사용자 및 작업입니다.

문제의 원인으로 알려진 특정 GTID 기반 트랜잭션을 건너뛰려면 mysql.rds_skip_transaction_with_gtid(Aurora MySQL 버전 2 및 3) 저장 프로시저를 사용할 수 있습니다. GTID 기반 복제 작업에 대한 자세한 내용은 GTID 기반 복제 사용 단원을 참조하세요.

예시

Aurora 기본 인스턴스에서 다음 예제를 실행하면 Aurora 클러스터가 Aurora 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본의 역할을 하도록 구성됩니다.

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position(Aurora MySQL 버전 2)

복제 모드를 바이너리 로그 파일 위치 또는 전역 트랜잭션 식별자(GTID)를 기반으로 설정합니다.

구문

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

파라미터

auto_position_mode

로그 파일 위치 복제 또는 GTID를 기반으로 하는 복제를 사용할지 여부를 나타내는 값:

- `0` – 바이너리 로그 파일 위치를 기반으로 한 복제 방법을 사용합니다. 기본값은 `0`입니다.
- 1 – GTID 기반 복제 방법을 사용합니다.

사용 노트

마스터 사용자는 `mysql.rds_set_master_auto_position` 프로시저를 실행해야 합니다.

이 프로시저는 Aurora MySQL 버전 2에서 지원됩니다.

mysql.rds_set_read_only(Aurora MySQL 버전 3)

DB 인스턴스에 대해 전역적으로 `read_only` 모드를 활성화하거나 비활성화합니다.

구문

```
CALL mysql.rds_set_read_only(mode);
```

파라미터

`mode`

DB 인스턴스에 대해 `read_only` 모드가 전역적으로 활성화되었는지 또는 비활성화되었는지를 나타내는 값입니다.

- `0` - `OFF`. 기본값은 `0`입니다.
- `1` – `ON`

사용 노트

`mysql.rds_set_read_only` 저장 프로시저는 `read_only` 파라미터만 수정합니다. 리더 DB 인스턴스에서는 `innodb_read_only` 파라미터를 변경할 수 없습니다.

`read_only` 파라미터 변경 사항은 재부팅 후에는 지속되지 않습니다. `read_only`를 영구적으로 변경하려면 `read_only` DB 클러스터 파라미터를 사용해야 합니다.

이 프로시저는 Aurora MySQL 버전 3.06 이상에서 지원됩니다.

mysql.rds_set_session_binlog_format(Aurora MySQL 버전 2)

현재 세션의 이진 로그 형식을 설정합니다.

구문

```
CALL mysql.rds_set_session_binlog_format(format);
```

파라미터

`format`

현재 세션의 이진 로그 형식을 나타내는 값:

- `STATEMENT` - 복제 소스가 SQL 명령문을 기반으로 이진 로그에 이벤트를 기록합니다.
- `ROW` - 복제 소스가 개별 테이블 행의 변경을 나타내는 이벤트를 이진 로그에 기록합니다.
- `MIXED` - 로깅이 일반적으로 SQL 명령문을 기반으로 하지만 특정 조건에서는 행으로 전환됩니다. 자세한 내용은 MySQL 설명서의 [혼합 이진 로깅 형식](#)을 참조하세요.

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

이 저장 프로시저를 사용하려면 현재 세션에 대해 이진 로깅을 구성해야 합니다.

Aurora의 경우, 이 프로시저는 Aurora MySQL 버전 2.12 이상과 MySQL 5.7 호환 버전에서 지원됩니다.

mysql.rds_set_source_auto_position(Aurora MySQL 버전 3)

복제 모드를 바이너리 로그 파일 위치 또는 전역 트랜잭션 식별자(GTID)를 기반으로 설정합니다.

구문

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

파라미터

auto_position_mode

로그 파일 위치 복제 또는 GTID를 기반으로 하는 복제를 사용할지 여부를 나타내는 값:

- `0` – 바이너리 로그 파일 위치를 기반으로 한 복제 방법을 사용합니다. 기본값은 `0`입니다.
- 1 – GTID 기반 복제 방법을 사용합니다.

사용 노트

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

관리 사용자는 `mysql.rds_set_source_auto_position` 프로시저를 실행해야 합니다.

mysql.rds_skip_repl_error

MySQL DB 읽기 전용 복제본의 복제 오류를 건너뛰고 삭제합니다.

구문

```
CALL mysql.rds_skip_repl_error;
```

사용 노트

마스터 사용자는 읽기 전용 복제본에서 `mysql.rds_skip_repl_error` 프로시저를 실행해야 합니다. 이 프로시저에 대한 자세한 내용은 현재 복제 오류 넘어가기를 참조하세요.

오류가 있는지 여부를 판별하려면 MySQL `SHOW REPLICA STATUS\G` 명령을 실행합니다. 중대한 복제 오류가 아닌 경우 `mysql.rds_skip_repl_error`를 사용하여 오류를 건너뛸 수 있습니다. 오류가 여러 개인 경우 `mysql.rds_skip_repl_error`는 첫 번째 오류를 삭제한 후 다른 오류가 있음을 경고합니다. `SHOW REPLICA STATUS\G`를 사용하여 다음 오류에 대한 적합한 조치를 결정할 수 있습니다. 반환된 값에 대한 자세한 내용은 MySQL 설명서의 SHOW SLAVE STATUS 문을 참조하세요.

Aurora MySQL의 복제 오류 해결에 대한 자세한 내용은 MySQL 읽기 복제 오류 진단 및 해결 섹션을 참조하세요.

복제 중지 오류

`mysql.rds_skip_repl_error` 프로시저를 호출할 때 복제본이 중단되었거나 사용 중지되었다는 오류 메시지가 표시될 수 있습니다.

읽기 전용 복제본 대신 기본 인스턴스에서 프로시저를 실행하면 이 오류 메시지가 나타납니다. 이 프로시저가 작동하려면 읽기 전용 복제본에서 이 프로시저를 실행해야 합니다.

이 오류 메시지는 읽기 전용 복제본에서 프로시저를 실행하지만 복제를 성공적으로 다시 시작할 수 없는 경우에도 나타날 수 있습니다.

많은 수의 오류를 건너뛰어야 하는 경우, 복제 지연이 바이너리 로그(binlog) 파일의 기본값 보관 기간 이상으로 늘어날 수 있습니다. 이 경우 binlog 파일이 읽기 전용 복제본에서 재생되기 전에 지워지기 때문에 치명적 오류가 발생할 수 있습니다. 이 제거는 복제를 중지시키며, 복제 오류를 건너뛰기 위해 더 이상 `mysql.rds_skip_repl_error` 명령을 호출할 수 없습니다.

이 문제는 소스 데이터베이스 인스턴스에서 binlog 파일이 보관되는 시간을 늘려서 완화할 수 있습니다. binlog 보관 시간을 늘린 후에 복제를 재시작하고 필요에 따라 `mysql.rds_skip_repl_error` 명령을 호출할 수 있습니다.

binlog 보관 기간을 설정하려면 mysql.rds_set_configuration 프로시저를 사용하여 `'binlog retention hours'` 구성 파라미터와 DB 클러스터에 binlog 파일을 보관할 시간을 함께 지정하십시오. 다음 예제에서는 binlog 파일의 보관 기간을 48시간으로 설정합니다.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Aurora MySQL DB 클러스터에서 복제를 시작합니다.

> **Note**
>
> mysql.rds_start_replication_until(Aurora MySQL 버전 3) 또는 mysql.rds_start_replication_until_gtid(Aurora MySQL 버전 3) 저장 프로시저를 사용하여 Aurora MySQL DB 인스턴스에서 복제를 시작하고 지정된 바이너리 로그 파일 위치에서 복제를 중지할 수 있습니다.

구문

```
CALL mysql.rds_start_replication;
```

사용 노트

마스터 사용자는 `mysql.rds_start_replication` 프로시저를 실행해야 합니다.

Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 데이터를 가져오려면 mysql.rds_set_external_master(Aurora MySQL 버전 2) 또는 mysql.rds_set_external_source(Aurora MySQL 버전 3)를 직접 호출하여 복제 구성을 빌드한 다음, 읽기 전용 복제본에서 `mysql.rds_start_replication`을 직접 호출하여 복제 프로세스를 시작합니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.

Amazon RDS 외부의 MySQL 인스턴스로 데이터를 내보내려면 읽기 전용 복제본에서 `mysql.rds_start_replication` 및 `mysql.rds_stop_replication`을 호출하여 바이너리 로그 삭제 등의 몇 가지 복제 작업을 제어합니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.

읽기 전용 복제본에서 mysql.rds_start_replication을 호출하여 이전에 mysql.rds_stop_replication을 호출하여 중지한 복제 프로세스를 재시작할 수도 있습니다. 자세한 내용은 [복제 중지 오류](#) 단원을 참조하십시오.

mysql.rds_start_replication_until(Aurora MySQL 버전 3)

Aurora MySQL DB 클러스터에서 복제를 시작하고 지정된 바이너리 로그 파일 위치에서 복제를 중지합니다.

구문

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

파라미터

replication_log_file

복제 정보를 포함하는 소스 데이터베이스 인스턴스의 이진 로그 이름입니다.

replication_stop_point

복제를 중지할 replication_log_file 바이너리 로그 내 위치입니다.

사용 노트

마스터 사용자는 mysql.rds_start_replication_until 프로시저를 실행해야 합니다.

이 프로시저는 Aurora MySQL 버전 3.04 이상에서 지원됩니다.

mysql.rds_start_replication_until 저장 프로시저는 다음을 포함하는 관리형 복제에 지원되지 않습니다.

- 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제
- Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

replication_log_file 파라미터에 지정된 파일 이름은 소스 데이터베이스 인스턴스 binlog 파일 이름과 일치해야 합니다.

replication_stop_point 파라미터가 과거에 해당하는 중지 위치를 지정하는 경우 복제가 즉시 중지됩니다.

예시

다음 예에서는 복제를 시작하고 120 바이너리 로그 파일의 mysql-bin-changelog.000777 위치에 도달할 때까지 변경 사항을 복제합니다.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

MySQL DB 인스턴스에서 복제를 중지합니다.

구문

```
CALL mysql.rds_stop_replication;
```

사용 노트

마스터 사용자는 mysql.rds_stop_replication 프로시저를 실행해야 합니다.

Amazon RDS 외부에서 실행 중인 MySQL 인스턴스에서 데이터를 가져오도록 복제를 구성하는 경우 가져오기가 완료된 후 읽기 전용 복제본에서 mysql.rds_stop_replication을 호출하여 복제 프로세스를 중지합니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.

Amazon RDS 외부의 MySQL 인스턴스로 데이터를 내보내도록 복제를 구성할 경우 읽기 전용 복제본에서 mysql.rds_start_replication 및 mysql.rds_stop_replication을 호출하여 이진 로그 삭제 등의 몇 가지 복제 작업을 제어합니다. 자세한 내용은 Aurora과 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터(이진 로그 복제본) 간의 복제 단원을 참조하십시오.

mysql.rds_stop_replication 저장 프로시저는 다음을 포함하는 관리형 복제에 지원되지 않습니다.

- 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제
- Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

세션 또는 쿼리 종료

다음 저장 프로시저는 세션 또는 쿼리를 종료합니다.

주제

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

MySQL Server와의 연결을 종료합니다.

구문

```
CALL mysql.rds_kill(processID);
```

파라미터

processID

종료할 연결 스레드의 ID입니다.

사용 노트

MySQL Server에 대한 각 연결은 별개의 스레드로 실행됩니다. 연결을 종료하려면 `mysql.rds_kill` 프로시저를 사용하여 해당 연결의 스레드 ID를 전달합니다. 스레드 ID를 확인하려면 MySQL SHOW PROCESSLIST 명령을 사용합니다.

예시

다음 예제는 스레드 ID가 4243인 연결을 종료합니다.

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

MySQL Server에서 실행 중인 쿼리를 종료합니다.

구문

```
CALL mysql.rds_kill_query(processID);
```

파라미터

processID

종료할 쿼리를 실행 중인 프로세스 또는 스레드의 ID입니다.

사용 노트

MySQL Server에서 실행 중인 쿼리를 중지하려면 `mysql_rds_kill_query` 프로시저를 사용하여 해당 쿼리를 실행 중인 스레드의 연결 ID를 전달합니다. 그러면 프로시저가 연결을 종료합니다.

ID를 확인하려면 MySQL [INFORMATION_SCHEMA PROCESSLIST 테이블](#)을 쿼리하거나 MySQL [SHOW PROCESSLIST](#) 명령을 사용합니다. `SHOW PROCESSLIST` 또는 `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST`에서 가져온 ID 열의 값은 *processID*입니다.

예시

다음 예시는 쿼리 스레드 ID가 230040인 쿼리를 중지합니다.

```
CALL mysql.rds_kill_query(230040);
```

GTID를 사용한 트랜잭션 복제

다음 저장 프로시저는 Aurora MySQL에서 전역 트랜잭션 식별자(GTID)를 사용하여 트랜잭션을 복제하는 방법을 제어합니다. Aurora MySQL에서 GTID 기반 복제를 사용하는 방법을 알아보려면 GTID 기반 복제 사용 섹션을 참조하세요.

주제

- mysql.rds_assign_gtids_to_anonymous_transactions(Aurora MySQL 버전 3)
- mysql.rds_gtid_purged(Aurora MySQL 버전 3)
- mysql.rds_skip_transaction_with_gtid(Aurora MySQL 버전 2 및 3)
- mysql.rds_start_replication_until_gtid(Aurora MySQL 버전 3)

mysql.rds_assign_gtids_to_anonymous_transactions(Aurora MySQL 버전 3)

`CHANGE REPLICATION SOURCE TO` 문의 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 옵션을 구성합니다. 이렇게 하면 복제 채널에서 GTID를 가지고 있지 않은 복제된 트랜잭션에 GTID를 할당합니다. 이렇게 하면 GTID 기반 복제를 사용하지 않는 소스에서 GTID 기반 복제를 사용하는 복제본으로 바이너리 로그 복제를 수행할 수 있습니다. 자세한 내용은 MySQL 참조 설명서의 복제 소스를 명령문으로 변경 및 GTID를 사용하지 않는 소스에서 GTID를 사용하는 복제본으로 복제를 참조하세요.

구문

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

파라미터

`gtid_option`

문자열 값입니다. 허용 값은 `OFF`, `LOCAL` 또는 지정된 UUID입니다.

사용 노트

이 프로시저에는 커뮤니티 MySQL에서 `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` 문을 실행하는 것과 동일한 효과가 있습니다.

GTID는 *`gtid_option`*에 대한 `ON`로 튜닝되어 `LOCAL` 또는 특정 UUID로 설정되어야 합니다.

기본값은 OFF이며, 기능이 사용되지 않음을 의미합니다.

LOCAL은 복제본의 자체 UUID(server_uuid 설정)를 포함하는 GTID를 할당합니다.

UUID인 파라미터를 전달하면 지정된 UUID(예: 복제 소스 서버에 대한 server_uuid 설정)를 포함하는 GTID를 할당합니다.

예시

이 기능을 해제하려면 다음과 같이 하세요.

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

복제본의 자체 UUID를 사용하려면 다음과 같이 하세요.

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+---------------------------------------------------------------+
| Message  |
+---------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+---------------------------------------------------------------+
1 row in set (0.07 sec)
```

지정된 UUID를 사용하려면 다음과 같이 하세요.

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------
+
```

```
1 row in set (0.07 sec)
```

mysql.rds_gtid_purged(Aurora MySQL 버전 3)

시스템 변수 `gtid_purged`의 글로벌 값을 지정된 글로벌 트랜잭션 식별자(GTID) 집합으로 설정합니다. `gtid_purged` 시스템 변수는 서버에서 커밋되었지만 서버의 이진 로그 파일에는 존재하지 않는 모든 트랜잭션의 GTID로 구성된 GTID 집합입니다.

MySQL 8.0과의 호환되도록 하기 위해 다음 두 가지 방법으로 `gtid_purged` 값을 설정할 수 있습니다.

- `gtid_purged`의 값을 지정된 GTID 집합으로 교체합니다.
- 지정된 GTID 집합을 `gtid_purged`에 이미 포함된 GTID 집합 세트에 덧붙입니다.

구문

`gtid_purged`의 값을 지정된 GTID 집합으로 교체하는 방법:

```
CALL mysql.rds_gtid_purged (gtid_set);
```

`gtid_purged`의 값을 지정된 GTID 집합에 덧붙이는 방법:

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

파라미터

`gtid_set`

*`gtid_set`*의 값은 현재 `gtid_purged` 값의 상위 집합이어야 하며, `gtid_subtract(gtid_executed,gtid_purged)`와 겹쳐서는 안 됩니다. 즉, 새 GTID 집합에는 `gtid_purged`에 이미 들어 있던 모든 GTID를 포함해야 하며, `gtid_executed`에서 아직 삭제되지 않은 GTID는 포함할 수 없습니다. 또한 *`gtid_set`* 파라미터는 글로벌 `gtid_owned` 집합에 있는 GTID, 즉 현재 서버에서 처리 중인 트랜잭션에 대한 GTID를 포함할 수 없습니다.

사용 노트

마스터 사용자는 `mysql.rds_gtid_purged` 프로시저를 실행해야 합니다.

이 프로시저는 Aurora MySQL 버전 3.04 이상에서 지원됩니다.

예시

다음 예에서는 GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`을 `gtid_purged` 글로벌 변수에 할당합니다.

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_transaction_with_gtid(Aurora MySQL 버전 2 및 3)

Aurora 기본 인스턴스에서 지정된 전역 트랜잭션 식별자(GTID)를 사용하여 트랜잭션 복제를 건너뜁니다.

특정 GTID 트랜잭션이 문제의 원인으로 알려진 경우 재해 복구를 위해 이 프로시저를 사용할 수 있습니다. 이 저장 프로시저를 사용하여 문제의 트랜잭션을 건너 뛰십시오. 문제의 트랜잭션의 예로는 복제를 비활성화하거나 중요한 데이터를 삭제하거나 DB 인스턴스를 사용할 수 없도록 하는 트랜잭션이 포함됩니다.

구문

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

파라미터

`gtid_to_skip`

건너 뛸 복제 트랜잭션의 GTID입니다.

사용 노트

마스터 사용자는 `mysql.rds_skip_transaction_with_gtid` 프로시저를 실행해야 합니다.

이 프로시저는 Aurora MySQL 버전 2 및 3에서 지원됩니다.

예시

다음 예에서는 GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`을 사용하여 트랜잭션의 복제를 건너뜁니다.

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid(Aurora MySQL 버전 3)

Aurora MySQL DB 클러스터에서 복제를 시작하고 지정된 글로벌 트랜잭션 식별자(GTID) 바로 다음에서 복제를 중지합니다.

구문

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

파라미터

gtid

GTID 이후 복제를 중지해야 합니다.

사용 노트

마스터 사용자는 `mysql.rds_start_replication_until_gtid` 프로시저를 실행해야 합니다.

이 프로시저는 Aurora MySQL 버전 3.04 이상에서 지원됩니다.

`mysql.rds_start_replication_until_gtid` 저장 프로시저는 다음을 포함하는 관리형 복제에 지원되지 않습니다.

- 여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제
- Aurora 읽기 전용 복제본을 사용하여 RDS for MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

`gtid` 파라미터가 복제본으로 이미 실행한 트랜잭션을 지정하는 경우 복제가 즉시 중지됩니다.

예시

다음 예제에서는 복제를 시작하고 GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`에 도달할 때까지 변경 사항을 복제합니다.

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

쿼리 로그 교체

다음 저장 프로시저는 MySQL 로그를 백업 테이블로 로테이션합니다. 자세한 내용은 Aurora MySQL 데이터베이스 로그 파일 단원을 참조하십시오.

주제

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

mysql.general_log 테이블을 백업 테이블로 로테이션합니다.

구문

```
CALL mysql.rds_rotate_general_log;
```

사용 노트

mysql.rds_rotate_general_log 프로시저를 호출하여 mysql.general_log 테이블을 백업 테이블로 로테이션할 수 있습니다. 로그 테이블이 순환되면 현재 로그 테이블은 백업 로그 테이블에 복사되며 현재 로그 테이블의 해당 항목들은 제거됩니다. 백업 로그 테이블이 이미 존재할 경우, 현재 로그 테이블이 백업으로 복사되기 전에 백업 로그 테이블이 삭제됩니다. 필요하다면 백업 로그 테이블을 쿼리할 수 있습니다. mysql.general_log 테이블에 대한 백업 로그 테이블 이름은 mysql.general_log_backup으로 지정됩니다.

이 절차는 log_output 파라미터를 TABLE로 설정한 경우에만 실행할 수 있습니다.

mysql.rds_rotate_slow_log

mysql.slow_log 테이블을 백업 테이블로 로테이션합니다.

구문

```
CALL mysql.rds_rotate_slow_log;
```

사용 노트

mysql.rds_rotate_slow_log 프로시저를 호출하여 mysql.slow_log 테이블을 백업 테이블로 로테이션할 수 있습니다. 로그 테이블이 순환되면 현재 로그 테이블은 백업 로그 테이블에 복사되며 현

재 로그 테이블의 해당 항목들은 제거됩니다. 백업 로그 테이블이 이미 존재할 경우, 현재 로그 테이블이 백업으로 복사되기 전에 백업 로그 테이블이 삭제됩니다.

필요하다면 백업 로그 테이블을 쿼리할 수 있습니다. `mysql.slow_log` 테이블에 대한 백업 로그 테이블 이름은 `mysql.slow_log_backup`으로 지정됩니다.

이진수 로그 구성 설정 및 표시

다음 저장 프로시저는 바이너리 로그 파일 유지에 대한 파라미터와 같은 구성 파라미터를 설정하고 표시합니다.

주제

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

바이너리 로그를 보관할 기간(시간) 또는 복제를 지연할 시간(초)을 지정합니다.

구문

```
CALL mysql.rds_set_configuration(name,value);
```

파라미터

##

설정할 구성 파라미터의 이름입니다.

USD ##

구성 파라미터의 값입니다.

사용 노트

`mysql.rds_set_configuration` 프로시저는 다음 구성 파라미터를 지원합니다.

- binlog retention hours

구성 파라미터는 영구적으로 저장되며 DB 인스턴스 재부팅 또는 장애 조치 이후에도 유지됩니다.

binlog retention hours

`binlog retention hours` 파라미터는 이진 로그 파일을 보관할 기간(시간)을 지정하는 데 사용됩니다. Amazon Aurora는 일반적으로 가능한 한 빨리 바이너리 로그를 삭제하지만 Aurora 외부의 MySQL 데이터베이스 복제에는 바이너리 로그가 필요할 수 있습니다.

`binlog retention hours`의 기본값은 NULL입니다. Aurora MySQL에 대해 NULL은 이진 로그가 느리게 정리됨을 의미합니다. Aurora MySQL 바이너리 로그는 특정 기간(보통 하루 이하) 시스템에 남아 있을 수 있습니다.

DB 클러스터에 대한 바이너리 로그를 유지할 기간(시간)을 지정하려면 다음 예에 나와 있는 것처럼 `mysql.rds_set_configuration` 저장 프로시저를 사용하여 복제 수행에 충분한 기간을 지정합니다.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> `binlog retention hours`에는 `0` 값을 사용할 수 없습니다.

Aurora MySQL 버전 2.11.0 이상 및 버전 3 DB 클러스터의 최대 `binlog retention hours` 값은 2160(90일)입니다.

보존 기간을 설정한 후, DB 인스턴스 스토리지의 사용량을 모니터링하여 보존된 바이너리 로그가 너무 많은 스토리지를 차지하지 않도록 합니다.

mysql.rds_show_configuration

바이너리 로그가 유지되는 시간입니다.

구문

```
CALL mysql.rds_show_configuration;
```

사용 노트

Amazon RDS의 이진수 로그 보관 시간을 확인하려면 `mysql.rds_show_configuration` 저장 프로시저를 사용합니다.

예시

다음 예제는 보존 기간을 표시합니다.

```
call mysql.rds_show_configuration;
```

```
                 name                           value      description
                 binlog retention hours         24         binlog retention hours specifies
  the duration in hours before binary logs are automatically deleted.
```

Aurora MySQL 전용 information_schema 테이블

Aurora MySQL에는 Aurora 전용 특정 `information_schema` 테이블이 있습니다.

information_schema.aurora_global_db_instance_status

`information_schema.aurora_global_db_instance_status` 테이블에는 글로벌 데이터베이스의 기본 및 보조 DB 클러스터에 있는 모든 DB 인스턴스의 상태에 대한 정보가 들어 있습니다. 사용할 수 있는 열이 다음 테이블에 나와 있습니다. 나머지 열은 Aurora 내부용으로만 사용됩니다.

> **Note**
>
> 이 정보 스키마 테이블은 Aurora MySQL 버전 3.04.0 이상의 글로벌 데이터베이스에서만 사용할 수 있습니다.

열	데이터 유형	설명
SERVER_ID	varchar(100)	DB 인스턴스의 식별자입니다.
SESSION_ID	varchar(100)	현재 세션에 대한 고유 식별자입니다. `MASTER_SESSION_ID` 의 값은 Writer(프라이머리) DB 인스턴스를 식별합니다.
AWS_REGION	varchar(100)	이 글로벌 데이터베이스 인스턴스가 실행되는 AWS 리전입니다. 리전 목록은 [리전 가용성] 섹션을 참조하세요.
DURABLE_LSN	bigint 서명 안 됨	스토리지에서 뛰어난 내구성을 갖게 된 로그 시퀀스 번호(LSN)입니다. LSN(로그 시퀀

열	데이터 유형	설명
		스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
HIGHEST_LSN_RCVD	bigint 서명 안 됨	라이터 DB 인스턴스로부터 DB 인스턴스가 수신한 가장 높은 LSN입니다.
OLDEST_READ_VIEW_TRX_ID	bigint 서명 안 됨	라이터 DB 인스턴스가 삭제할 수 있는 가장 오래된 트랜잭션의 ID입니다.
OLDEST_READ_VIEW_LSN	bigint 서명 안 됨	DB 인스턴스가 스토리지에서 읽는 데 사용하는 가장 오래된 LSN입니다.
VISIBILITY_LAG_IN_MSEC	float(10,0) 서명 안 됨	기본 DB 클러스터의 리더의 경우 이 DB 인스턴스가 라이터 DB 인스턴스보다 얼마나 뒤처지는지 밀리초 단위로 나타낸 것입니다. 보조 DB 클러스터의 리더의 경우 이 DB 인스턴스가 보조 볼륨보다 얼마나 뒤처지는지 밀리초 단위로 나타낸 것입니다.

information_schema.aurora_global_db_status

`information_schema.aurora_global_db_status` 테이블에는 Aurora Global Database 지연의 다양한 측면, 특히 기본 Aurora 스토리지의 지연(내구성 지연) 및 Recovery Point Objective(RPO) 간의 지연에 대한 정보가 표시됩니다. 사용할 수 있는 열이 다음 테이블에 나와 있습니다. 나머지 열은 Aurora 내부용으로만 사용됩니다.

> **Note**
>
> 이 정보 스키마 테이블은 Aurora MySQL 버전 3.04.0 이상의 글로벌 데이터베이스에서만 사용할 수 있습니다.

열	데이터 유형	설명
AWS_REGION	varchar(100)	이 글로벌 데이터베이스 인스턴스가 실행되는 AWS 리전입니다. 리전 목록은 [리전 가용성](#) 섹션을 참조하세요.
HIGHEST_LSN_WRITTEN	bigint 서명 안 됨	이 DB 클러스터에 현재 존재하는 가장 높은 로그 시퀀스 번호(LSN)입니다. LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
DURABILITY_LAG_IN_MILLISECONDS	float(10,0) 서명 안 됨	보조 DB 클러스터의 `HIGHEST_LSN_WRITTEN` 과 기본 DB 클러스터의 `HIGHEST_LSN_WRITTEN` 간 타임스탬프 값 차이입니다. 이 값은 Aurora 글로벌 데이터베이스의 기본 DB 클러스터에서 항상 0입니다.
RPO_LAG_IN_MILLISECONDS	float(10,0) 서명 안 됨	Recovery Point Objective (RPO) 지연입니다. RPO 지연은 Aurora Global Databse의 기본 DB 클러스터에 저장된 후 가장 최근의 사용자 트랜잭션

열	데이터 유형	설명
		COMMIT 보조 DB 클러스터에 저장하는 데 걸리는 시간입니다. 이 값은 Aurora 글로벌 데이터베이스의 기본 DB 클러스터에서 항상 0입니다. 간단히 말해서 이 지표는 Aurora 글로벌 데이터베이스의 각 Aurora MySQL DB 클러스터에 대한 복구 시점 목표, 즉 중단 시 손실될 수 있는 데이터의 양을 계산합니다. 지연과 마찬가지로 RPO는 시간 단위로 측정됩니다.
LAST_LAG_CALCULATION_TIMESTAMP	datetime	`DURABILITY_LAG_IN_MILLISECONDS` 및 `RPO_LAG_IN_MILLISECONDS` 에 대한 값이 마지막으로 계산된 시점을 나타내는 타임스탬프입니다. 시간 값(예: `1970-01-01 00:00:00+00`)은 이것이 기본 DB 클러스터임을 의미합니다.
OLDEST_READ_VIEW_TRX_ID	bigint 서명 안 됨	라이터 DB 인스턴스가 삭제할 수 있는 가장 오래된 트랜잭션의 ID입니다.

information_schema.replica_host_status

`information_schema.replica_host_status` 테이블에는 복제 정보가 들어 있습니다. 사용할 수 있는 열이 다음 테이블에 나와 있습니다. 나머지 열은 Aurora 내부용으로만 사용됩니다.

열	데이터 유형	설명
CPU	double	복제본 호스트의 CPU 사용률(%)입니다.
IS_CURRENT	tinyint	복제본이 최신 상태인지를 나타냅니다.
LAST_UPDATE_TIMESTAMP	datetime(6)	마지막 업데이트가 발생한 시간입니다. 레코드가 오래되었는지를 확인하는 데 사용됩니다.
REPLICA_LAG_IN_MILLISECONDS	double	밀리 초로 나타낸 복제본 지연 시간입니다.
SERVER_ID	varchar(100)	데이터베이스 서버의 ID입니다.
SESSION_ID	varchar(100)	데이터베이스 세션의 ID입니다. DB 인스턴스가 라이터 인스턴스인지 아니면 리더 인스턴스인지를 결정하는 데 사용됩니다.

> **Note**
>
> 복제본 인스턴스가 지연되면 해당 `information_schema.replica_host_status` 테이블에서 쿼리된 정보가 만료될 수 있습니다. 이러한 경우 라이터 인스턴스에서 쿼리하는 것이 좋습니다.
>
> `mysql.ro_replica_status` 테이블에 비슷한 정보가 있지만 사용하지 않는 것이 좋습니다.

information_schema.aurora_forwarding_processlist

`information_schema.aurora_forwarding_processlist` 테이블에는 쓰기 전달과 관련된 프로세스에 대한 정보가 들어 있습니다.

이 테이블의 내용은 글로벌 또는 클러스터 내 쓰기 전달이 켜진 DB 클러스터의 라이터 DB 인스턴스에서만 볼 수 있습니다. 리더 DB 인스턴스에는 빈 결과 세트가 반환됩니다.

필드	데이터 유형	설명
ID	bigint	라이터 DB 인스턴스의 연결 식별자입니다. 이 식별자는 `SHOW PROCESSLIST` 명령문의 `Id` 열에 표시된 값 및 스레드 내에서 `CONNECTION_ID()` 함수가 반환하는 값과 동일한 값입니다.
USER	varchar(32)	명령문을 발행한 MySQL 사용자입니다.
HOST	varchar(255)	명령문을 발행한 MySQL 클라이언트입니다. 전달된 명령문의 경우 이 필드에는 전달하는 리더 DB 인스턴스에 연결을 설정한 애플리케이션 클라이언트 호스트 주소가 표시됩니다.
DB	varchar(64)	스레드의 기본 데이터베이스입니다.
명령	varchar(16)	스레드가 클라이언트를 대신하여 실행하는 명령 유형 또는 세션이 유휴 상태인 경우 `Sleep`입니다. 스레드 명령에 대한 설명은 MySQL 설명서에서 스레드 명령 값을 참조하세요.
TIME	int	스레드가 현재 상태를 유지한 시간(초)입니다.
STATE	varchar(64)	스레드가 수행하는 일을 나타내는 작업, 이벤트 또는 상태입니다. 상태 값에 대한 설명은 MySQL 설명서에서 일반 스레드 상태를 참조하세요.
INFO	longtext	스레드가 실행 중이라는 명령문 또는 명령문을 실행하지 않는 경우 `NULL`입니다. 명령문은 서버로 전송된 명령문일 수도 있고 해당 명령문이 다른 명령문을 실행하는 경우 가장 안쪽에 있는 명령문일 수도 있습니다.
IS_FORWARDED	bigint	스레드가 리더 DB 인스턴스에서 전달되었는지를 나타냅니다.

필드	데이터 유형	설명
REPLICA_SESSION_ID	bigint	Aurora 복제본의 연결 식별자입니다. 이 식별자는 전달하는 Aurora 리더 DB 인스턴스에서 `SHOW PROCESSLIST` 명령문의 `Id` 열에 표시된 값과 동일한 값입니다.
REPLICA_INSTANCE_IDENTIFIER	varchar(64)	전달하는 스레드의 DB 인스턴스 식별자입니다.
REPLICA_CLUSTER_NAME	varchar(64)	전달하는 스레드의 DB 클러스터 식별자입니다. 클러스터 내 쓰기 전달의 경우 이 식별자는 라이터 DB 인스턴스와 동일한 DB 클러스터입니다.
REPLICA_REGION	varchar(64)	전달하는 스레드의 출처가 되는 AWS 리전입니다. 클러스터 내 쓰기 전달의 경우 이 리전은 라이터 DB 인스턴스와 동일한 AWS 리전입니다.

Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트

Amazon Aurora는 정기적으로 업데이트를 릴리스합니다. 업데이트는 시스템 유지 관리 기간 중에 Aurora DB 클러스터에 적용됩니다. 업데이트가 적용되는 시기는 업데이트의 유형뿐 아니라 DB 클러스터에 대한 유지 관리 기간 설정 및 리전에 따라 다릅니다.

Amazon Aurora 릴리스는 며칠 이내에 모든 AWS 리전에 공개됩니다. 일부 리전에서는 아직 다른 리전에서 사용할 수 없는 엔진 버전이 일시적으로 표시될 수 있습니다.

업데이트는 DB 클러스터의 모든 인스턴스에 동시에 적용됩니다. 업데이트하려면 DB 클러스터의 모든 인스턴스에서 데이터베이스를 다시 시작해야 하므로 다운타임이 20-30초간 발생할 수 있습니다. 다운타임 후에 DB 클러스터 또는 클러스터 사용을 재개할 수 있습니다. 에서 유지 관리 기간 설정을 확인하거나 변경할 수 있습니다..AWS Management Console

Amazon Aurora에서 지원하는 Aurora MySQL 버전에 대한 자세한 내용은 Aurora MySQL 릴리스 정보를 참조하세요.

다음 단원에서는 클러스터에 적합한 Aurora MySQL 버전을 선택하는 방법, 클러스터를 만들거나 업그레이드할 때 버전을 지정하는 방법, 중단을 최소화하면서 한 버전에서 다른 버전으로 클러스터를 업그레이드하는 절차를 학습할 수 있습니다.

주제

Aurora MySQL 버전 번호 확인

Aurora MySQL 호환 버전는 MySQL과 호환되는 데이터베이스 엔진이지만 Aurora MySQL에는 특정 Aurora MySQL 버전에만 해당하는 기능 및 버그 수정 사항이 포함되어 있습니다. 애플리케이션 개발자는 SQL을 사용하여 애플리케이션에서 Aurora MySQL 버전을 확인할 수 있습니다. 데이터베이스 관리자는 Aurora MySQL DB 클러스터 및 DB 인스턴스를 생성하거나 업그레이드할 때 Aurora MySQL 버전을 확인하고 지정할 수 있습니다.

주제

- [AWS를 통해 Aurora MySQL 엔진 버전 확인 또는 지정]
- [SQL을 사용하여 Aurora MySQL 버전 확인]

AWS를 통해 Aurora MySQL 엔진 버전 확인 또는 지정

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 관리 태스크를 수행하는 경우 Aurora MySQL 버전을 설명이 포함된 영숫자 형식으로 지정합니다.

Aurora MySQL 버전 2부터는 Aurora 엔진 버전에 다음 구문이 제공됩니다.

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

`mysql-major-version-` 부분은 `5.7` 또는 `8.0`입니다. 이 값은 클라이언트 프로토콜 버전과 해당 Aurora MySQL 버전에 대한 전반적인 MySQL 기능 지원 수준을 나타냅니다.

`aurora-mysql-version` 은 점으로 구분된 세 부분(Aurora MySQL 주 버전, Aurora MySQL 부 버전, 패치 수준)으로 이루어진 값입니다. 메이저 버전은 2 또는 3입니다. 이러한 값은 각각 MySQL 5.7 또는 8.0과 호환되는 Aurora MySQL을 나타냅니다. 마이너 버전은 2.x 또는 3.x 시리즈 내 기능 릴리스를 나타냅니다. 패치 수준은 각 마이너 버전에 대해 `0`에서 시작하며 마이너 버전에 적용되는 후속 버그 수정 모음을 나타냅니다. 경우에 따라 새로운 기능이 마이너 버전에 통합되지만 즉시 공개되지 않습니다. 이러한 경우 기능이 미세 조정되고 이후 패치 수준에서 공개됩니다.

모든 2.x Aurora MySQL 엔진 버전은 Community MySQL 5.7.12 이상과 유선 호환됩니다. 모든 3.x Aurora MySQL 엔진 버전은 MySQL 8.0.23 이상과 유선 호환됩니다. 특정 3.x 버전의 릴리스 정보를 참조하여 해당 MySQL 호환 버전을 찾아볼 수 있습니다.

예를 들어, Aurora MySQL 3.04.0 및 2.11.2의 엔진 버전은 다음과 같습니다.

```
8.0.mysql_aurora.3.04.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> 커뮤니티 MySQL 버전과 Aurora MySQL 2.x 버전 간에는 일대일 대응이 없습니다. Aurora MySQL 버전 3의 경우 더 직접적인 매핑이 있습니다. 특정 Aurora MySQL 릴리스에 포함된 버그 수정 사항 및 새로운 기능을 확인하려면 Aurora MySQL 릴리스 정보의 [Amazon Aurora MySQL 버전 3에 대한 데이터베이스 엔진 업데이트] 및 [Amazon Aurora MySQL 버전 2에 대한 데이터베이스 엔진 업데이트]를 참조하세요. 새로운 기능 및 릴리스의 시간 순 목록은 [문서 기

록 단원을 참조하세요. 보안 관련 수정 사항에 필요한 최소 버전을 확인하려면 Aurora MySQL 릴리스 정보의 Aurora MySQL에서 수정된 보안 취약성을 참조하세요.

일부 AWS CLI 명령 및 RDS API 작업에서 Aurora MySQL 엔진 버전을 지정할 수 있습니다. 예를 들면 `--engine-version` 명령 create-db-cluster 및 modify-db-cluster 실행 시 AWS CLI 옵션을 지정할 수 있습니다. RDS API 작업 CreateDBCluster 및 ModifyDBCluster 실행 시 `EngineVersion` 파라미터를 지정할 수 있습니다.

Aurora MySQL 버전 2 이상에서는 AWS Management Console의 엔진 버전에 Aurora 버전도 포함됩니다. 클러스터를 업그레이드하면 표시된 값이 변경됩니다. 이 변경 사항을 통해 클러스터에 연결하거나 SQL 명령을 실행할 필요 없이 정확한 Aurora MySQL 버전을 지정하고 확인할 수 있습니다.

> **Tip**
>
> AWS CloudFormation을 통해 관리되는 Aurora 클러스터의 경우, `EngineVersion` 설정의 이 변경 사항은 AWS CloudFormation에 의해 작업을 트리거할 수 있습니다. AWS CloudFormation에서 `EngineVersion` 설정에 대한 변경을 처리하는 방법에 대한 자세한 내용은 AWS CloudFormation 설명서를 참조하세요.

SQL을 사용하여 Aurora MySQL 버전 확인

SQL 쿼리를 사용하여 애플리케이션에서 검색할 수 있는 Aurora 버전 번호는 *<major version>*.*<minor version>*.*<patch version>* 형식을 사용합니다. `AURORA_VERSION` 시스템 변수를 쿼리하여 Aurora MySQL 클러스터의 모든 DB 인스턴스에 대해 이 버전 번호를 확인할 수 있습니다. 이 버전 번호를 확인하려면 다음 쿼리 중 하나를 사용합니다.

```
select aurora_version();
select @@aurora_version;
```

이러한 쿼리는 다음과 유사한 출력을 생성합니다.

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

AWS를 통해 Aurora MySQL 엔진 버전 확인 또는 지정에 설명된 기법을 사용하여 콘솔, CLI 및 RDS API에서 반환되는 버전 번호는 일반적으로 좀더 설명적입니다.

Amazon Aurora MySQL 장기 지원(LTS) 및 베타 릴리스

Aurora MySQL은 일부 Aurora MySQL 엔진 버전에 대해 장기 지원(LTS) 및 베타 릴리스를 제공합니다.

주제

Aurora MySQL LTS(장기 지원) 릴리스

DB 클러스터를 생성하거나 업그레이드할 때 각 새 Aurora MySQL 버전을 일정 시간 동안 사용할 수 있습니다. 이 기간이 지나면 해당 버전을 사용하는 모든 클러스터를 업그레이드해야 합니다. 지원 기간이 끝나기 전에 클러스터를 수동으로 업그레이드하거나 Aurora MySQL 버전이 더 이상 지원되지 않는 경우 Aurora에서 자동으로 클러스터를 업그레이드할 수 있습니다.

Aurora는 특정 Aurora MySQL 버전을 "LTS(장기 지원)" 릴리스로 지정합니다. LTS 릴리스를 사용하는 DB 클러스터는 LTS가 아닌 릴리스를 사용하는 클러스터보다 동일한 버전을 더 오래 유지하고 더 적은 업그레이드 주기를 거칠 수 있습니다. Aurora는 해당 릴리스가 출시된 후 최소 3년 동안 각 LTS 릴리스를 지원합니다. LTS 릴리스에 있는 DB 클러스터를 업그레이드해야 할 경우 Aurora는 이 클러스터를 다음 LTS 릴리스로 업그레이드합니다. 이렇게 하면 클러스터를 오랫동안 다시 업그레이드할 필요가 없습니다.

Aurora MySQL LTS 릴리스의 수명 동안 새로운 패치 수준을 통해 중요한 문제에 대한 수정 사항을 적용합니다. 패치 수준에는 새로운 기능이 포함되지 않습니다. 이러한 패치를 LTS 릴리스를 실행하는 DB 클러스터에 적용할지 여부를 선택할 수 있습니다. 일부 중요 수정 사항의 경우 Amazon은 동일한 LTS 릴리스 내에서 특정 패치 수준으로 관리형 업그레이드를 수행할 수 있습니다. 이러한 관리형 업그레이드는 클러스터 유지 관리 기간 내에 자동으로 수행됩니다.

대부분의 Aurora MySQL 클러스터에서 LTS 릴리스를 사용하는 대신 최신 릴리스로 업그레이드하는 것이 좋습니다. 이렇게 하면 Aurora를 관리형 서비스로 활용하고 최신 기능 및 버그 수정에 액세스할 수 있습니다. LTS 릴리스는 다음과 같은 특성을 가진 클러스터 전용입니다.

- 중요 패치의 경우 드문 경우를 제외하고는 업그레이드에 대한 Aurora MySQL 애플리케이션의 가동 중지를 감당할 수 없는 경우

- Aurora MySQL 데이터베이스 엔진을 업데이트할 때마다 클러스터 및 관련 애플리케이션의 테스트 주기 시간이 오래 걸리는 경우
- Aurora MySQL 클러스터의 데이터베이스 버전에 애플리케이션에 필요한 모든 DB 엔진 기능과 버그 수정이 있는 경우

Aurora MySQL에 대한 현재 LTS 릴리스는 다음과 같습니다.

- Aurora MySQL 버전 3.04.*. LTS 버전에 대한 자세한 내용은 Aurora MySQL 릴리스 정보의 Amazon Aurora MySQL 버전 3 데이터베이스 엔진 업데이트를 참조하세요.

> **Note**
> LTS 버전의 경우 자동 마이너 버전 업그레이드를 비활성화하는 것이 좋습니다. `AutoMinorVersionUpgrade` 파라미터를 `false`로 설정하거나 AWS Management Console에서 자동 마이너 버전 업그레이드 활성화 확인란의 선택을 취소합니다.
> 비활성화하지 않으면 DB 클러스터가 3.05.2와 같은 비LTS 버전으로 업그레이드될 수 있습니다.

Aurora MySQL 베타 릴리스

Aurora MySQL 베타 릴리스는 초기 보안 수정만 적용된 릴리스로, 제한된 AWS 리전에만 해당됩니다. 다음 패치 릴리스에서는 이러한 수정 사항이 모든 리전에 더 광범위하게 배포될 예정입니다.

베타 릴리스의 버전 번호는 Aurora MySQL 마이너 버전과 비슷하지만 추가로 네 번째 숫자가 있습니다 (예: 2.12.0.1 또는 3.05.0.1).

자세한 내용은 Aurora MySQL 릴리스 정보에서 Amazon Aurora MySQL 버전 2 데이터베이스 엔진 업데이트, Amazon Aurora MySQL 버전 3 데이터베이스 엔진 업데이트를 참조하세요.

Amazon Aurora MySQL 호환 버전 2 표준 지원 종료 준비

Amazon Aurora MySQL 호환 버전 2(MySQL 5.7 호환성 지원)는 2024년 10월 31일에 표준 지원이 종료될 계획입니다. Aurora MySQL 버전 2의 표준 지원 기간이 종료되기 전에 Aurora MySQL 버전 2를 실행하는 모든 클러스터를 기본 Aurora MySQL 버전 3(MySQL 8.0 호환성 지원) 이상으로 업그레이드하는 것을 권장합니다. 2024년 10월 31일에 Amazon RDS는 데이터베이스를 Amazon RDS 추가 지원에 자동으로 등록합니다. Aurora Serverless 버전 1 클러스터에서 Aurora MySQL 버전 2(MySQL 5.7 호환성 지원)를 실행하는 경우에는 확장 지원이 해당하지 않습니다. Aurora Serverless 버전 1 클러스

터를 Aurora MySQL 버전 3으로 업그레이드하려면 Aurora Serverless v1 DB 클러스터의 업그레이드 경로 섹션을 참조하세요.

Aurora MySQL 메이저 버전용 릴리스 일정에서 Aurora MySQL 메이저 버전에 대한 향후 지원 종료일을 확인할 수 있습니다.

Aurora MySQL 버전 2를 실행하는 클러스터가 있는 경우, 표준 지원 종료 날짜가 가까워지면 업그레이드를 수행하는 방법에 대한 최신 정보가 포함된 정기 알림을 받게 됩니다. 이 페이지는 최신 정보로 정기적으로 업데이트됩니다.

표준 지원 종료 일정

1. 지금부터 2024년 10월 31일까지 - Aurora MySQL 버전 2(MySQL 5.7 호환성 지원) 클러스터를 Aurora MySQL 버전 3(MySQL 8.0 호환성 지원)로 업그레이드할 수 있습니다.
2. 2024년 10월 31일 - 이 날짜에 Aurora MySQL 버전 2의 표준 지원이 종료됩니다. Amazon RDS는 자동으로 클러스터를 Amazon RDS 확장 지원에 등록합니다.

RDS 확장 지원에 자동으로 등록해 드립니다. 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 단원을 참조하십시오.

이 지원 종료 프로세스의 영향을 받는 클러스터 찾기

이 지원 종료 프로세스의 영향을 받는 클러스터를 찾으려면 다음 절차를 따르세요.

> **Important**
>
> 리소스가 있는 모든 AWS 리전과 각 AWS 계정에서 이 지침을 수행해야 합니다.

콘솔

Aurora MySQL 버전 2 클러스터를 찾는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 데이터베이스별 필터링 상자에서 5.7을 입력합니다.
4. 엔진 열에서 Aurora MySQL 확인합니다.

AWS CLI

AWS CLI를 사용하여 이 지원 종료 프로세스의 영향을 받는 클러스터를 찾으려면 describe-db-clusters 명령을 호출합니다. 다음 샘플 스크립트를 사용할 수 있습니다.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1

        +----------------------------------------------------------------+
        |                        DescribeDBClusters                      |
        +---------------------+---------------+--------------------------+
        |         DBCI        |      EM       |            EV            |
        +---------------------+---------------+--------------------------+
        |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
        | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
        +---------------------+---------------+--------------------------+
```

RDS API

Aurora MySQL 버전 2를 실행하는 Aurora MySQL DB 클러스터를 찾으려면 다음 필수 파라미터를 사용하여 RDS DescribeDBClusters API 작업을 사용합니다.

- `DescribeDBClusters`
  - Filters.Filter.N
    - 명칭
      - engine
    - Values.Value.N
      - ['aurora']

Amazon RDS 추가 지원

지원 종료일인 2024년 10월 31일까지 커뮤니티 MySQL 5.7에서 Amazon RDS 확장 지원을 무료로 사용할 수 있습니다. 2024년 10월 31일에 Amazon RDS는 Aurora MySQL 버전 2에 대한 RDS 추가 지원에 데이터베이스를 자동으로 등록합니다. Aurora에 대한 RDS 확장 지원은 2027년 2월 RDS 확장 지원

이 종료될 때까지 Aurora MySQL 버전 2에 대해 최대 28개월의 확장 지원을 제공하는 유료 서비스입니다. RDS 추가 지원은 Aurora MySQL 마이너 버전 2.11 및 2.12에만 제공됩니다. 표준 지원이 종료된 이후에 Amazon Aurora MySQL 버전 2를 사용하려면 2024년 10월 31일 이전에 이러한 마이너 버전 중 하나에서 데이터베이스를 실행하도록 계획하시기 바랍니다.

요금 및 기타 고려 사항 등의 RDS 추가 지원에 대한 자세한 내용은 Amazon Aurora를 사용한 Amazon RDS 추가 지원 섹션을 참조하세요.

업그레이드 수행

주 버전 간에 업그레이드하려면 부 버전보다 더 광범위한 계획 및 테스트가 필요합니다. 이 과정은 상당한 시간이 걸릴 수 있습니다. 업그레이드 전, 업그레이드, 업그레이드 후의 활동을 포함한 3단계 프로세스로 업그레이드를 살펴보고자 합니다.

업그레이드 전:

업그레이드 전에는 업그레이드된 클러스터에 대한 애플리케이션 호환성, 성능, 유지 관리 절차 및 이와 유사한 고려 사항을 확인하여 업그레이드 후에 애플리케이션이 예상대로 작동하도록 하는 것이 좋습니다. 다음은 더 나은 업그레이드 경험을 제공하는 데 도움이 되는 다섯 가지 권장 사항입니다.

- 우선 Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식을 이해해야 합니다.
- 다음으로 Aurora MySQL 버전 2에서 버전 3으로 업그레이드 시 사용할 수 있는 업그레이드 기법을 살펴봅니다.
- 업그레이드할 적절한 시기와 접근 방식을 결정하는 데 도움이 되도록 Aurora MySQL 버전 2와 Aurora MySQL 버전 3의 비교을 통해 Aurora MySQL 버전 3와 현재 환경의 차이점을 알아볼 수 있습니다.
- 편리하고 가장 효과적인 옵션을 결정한 후 Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획를 사용하여 복제된 클러스터에서 모의로 현재 위치 업그레이드를 시도해 보세요.
- Aurora MySQL에 대한 메이저 버전 업그레이드 사전 확인을(를) 검토합니다. 업그레이드 사전 확인기를 실행하여 데이터베이스를 성공적으로 업그레이드할 수 있는지, 업그레이드 후 애플리케이션 비호환성 문제나 성능, 유지 관리 절차 및 유사한 고려 사항이 있는지 확인합니다.
- 모든 종류 또는 버전의 Aurora MySQL 클러스터가 현재 위치 업그레이드 메커니즘을 사용할 수 있는 것은 아닙니다. 자세한 내용은 Aurora MySQL 주 버전 업그레이드 경로 단원을 참조하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 Premium Support를 통해 AWS Support 팀에 도움을 요청할 수 있습니다.

업그레이드 수행:

다음 업그레이드 방법을 사용할 수 있습니다. 시스템에서 발생하는 가동 중지 정도는 선택한 업그레이드 방법에 따라 다릅니다.

- 블루/그린 배포 - 애플리케이션 가동 중지 시간을 줄이는 것이 최우선 순위인 상황에서는 프로비저닝된 Amazon Aurora DB 클러스터에서 메이저 버전 업그레이드를 수행하기 위해 Amazon RDS 블루/그린 배포를 사용할 수 있습니다. 블루/그린 배포는 프로덕션 환경을 복사하는 스테이징 환경을 만듭니다. 프로덕션 워크로드에 영향을 주지 않고 그린(스테이징) 환경에서 Aurora DB 클러스터의 특정 부분을 변경할 수 있습니다. 전환은 일반적으로 1분도 걸리지 않으며 데이터 손실이 발생하지 않습니다. 자세한 내용은 Aurora용 Amazon RDS 블루/그린 배포 개요 단원을 참조하십시오. 이렇게 하면 가동 중지가 최소화되지만 업그레이드를 수행하는 동안 추가 리소스를 실행해야 합니다.
- 인플레이스 업그레이드 - Aurora가 자동으로 사전 확인 프로세스를 수행하고, 클러스터를 오프라인으로 전환하고, 클러스터를 백업하고, 업그레이드를 수행하고, 클러스터를 다시 온라인으로 전환하는 인플레이스 업그레이드를 수행할 수 있습니다. 인플레이스 메이저 버전 업그레이드는 클릭 몇 번으로 수행할 수 있으며 다른 클러스터와의 기타 조정이나 장애 조치는 필요하지 않지만 가동 중지가 수반됩니다. 자세한 내용은 현재 위치 업그레이드 수행 방법 단원을 참조하세요.
- 스냅샷 및 복원 - Aurora MySQL 버전 2 스냅샷에서 Aurora MySQL 버전 3 클러스터로 복원하여 Aurora MySQL 버전 2 클러스터를 업그레이드할 수 있습니다. 이렇게 하려면 스냅샷을 만들고 스냅샷에서 복원하는 프로세스를 따라야 합니다. 이 프로세스는 스냅샷에서 복원하기 때문에 데이터베이스 중단을 수반합니다.

업그레이드 후:

업그레이드 후 시스템(애플리케이션 및 데이터베이스)을 면밀히 모니터링하고 필요한 경우 미세 조정하여 변경해야 합니다. 업그레이드 전 단계를 착실히 따르면 필요한 변경 사항을 최소화할 수 있습니다. 자세한 내용은 Amazon Aurora MySQL 데이터베이스 성능 문제 해결 단원을 참조하십시오.

Aurora MySQL 메이저 버전 업그레이드의 방법, 계획, 테스트 및 문제 해결에 대해 자세히 알아보려면 Aurora MySQL 현재 위치 업그레이드를 위한 문제 해결을 포함하여 DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드 섹션을 철저히 확인하세요. 또한 일부 인스턴스 유형은 Aurora MySQL 버전 3에서 지원되지 않는다는 점을 참조하세요. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.

Aurora Serverless v1 DB 클러스터의 업그레이드 경로

주 버전 간에 업그레이드하려면 부 버전보다 더 광범위한 계획 및 테스트가 필요합니다. 이 과정은 상당한 시간이 걸릴 수 있습니다. 업그레이드 전, 업그레이드, 업그레이드 후의 활동을 포함한 3단계 프로세스로 업그레이드를 살펴보고자 합니다.

Aurora MySQL 버전 2(MySQL 5.7 호환성 지원)는 계속해서 Aurora Serverless v1 클러스터에 대한 표준 지원을 받습니다.

Aurora MySQL 버전 3(MySQL 8.0 호환성 지원)로 업그레이드하고 계속해서 Aurora Serverless를 실행하려면 Amazon Aurora Serverless v2를 사용하세요. Aurora Serverless v1 및 Aurora Serverless v2 사이의 차이점을 이해하려면 Aurora Serverless v2 및 Aurora Serverless v1 비교 섹션을 참조하세요.

Aurora Serverless v2로 업그레이드: Aurora Serverless v1 클러스터를 Aurora Serverless v2로 업그레이드할 수 있습니다. 자세한 내용은 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드 단원을 참조하십시오.

Amazon Aurora MySQL 호환 버전 1 수명 종료 준비

Amazon Aurora MySQL 호환 버전 1(MySQL 5.6 호환성 지원)은 2023년 2월 28일에 수명 종료될 계획입니다. Amazon은 Aurora MySQL 버전 1을 실행하는 모든 클러스터(프로비저닝됨 및 Aurora Serverless)를 Aurora MySQL 버전 2(MySQL 5.7 호환성 지원) 또는 Aurora MySQL 버전 3(MySQL 8.0 호환성 지원)로 업그레이드하는 것을 권장합니다. Aurora MySQL 버전 1의 지원 기간이 종료되기 전에 이 작업을 수행합니다.

Aurora 프로비저닝 DB 클러스터의 경우 여러 방법으로 Aurora MySQL 버전 1을 Aurora MySQL 버전 2로 업그레이드할 수 있습니다. 현재 위치 업그레이드 수행 방법에서 현재 위치 업그레이드 메커니즘에 대한 지침을 찾을 수 있습니다. 업그레이드를 완료하는 또 다른 방법은 Aurora MySQL 버전 1 클러스터의 스냅샷을 생성하고 스냅샷을 Aurora MySQL 버전 2 클러스터로 복원하는 것입니다. 또는 이전 클러스터와 새 클러스터를 나란히 실행하는 다단계 프로세스를 따를 수도 있습니다. 각 방법에 대한 자세한 내용은 DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드 단원을 참조하세요.

Aurora Serverless v1 DB 클러스터의 경우 Aurora MySQL 버전 1에서 Aurora MySQL 버전 2로 인플레이스 업그레이드를 수행할 수 있습니다. 자세한 방법은 Aurora Serverless v1 DB 클러스터 수정 섹션을 참조하세요.

Aurora 프로비저닝 DB 클러스터의 경우 2단계 업그레이드 프로세스를 사용하여 Aurora MySQL 버전 1을 Aurora MySQL 버전 3으로 업그레이드할 수 있습니다.

1. 앞에서 설명한 방법을 사용하여 Aurora MySQL 버전 1을 Aurora MySQL 버전 2로 업그레이드합니다.
2. 버전 1에서 버전 2로 업그레이드하는 것과 동일한 방법을 사용하여 Aurora MySQL 버전 2에서 Aurora MySQL 버전 3으로 업그레이드합니다. 자세한 내용은 Aurora MySQL 버전 2에서 버전 3으로 업그레이드를 참조하세요. Aurora MySQL 버전 2와 3의 기능 차이점을 기록합니다.

Aurora 메이저 버전의 지원 종료 날짜는 Amazon Aurora에서 확인할 수 있습니다. Amazon은 지원 종료일 이전에 직접 업그레이드하지 않은 클러스터를 자동으로 업그레이드합니다. 지원 종료 날짜 이후 클러스터에 대해 예약된 유지 관리 기간 동안 후속 메이저 버전으로 이러한 자동 업그레이드가 수행됩니다.

다음은 지원이 끝나는 Aurora MySQL 버전 1 클러스터(프로비저닝됨 및 Aurora Serverless)를 업그레이드하기 위한 추가 마일스톤입니다. 각 시작 시간은 00:00 UTC(협정 세계 표준시)입니다.

1. 이제 2023년 2월 28일까지 - Aurora MySQL 버전 1(MySQL 5.6 호환성 지원) 클러스터를 Aurora MySQL 버전 2(MySQL 5.7 호환성 지원)으로 업그레이드할 수 있습니다. Aurora 프로비저닝 DB 클러스터의 경우 Aurora MySQL 버전 2를 Aurora MySQL 버전 3(MySQL 8.0 호환성 지원)로 업그레이드할 수 있습니다.
2. 2023년 1월 16일 - 이 시간 이후에는 AWS Management Console 또는 AWS Command Line Interface(AWS CLI) 중 하나에서 새 Aurora MySQL 버전 1 클러스터 또는 인스턴스를 생성할 수 없습니다. 새 보조 리전을 Aurora 글로벌 데이터베이스에 추가할 수 없습니다. 이 시간 이후에는 5단계와 6단계를 완료할 수 없기 때문에 계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구에 설명된 대로 예상치 못한 중단을 복구할 수 있는 능력에 영향을 줄 수 있습니다. 또한 Aurora MySQL 버전 1을 실행하는 리전 간 읽기 전용 복제본을 새로 생성할 수 없습니다. 2023년 2월 28일까지 기존 Aurora MySQL 버전 1 클러스터에 대해 다음 작업을 수행할 수 있습니다.
   - Aurora MySQL 버전 1 클러스터에서 가져온 스냅샷을 원본 스냅샷 클러스터와 동일한 버전으로 복원합니다.
   - 읽기 전용 복제본 추가(Aurora Serverless DB 클러스터에는 해당되지 않음)
   - 인스턴스 구성을 변경합니다.
   - 특정 시점으로 복원을 수행합니다.
   - 기존 버전 1 클러스터의 복제를 생성합니다.
   - Aurora MySQL 버전 2 이상을 실행하는 새로운 리전 간 읽기 전용 복제본을 생성합니다.
3. 2023년 2월 28일 - 이 시간이 지나면 다음 예정된 유지 관리 기간 내에 Aurora MySQL 버전 1 클러스터를 Aurora MySQL 버전 2의 기본 버전으로 자동 업그레이드할 계획입니다. Aurora MySQL 버전 1 DB 스냅샷을 복원하면 복원된 클러스터가 자동으로 Aurora MySQL 버전 2의 기본 버전으로 업그레이드됩니다.

주 버전 간에 업그레이드하려면 부 버전보다 더 광범위한 계획 및 테스트가 필요합니다. 이 과정은 상당한 시간이 걸릴 수 있습니다.

가동 중지 시간을 줄이는 것이 최우선 순위인 상황에서는 프로비저닝된 Amazon Aurora DB 클러스터에서 메이저 버전 업그레이드를 수행하기 위해 블루/그린 배포를 사용할 수도 있습니다. 블루/그린

배포는 프로덕션 환경을 복사하는 스테이징 환경을 만듭니다. 프로덕션 워크로드에 영향을 주지 않고 그린(스테이징) 환경에서 Aurora DB 클러스터를 변경할 수 있습니다. 전환은 일반적으로 1분도 걸리지 않으며 데이터 손실이 발생하지 않고 애플리케이션을 변경할 필요도 없습니다. 자세한 내용은 Aurora용 Amazon RDS 블루/그린 배포 개요 단원을 참조하십시오.

업그레이드가 완료되면 후속 작업을 수행해야 할 수도 있습니다. 예를 들어 SQL 호환성의 차이, 특정 MySQL 관련 기능의 작동 방식 또는 이전 버전과 새 버전 간의 매개 변수 설정으로 인해 필요할 수 있습니다.

Aurora MySQL 메이저 버전 업그레이드의 방법, 계획, 테스트 및 문제 해결에 대해 자세히 알아보려면 DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드 섹션을 철저히 확인하세요.

이 지원 종료 프로세스의 영향을 받는 클러스터 찾기

이 지원 종료 프로세스의 영향을 받는 클러스터를 찾으려면 다음 절차를 따르세요.

> **Important**
>
> 리소스가 있는 모든 AWS 리전과 각 AWS 계정에서 이 지침을 수행해야 합니다.

콘솔

Aurora MySQL 버전 1 클러스터 찾기

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 데이터베이스별 필터(Filter by databases) 상자에서 5.6을 입력합니다.
4. 엔진 열에서 Aurora MySQL 확인합니다.

AWS CLI

AWS CLI를 사용하여 이 지원 종료 프로세스의 영향을 받는 클러스터를 찾으려면 describe-db-clusters 명령을 호출합니다. 다음 샘플 스크립트를 사용할 수 있습니다.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +------------------------------------------+
        |             DescribeDBClusters           |
        +---------------+--------------+-----------+
        |     DBCI      |      EM      |    EV     |
        +---------------+--------------+-----------+
        |  my-database-1|  serverless  |  5.6.10a  |
        +---------------+--------------+-----------+
```

RDS API

Aurora MySQL 버전 1을 실행하는 Aurora MySQL DB 클러스터를 찾으려면 다음 필수 파라미터를 사용하는 RDS DescribeDBClusters API 작업을 사용합니다.

- `DescribeDBClusters`
  - Filters.Filter.N
    - 명칭
      - engine
    - Values.Value.N
      - ['aurora']

Amazon Aurora MySQL DB 클러스터 업그레이드

Aurora MySQL DB 클러스터를 업그레이드하여 버그 수정, 새로운 Aurora MySQL 기능 또는 완전히 새로운 버전의 기본 데이터베이스 엔진으로 변경할 수 있습니다. 다음 단원에서는 방법을 보여 줍니다.

> **Note**
>
> 수행하는 업그레이드 유형은 클러스터에 대해 감당할 수 있는 가동 중지 시간, 수행할 검증 테스트의 정도, 사용 사례에 대한 특정 버그 수정 또는 새로운 기능이 얼마나 중요한지, 그리고 자주 소규모 업그레이드를 수행할지 또는 여러 중간 버전에 따라 달라집니다. 각 업그레이드에 대해 클러스터의 주 버전, 부 버전 및 패치 수준을 변경할 수 있습니다. Aurora MySQL 주 버전,

부 버전 및 패치 수준 간의 구별에 익숙하지 않은 경우 Aurora MySQL 버전 번호 확인에서 배경 정보를 참조하십시오.

Tip

블루/그린 배포를 사용하면 DB 클러스터 업그레이드에 필요한 가동 중지를 최소화할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 단원을 참조하십시오.

주제

- Aurora MySQL DB 클러스터의 부 버전 또는 패치 수준 업그레이드
- DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드

Aurora MySQL DB 클러스터의 부 버전 또는 패치 수준 업그레이드

다음 방법을 사용하여 DB 클러스터의 마이너 버전을 업그레이드하거나 DB 클러스터에 패치를 적용할 수 있습니다.

- 엔진 버전을 수정하여 Aurora MySQL 업그레이드(Aurora MySQL 버전 2 및 3의 경우)
- 마이너 Aurora MySQL 버전 간 자동 업그레이드 활성화

제로 중지 시간 패치로 업그레이드 프로세스 중 중단을 줄이는 방법에 대한 자세한 내용은 제로 가동 중지 패치 적용 기능 사용 단원을 참조하세요.

Aurora MySQL DB 클러스터의 마이너 버전 업그레이드 수행에 대한 자세한 내용은 다음 주제를 참조하세요.

주제

- 마이너 버전 업그레이드를 수행하기 전
- Aurora MySQL에 대한 마이너 버전 업그레이드 사전 확인
- 엔진 버전을 수정하여 Aurora MySQL 업그레이드
- 마이너 Aurora MySQL 버전 간 자동 업그레이드 활성화
- 제로 가동 중지 패치 적용 기능 사용

- 대체 블루/그린 업그레이드 기법

마이너 버전 업그레이드를 수행하기 전

마이너 버전 업그레이드 중에 가동 중지 시간을 줄이려면 다음 작업을 수행하는 것이 좋습니다.

- Aurora DB 클러스터 유지 관리는 트래픽이 적은 기간에 수행해야 합니다. 성능 개선 도우미를 사용하여 이러한 기간을 식별하여 유지 관리 기간을 올바르게 구성하세요. 성능 개선 도우미에 대한 자세한 내용은 Amazon RDS에서 성능 개선 도우미를 통한 DB 로드 모니터링을 참조하세요. DB 클러스터 유지 관리 기간에 대한 자세한 내용은 기본 DB 클러스터 유지 관리 기간 조정 섹션을 참조하세요.
- 지수 백오프와 지터를 지원하는 AWS SDK를 모범 사례로 사용하세요. 자세한 내용은 지수 백오프 및 지터를 참조하세요.

Aurora MySQL에 대한 마이너 버전 업그레이드 사전 확인

마이너 버전 업그레이드를 시작하면 Amazon Aurora가 자동으로 사전 검사를 실행합니다.

이러한 사전 점검은 필수입니다. 건너뛸 수 없습니다. 사전 점검은 다음과 같은 이점을 제공합니다.

- 이를 통해 업그레이드 중 예기치 않은 가동 중단을 피할 수 있습니다.
- 비호환성이 있는 경우 Amazon Aurora가 업그레이드를 차단하고 이에 대해 알 수 있는 로그를 제공합니다. 그러면 로그를 사용해 비호환성을 제거함으로써 업그레이드하기 위한 데이터베이스 준비를 마칠 수 있습니다. 비호환성 문제를 제거하는 방법에 대한 자세한 내용은 MySQL 설명서의 업그레이드를 위한 설치 준비를 참조하세요.

사전 점검은 업그레이드를 위해 DB 인스턴스가 중지되기 전에 실행됩니다. 즉, 점검을 실행해도 가동 중지를 일으키지 않습니다. 사전 확인에서 비호환성이 발견되면 Aurora는 DB 인스턴스가 중지되기 전에 자동으로 업그레이드를 취소합니다. 또한 Aurora는 비호환성에 대한 이벤트를 생성합니다. Amazon Aurora 이벤트에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 섹션을 참조하세요.

Aurora는 각 비호환성에 대한 자세한 정보를 로그 파일 `PrePatchCompatibility.log`에 기록합니다. 대부분의 경우 로그 항목에는 비호환성 문제를 해결하기 위한 MySQL 설명서 링크가 포함되어 있습니다. 로그 파일 보기에 대한 자세한 내용은 데이터베이스 로그 파일 보기 및 나열 단원을 참조하십시오.

사전 점검의 특성으로 인해 데이터베이스의 객체를 분석합니다. 이 분석은 리소스를 소비하고 업그레이드가 완료되는 시간을 늘립니다.

엔진 버전을 수정하여 Aurora MySQL 업그레이드

Aurora MySQL DB 클러스터의 마이너 버전을 업그레이드하면 기존 클러스터에 추가 수정 사항과 새로운 기능이 적용됩니다.

이 유형의 업그레이드는 원래 버전과 업그레이드된 버전이 모두 동일한 Aurora MySQL 메이저 버전 2 또는 3인 Aurora MySQL 클러스터에 적용됩니다. 이 프로세스는 Aurora MySQL 메타데이터 변환이나 테이블 데이터 재구성을 포함하지 않기 때문에 빠르고 간단합니다.

이 유형의 업그레이드를 수행하려면 AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터의 엔진 버전을 수정합니다. 예를 들어, 클러스터가 Aurora MySQL 3.x를 실행하는 경우 더 높은 3.x 버전을 선택합니다.

Aurora 글로벌 데이터베이스에서 마이너 업그레이드를 수행하는 경우 기본 클러스터를 업그레이드하기 전에 모든 보조 클러스터를 업그레이드합니다.

> **Note**
>
> Aurora MySQL 버전 3.04.* 이상 또는 버전 2.12*로 마이너 버전 업그레이드를 수행하려면 다음 프로세스를 사용하세요.
>
> 1. 글로벌 클러스터에서 모든 보조 DB 클러스터를 제거합니다. Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거 섹션의 단계를 따르세요.
> 2. 기본 리전의 엔진 버전을 버전 3.04.* 이상 또는 버전 2.12.* 중 적합한 것으로 업그레이드합니다. To modify the engine version of a DB cluster 섹션의 단계를 따르세요.
> 3. 글로벌 클러스터에 보조 리전을 추가합니다. Amazon Aurora Global Database에 AWS 리전 추가 섹션의 단계를 따르세요.

DB 클러스터의 엔진 버전을 수정하는 방법

- 콘솔 사용하여 – 클러스터의 속성을 수정합니다. DB 클러스터 수정(Modify DB cluster) 창의 DB 엔진 버전(DB engine version) 상자에서 Aurora MySQL 엔진 버전을 변경합니다. 일반적인 클러스터 수정 절차에 익숙하지 않을 경우 콘솔, CLI, API를 사용하여 DB 클러스터 수정의 지침을 따르세요.
- AWS CLI를 사용하여 modify-db-cluster AWS CLI 명령을 호출한 후 `--db-cluster-identifier` 옵션에 대한 DB 클러스터 이름, `--engine-version` 옵션에 엔진 버전을 지정합니다.

예를 들어, Aurora MySQL 버전 3.04.1로 업그레이드하려면 `--engine-version` 옵션을 `8.0.mysql_aurora.3.04.1`로 설정합니다. DB 클러스터의 엔진 버전을 즉시 업데이트하려면 `--apply-immediately` 옵션을 설정합니다.

- RDS API를 사용하여 – ModifyDBCluster API 작업을 호출한 후 `DBClusterIdentifier` 파라미터에 대한 DB 클러스터 이름을 지정하고 `EngineVersion` 파라미터에 엔진 버전을 지정합니다. DB 클러스터의 엔진 버전을 즉시 업데이트하려면 `ApplyImmediately` 파라미터를 `true`로 설정합니다.

마이너 Aurora MySQL 버전 간 자동 업그레이드 활성화

Amazon Aurora MySQL DB 클러스터의 경우 Aurora가 DB 클러스터를 자동으로 새로운 마이너 버전으로 업그레이드하도록 지정할 수 있습니다. 이렇게 하려면 DB 클러스터의 `AutoMinorVersionUpgrade` 속성(AWS Management Console의 자동 마이너 버전 업그레이드)을 설정합니다.

자동 업그레이드는 유지 관리 기간 중에 발생합니다. DB 클러스터에서 개별 DB 인스턴스의 유지 관리 기간이 클러스터 유지 관리 기간과 다른 경우 클러스터 유지 관리 기간이 우선합니다.

마이너 버전 자동 업그레이드는 다음 종류의 Aurora MySQL 클러스터에는 적용되지 않습니다.

- Aurora 글로벌 데이터베이스의 일부로 포함된 클러스터
- 리전 간 복제본이 있는 클러스터

운영 중단 기간은 워크로드, 클러스터 크기, 이진 로그 데이터의 양 및 Aurora가 제로 다운타임 패치 적용(ZDP) 기능을 사용할 수 있는지 여부에 따라 달라집니다. Aurora는 데이터베이스 클러스터를 재시작하므로 클러스터를 다시 사용하기 전에 잠시 동안 사용할 수 없게 될 수 있습니다. 특히 이진 로그 데이터의 양은 복구 시간에 영향을 미칩니다. DB 인스턴스는 복구 중에 이진 로그 데이터를 처리합니다. 따라서 이진 로그 데이터의 양이 많으면 복구 시간이 늘어납니다.

> **Note**
>
> Aurora는 DB 클러스터의 모든 DB 인스턴스에 `AutoMinorVersionUpgrade` 설정이 활성화된 경우에만 자동 업그레이드를 수행합니다. 설정하는 방법과 클러스터 및 인스턴스 수준에서 설정을 적용한 경우 작동 방식에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하세요.

그러면 AutoUpgrade가 true로 설정된 DB 클러스터의 인스턴스에 대해 마이너 DB 엔진 버전으로의 업그레이드 경로가 있는 경우 업그레이드가 수행됩니다. AutoUpgrade 설정은 동적이며 RDS에서 설정합니다.
자동 마이너 버전 업그레이드는 기본 마이너 버전으로 수행됩니다.

다음과 같은 CLI 명령을 사용하여 Aurora MySQL 클러스터의 모든 DB 인스턴스에 대한 AutoMinorVersionUpgrade 설정의 상태를 확인할 수 있습니다.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

이 명령을 실행하면 다음과 비슷한 출력이 생성됩니다.

```
[
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

이 예시에서는 클러스터의 DB 인스턴스 중 하나에 대해 마이너 버전 자동 업그레이드 사용이 꺼져 있으므로, DB 클러스터 cluster-57-2020-06-03-6411에 대한 마이너 버전 자동 업그레이드 사용도 꺼져 있습니다.

제로 가동 중지 패치 적용 기능 사용

Aurora MySQL DB 클러스터에 대한 업그레이드를 수행할 때는 데이터베이스가 종료되고 업그레이드되는 동안 중단될 가능성이 있습니다. 기본적으로 데이터베이스가 사용 중인 동안 업그레이드를 시작

하면 DB 클러스터에서 처리하는 모든 연결 및 트랜잭션이 손실됩니다. 업그레이드를 수행하기 위해 데이터베이스가 유휴 상태가 될 때까지 기다리려면 오랜 시간을 기다려야 할 수 있습니다.

제로 가동 중지 패치 적용(ZDP) 기능은 최선의 노력을 기반으로 Aurora MySQL 업그레이드 중에 클라이언트 연결을 유지하려고 시도합니다. ZDP가 성공적으로 완료되면 업그레이드 진행 중에 애플리케이션 세션이 유지되고 데이터베이스 엔진이 다시 시작됩니다. 데이터베이스 엔진이 다시 시작되면 몇 초에서 약 1분간 처리량이 저하될 수 있습니다.

ZDP는 다음에는 적용되지 않습니다.

- 운영 체제(OS) 패치 및 업그레이드
- 메이저 버전 업그레이드

ZDP를 사용할 수 있는 모든 Aurora MySQL 버전과 DB 인스턴스 클래스에 사용할 수 있습니다.

Aurora Serverless v1 또는 Aurora Global Database에는 ZDP가 지원되지 않습니다.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 개발 및 테스트에 T 인스턴스 클래스 사용 섹션을 참조하세요.

ZDP 중에 MySQL 오류 로그에서 중요한 속성의 지표를 볼 수 있습니다. AWS Management Console의 이벤트 페이지에서 Aurora MySQL이 ZDP를 사용하거나 ZDP를 사용하지 않도록 선택하는 경우에 대한 정보도 볼 수 있습니다.

Aurora MySQL에서 Aurora는 이진 로그 복제가 활성화되었는지와 상관없이 가동 중지 시간이 없는 패치 적용을 수행할 수 있습니다. 바이너리 로그 복제가 활성화되어 있으면 Aurora MySQL은 ZDP 작업 중에 binlog 대상에 대한 연결을 자동으로 삭제합니다. Aurora MySQL은 binlog 대상에 자동으로 다시 연결하고 재시작이 완료된 후 복제를 재개합니다.

또한 ZDP는 Aurora MySQL의 재부팅 개선 사항과 함께 작동합니다. 라이터 DB 인스턴스에 패치를 적용하면 동시에 리더에도 자동으로 패치가 적용됩니다. 패치를 수행한 후 Aurora는 라이터와 리더 DB 인스턴스에서 연결을 복원합니다.

다음 조건에서는 ZDP가 성공적으로 완료되지 않을 수 있습니다.

- 장기간 쿼리 또는 트랜잭션이 진행 중인 경우 이 경우 Aurora가 ZDP를 수행할 수 있다면 열린 트랙잭션은 취소되지만, 연결은 유지됩니다.
- 임시 표, 사용자 잠금 또는 표 잠금은 가령 데이터 정의 언어(DDL) 문이 실행되는 동안 사용됩니다. Aurora는 이러한 연결을 끊습니다.
- 보류 중인 파라미터 변경 사항이 존재하는 경우

이러한 조건 때문에 ZDP 수행을 위한 적절한 기간을 확보할 수 없는 경우 패치 적용이 표준 동작으로 돌아갑니다.

성공적인 ZDP 작업 후에 연결은 그대로 유지되지만 일부 변수와 기능은 다시 초기화됩니다. 다음 유형의 정보는 제로 다운타임 패치 적용으로 인한 다시 시작 중에 보관되지 않습니다.

- 글로벌 변수 Aurora는 세션 변수를 복원하지만 다시 시작 후 글로벌 변수를 복원하지 않습니다.
- 상태 변수. 특히 엔진 상태로 보고되는 가동 시간 값은 ZDR 또는 ZDP 메커니즘을 사용하는 다시 시작 후 재설정됩니다.
- `LAST_INSERT_ID`.
- 테이블의 인 메모리 `auto_increment` 상태. 인 메모리 자동 증분 상태는 다시 초기화됩니다. 자동 증분 값에 대한 자세한 내용은 MySQL 참조 매뉴얼을 참조하세요.
- `INFORMATION_SCHEMA` 및 `PERFORMANCE_SCHEMA` 테이블의 진단 정보. 이 진단 정보는 `SHOW PROFILE` 및 `SHOW PROFILES`와 같은 명령 출력에도 표시됩니다.

제로 다운타임 다시 시작과 관련된 다음 활동은 [이벤트(Events) 페이지에 보고됩니다.

- 제로 다운타임으로 데이터베이스를 업그레이드하려는 시도
- 제로 다운타임으로 데이터베이스를 업그레이드하려는 시도가 완료되었습니다. 이벤트는 프로세스에 걸린 시간을 보고합니다. 또한 이벤트는 다시 시작 중에 보관된 연결 수와 삭제된 연결 수를 보고합니다. 데이터베이스 오류 로그를 참조하여 다시 시작 중에 발생한 활동에 대한 자세한 내용을 확인할 수 있습니다.

대체 블루/그린 업그레이드 기법

경우에 따라서는 오래된 클러스터에서 업그레이드된 클러스터로 즉시 전환하는 것이 가장 중요한 우선순위일 수 있습니다. 또는 이전 클러스터와 새 클러스터를 나란히 실행하는 다단계 프로세스를 따를 수도 있습니다. 이 경우 새 클러스터가 인수할 준비가 될 때까지 이전 클러스터의 데이터를 새 클러스터로 복제합니다. 세부 정보는 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용을 참조하세요.

DB 클러스터의 Amazon Aurora MySQL 메이저 버전 업그레이드

3.04.1과 같은 Aurora MySQL 버전 번호에서 3은 메이저 버전을 나타냅니다. Aurora MySQL 버전 2는 MySQL 5.7과 호환됩니다. Aurora MySQL 버전 3은 MySQL 8.0과 호환됩니다.

주 버전 간에 업그레이드하려면 부 버전보다 더 광범위한 계획 및 테스트가 필요합니다. 이 과정은 상당한 시간이 걸릴 수 있습니다. 업그레이드가 완료되면 후속 작업을 수행해야 할 수도 있습니다. 예를 들어 SQL 호환성의 차이 또는 특정 MySQL 관련 기능의 작동 방식 때문에 이러한 문제가 발생할 수 있습니다. 또는 이전 버전과 새 버전 간의 파라미터 설정이 다르기 때문에 발생할 수 있습니다.

목차

Aurora MySQL 버전 2에서 버전 3으로 업그레이드

MySQL 5.7 호환 클러스터가 있고 이를 MySQL 8.0 호환 클러스터로 업그레이드하려는 경우 클러스터 자체에서 업그레이드 프로세스를 실행하여 업그레이드할 수 있습니다. 이러한 종류의 업그레이드는 새 클러스터를 생성하여 수행하는 업그레이드와 달리 현재 위치 업그레이드입니다. 이 기술은 클러스터의 동일한 엔드포인트 및 기타 특성을 유지합니다. 해당 업그레이드는 모든 데이터를 새 클러스터 볼륨으로 복사할 필요가 없기 때문에 비교적 빠릅니다. 이러한 안정성은 애플리케이션 내 구성 변경을 최소화하는 데 도움이 됩니다. 또한 업그레이드된 클러스터에 대한 테스트 횟수를 줄이는 데도 도움이 됩니다. DB 인스턴스와 인스턴스 클래스의 수가 모두 동일하게 유지되기 때문입니다.

현재 위치 업그레이드 메커니즘에는 작업이 진행되는 동안 DB 클러스터를 종료하는 작업이 포함됩니다. Aurora는 완전히 종료하고 트랜잭션 롤백 및 제거 실행 취소와 같은 미결 작업을 완료합니다. 자세한 내용은 [Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식] 단원을 참조하십시오.

현재 위치 업그레이드 방법은 수행 방법이 단순하며 관련 애플리케이션에 대한 구성 변경을 최소화하므로 편리합니다. 예를 들어, 현재 위치 업그레이드는 클러스터의 엔드포인트 및 DB 인스턴스 세트를 보관합니다. 그러나 현재 위치 업그레이드에 필요한 시간은 스키마의 속성과 클러스터 사용 수준에 따라 달라질 수 있습니다. 따라서 클러스터의 요구 사항에 따라 다음의 업그레이드 방법 중에서 선택할 수 있습니다.

- [현재 위치 업그레이드]
- [블루/그린 배포]
- [스냅샷 복원]

> **Note**
>
> 스냅샷 복원 업그레이드 방법에 AWS CLI 또는 RDS API를 사용할 경우, 후속 작업을 실행하여 복원된 DB 클러스터에서 라이터 DB 인스턴스를 생성해야 합니다.

Aurora MySQL 버전 3 및 새로운 기능에 대한 일반적인 정보는 Aurora MySQL 버전 3은 MySQL 8.0과 호환 섹션을 참조하세요.

업그레이드 계획에 대한 자세한 내용은 Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획 및 현재 위치 업그레이드 수행 방법을 참조하세요.

Aurora MySQL 주 버전 업그레이드 경로

모든 종류 또는 버전의 Aurora MySQL 클러스터가 현재 위치 업그레이드 메커니즘을 사용할 수 있는 것은 아닙니다. 다음 테이블을 참조하여 각 Aurora MySQL 클러스터에 대한 적절한 업그레이드 경로를 확인할 수 있습니다.

Aurora MySQL DB 클러스터 유형	현재 위치 업그레이드를 사용할 수 있습니까?	작업
Aurora MySQL 프로비저닝된 클러스터(버전 2)	예	현재 위치 업그레이드는 MySQL 5.7 호환 Aurora MySQL 클러스터에 지원됩니다. Aurora MySQL 버전 3으로의 업그레이드에 대한 자세한 내용은 Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획 및 현재 위치 업그레이드 수행 방법 섹션을 참조하세요.
Aurora MySQL 프로비저닝된 클러스터(버전 3)	해당 사항 없음	마이너 버전 업그레이드 절차를 사용하여 Aurora MySQL 버전 3 간에 업그레이드하세요.
Aurora Serverless v1 클러스터	해당 사항 없음	Aurora Serverless v1은 Aurora MySQL의 버전 2에서만 지원됩니다.

Aurora MySQL DB 클러스터 유형	현재 위치 업그레이드를 사용할 수 있습니까?	작업
Aurora Serverless v2 클러스터	해당 사항 없음	Aurora Serverless v2는 Aurora MySQL의 버전 3에서만 지원됩니다.
Aurora Global Database의 클러스터	예	Aurora MySQL을 버전 2에서 버전 3으로 업그레이드하려면 Aurora Global Database에 있는 클러스터에 대한 현재 위치 업그레이드 수행 방법을 따르세요. 글로벌 클러스터에 업그레이드를 수행합니다. Aurora는 기본 클러스터와 글로벌 데이터베이스의 모든 보조 클러스터를 동시에 업그레이드합니다. AWS CLI 또는 RDS API를 사용한다면, `modify-global-cluster` 명령 또는 `ModifyGlobalCluster` 작업을 `modify-db-cluster` 또는 `ModifyDBCluster` 대신 호출합니다. `lower_case_table_names` 파라미터를 기본값으로 설정하고 글로벌 데이터베이스를 재부팅하는 경우에만 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수 있습니다. 자세한 내용은 메이저 버전 업그레이드 단원을 참조하십시오.
병렬 쿼리 클러스터	예	현재 위치 업그레이드를 수행할 수 있습니다.
이진 로그 복제 대상인 클러스터	가능	Aurora MySQL 클러스터에서 바이너리 로그 복제를 했다면 현재 위치 업그레이드를 수행할 수 있습니다. 바이너리 로그 복제가 RDS for MySQL 또는 온프레미스 MySQL DB 인스턴스에서 생성되었다면 업그레이드를 수행할 수 없습니다. 이 경우 스냅샷 복원 메커니즘을 사용하여 업그레이드할 수 있습니다.

Aurora MySQL DB 클러스터 유형	현재 위치 업그레이드를 사용할 수 있습니까?	작업
DB 인스턴스가 0인 클러스터	아니요	AWS CLI 또는 RDS API를 사용하여 연결된 DB 인스턴스 없이 Aurora MySQL 클러스터를 생성할 수 있습니다. 같은 방법으로 Aurora MySQL 클러스터 볼륨의 데이터는 그대로 유지하면서 클러스터에서 모든 DB 인스턴스를 제거할 수도 있습니다. 클러스터에 0의 DB 인스턴스가 있는 동안에는 현재 위치 업그레이드를 수행할 수 없습니다. 업그레이드 메커니즘을 사용하려면 클러스터의 라이터 인스턴스가 시스템 테이블, 데이터 파일 등에 대한 변환을 수행해야 합니다. 이 경우 AWS CLI 또는 RDS API를 사용하여 클러스터의 라이터 인스턴스를 만듭니다. 그런 다음 현재 위치 업그레이드를 수행할 수 있습니다.
백트랙이 활성화된 클러스터	예	백트랙 기능을 사용하는 Aurora MySQL 클러스터에 현재 위치 업그레이드를 수행할 수 있습니다. 그러나 업그레이드 후에는 업그레이드 전에 클러스터를 백트랙할 수 없습니다.

Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식

Aurora MySQL 는 주 버전 업그레이드를 다단계 프로세스로 수행합니다. 업그레이드의 현재 상태를 확인할 수 있습니다. 일부 업그레이드 단계는 진행 정보도 제공합니다. 각 단계가 시작되면 Aurora MySQL 에서는 이벤트를 기록합니다. RDS 콘솔의 이벤트 페이지에서 발생하는 이벤트를 검사할 수 있습니다. 이벤트 작업에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 단원을 참조하십시오.

> **Important**
>
> 프로세스가 시작되면 업그레이드가 성공하거나 실패할 때까지 프로세스가 실행됩니다. 진행 중에는 업그레이드를 취소할 수 없습니다. 업그레이드가 실패하면 Aurora 은 모든 변경 사항을 롤백하며 클러스터는 이전과 동일한 엔진 버전, 메타데이터 등을 갖춥니다.

업그레이드 프로세스는 세 단계로 구성됩니다:

1. Aurora는 업그레이드 프로세스를 시작하기 전에 일련의 [사전 확인](#)을 수행합니다. Aurora 가 이러한 검사를 수행하는 동안 클러스터가 계속 실행됩니다. 예를 들어 클러스터는 준비된 상태의 XA 트랜잭션을 가질 수 없거나 DDL (데이터 정의 언어) 문을 처리할 수 없습니다. 예를 들어 특정 종류의 SQL문을 제출하는 응용 프로그램을 종료해야 할 수 있습니다. 또는 특정 장기 실행 명령문이 완료될 때까지 기다릴 수도 있습니다. 그런 다음 업그레이드를 다시 시도하십시오. 일부 검사는 업그레이드를 방해하지는 않지만 업그레이드 시간이 오래 걸리게 할 수 있는 조건을 테스트합니다.

   Aurora 은 필수 조건이 충족되지 않음을 감지하면 이벤트 세부 정보에서 식별된 조건을 수정합니다. [Aurora MySQL 현재 위치 업그레이드를 위한 문제 해결](#)의 지침을 따릅니다. Aurora 가 느리게 업그레이드될 수 있는 조건을 감지하면 장기간 업그레이드를 모니터링하도록 계획합니다.

2. Aurora 는 클러스터를 오프라인으로 전환합니다. 그 다음 Aurora 가 이전 단계에서의 테스트와 유사한 테스트 세트를 수행하여 종료 프로세스 중 새로운 문제가 발생하지 않았는지 확인합니다. Aurora 가 이 시점에서 업그레이드를 방해하는 조건을 감지하면 Aurora는 업그레이드를 취소하고 클러스터를 다시 온라인 상태로 만듭니다. 이 경우 조건이 더 이상 적용되지 않는 시점을 확인하고 업그레이드를 다시 시작하십시오.

3. Aurora 는 클러스터 볼륨의 스냅샷을 생성합니다. 업그레이드가 완료된 후 호환성 또는 다른 종류의 문제를 발견했다고 가정합니다. 또는 기존 클러스터와 업그레이드된 클러스터를 모두 사용하여 테스트를 수행한다고 가정합니다. 이 경우 이 스냅샷에서 복원하여 기존 엔진 버전 및 원본 데이터로 새 클러스터를 만들 수 있습니다.

   > **Tip**
   >
   > 이 스냅샷은 수동 스냅샷입니다. 그러나 Aurora 는 수동 스냅샷에 대한 할당량에 도달한 경우에도 이를 생성하고 업그레이드 프로세스를 계속할 수 있습니다. 이 스냅샷은 삭제할 때까지 영구적으로(필요한 경우) 유지됩니다. 모든 업그레이드 후 테스트를 완료한 후 이 스냅샷을 삭제하여 스토리지 요금을 최소화할 수 있습니다.

4. Aurora 는 클러스터 볼륨을 복제합니다. 복제는 실제 테이블 데이터를 포함하지 않는 빠른 작업입니다. Aurora 는 업그레이드 중에 문제가 발생하면 복제된 클러스터 볼륨에서 원래 데이터로 복구되며 클러스터가 다시 온라인 상태로 전환됩니다. 업그레이드 중 임시 복제된 볼륨은 단일 클러스터 볼륨에 관한 클론 수에 부여되는 일반적인 제한이 적용되지 않습니다.

5. Aurora 는 작성자 DB 인스턴스에 대하여 새로 종료를 수행합니다. 클린 종료 중 다음 작업에 대해 15분마다 진행률 이벤트를 기록합니다. RDS 콘솔의 이벤트 페이지에서 발생하는 이벤트를 검사할 수 있습니다.

   • Aurora 는 이전 버전의 행에 대한 실행 취소 레코드를 소거합니다.

- Aurora 커밋되지 않은 트랜잭션을 롤백합니다.

6. Aurora 는 라이터 DB 인스턴스에서 엔진 버전을 업그레이드합니다.
   - Aurora 는 라이터 DB 인스턴스에 새 엔진 버전용 바이너리를 설치합니다.
   - Aurora 는 라이터 DB 인스턴스를 사용하여 데이터를 MySQL 5.7 호환 형식으로 업그레이드합니다. 이 단계에서 Aurora는 시스템 테이블을 수정하고 클러스터 볼륨의 데이터에 영향을 주는 다른 변환을 수행합니다. 특히, Aurora는 MySQL 5.7 파티션 형식과 호환되도록 시스템 테이블의 파티션 메타 데이터를 업그레이드합니다. 클러스터의 테이블에 파티션 수가 많은 경우 이 단계에 시간이 오래 걸릴 수 있습니다.

     이 단계에서 오류가 발생하면 MySQL 오류 로그에서 세부 정보를 찾을 수 있습니다. 이 단계가 시작된 후 어떤 이유로든 업그레이드 프로세스가 실패하면 Aurora 는 복제된 클러스터 볼륨에서 원래 데이터를 복원합니다.
7. Aurora 는 Reader DB 인스턴스에서 엔진 버전을 업그레이드합니다.
8. 업그레이드 프로세스가 완료되었습니다. Aurora는 업그레이드 프로세스가 성공적으로 완료되었음을 나타내는 최종 이벤트를 기록합니다. 이제 DB 클러스터가 새로운 주 버전을 실행하고 있습니다.

Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획

업그레이드할 적절한 시기와 접근 방식을 결정하는 데 도움이 되도록 Aurora MySQL 버전 3과 현재 환경의 차이점을 알아볼 수 있습니다.

- RDS for MySQL 8.0 또는 MySQL 8.0 커뮤니티 버전에서 변환하는 경우 Aurora MySQL 버전 3과 MySQL 8.0 커뮤니티 에디션 비교 섹션을 참조하세요.
- Aurora MySQL 버전 2, RDS for MySQL 5.7 또는 커뮤니티 MySQL 5.7에서 업그레이드하는 경우 Aurora MySQL 버전 2와 Aurora MySQL 버전 3의 비교 섹션을 참조하세요.
- 모든 사용자 지정 파라미터 그룹의 MySQL 8.0 호환 버전을 새로 만듭니다. 필요한 모든 사용자 파라미터 값을 새 파라미터 그룹에 적용합니다. 파라미터 변경에 대해 알아보려면 Aurora MySQL 버전 3에 대한 파라미터 변경 사항 섹션을 참조하세요.
- MySQL 8.0 Community Edition에 도입된 새로운 예약 키워드의 사용에 대해 Aurora MySQL 버전 2 데이터베이스 스키마 및 객체 정의를 검토하세요. 업그레이드하기 전에 완료하세요. 자세한 내용은 MySQL 설명서의 MySQL 8.0 새 키워드 및 예약어를 참조하세요.

또한 MySQL별 업그레이드 고려 사항과 팁에 대한 자세한 내용은 MySQL 참조 설명서의 MySQL 8.0의 변경 사항을 참조하세요. 예를 들어 `mysqlcheck --check-upgrade` 명령을 사용하여 기존 Aurora MySQL 데이터베이스를 분석하고 잠재적인 업그레이드 문제를 식별할 수 있습니다.

> **Note**
>
> 현재 위치 업그레이드 또는 스냅샷 복원 기술을 사용하여 Aurora MySQL 버전 3으로 업그레이드할 때는 더 큰 DB 인스턴스 클래스를 사용하는 것이 좋습니다. 예로는 db.r5.24xlarge 및 db.r6g.16xlarge가 있습니다. 이렇게 하면 DB 인스턴스에서 사용 가능한 CPU 용량의 대부분을 사용하여 업그레이드 프로세스를 더 빠르게 완료할 수 있습니다. 메이저 버전 업그레이드가 완료된 후 원하는 DB 인스턴스 클래스로 변경할 수 있습니다.

업그레이드를 완료하면 Aurora MySQL 버전 3에 대한 업그레이드 후 정리에서 업그레이드 후 절차를 따를 수 있습니다. 마지막으로 애플리케이션의 기능과 성능을 테스트합니다.

RDS에서 MySQL 또는 커뮤니티 MySQL로 변환하는 경우 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션에 설명된 마이그레이션 절차를 따르세요. 경우에 따라 마이그레이션의 일부로 이진 로그 복제를 사용하여 데이터를 Aurora MySQL 버전 3 클러스터와 동기화할 수 있습니다. 그렇다면 소스 시스템에서 대상 DB 클러스터와 호환되는 버전을 실행해야 합니다.

메이저 버전 간의 클러스터를 업그레이드한 후 애플리케이션 및 관리 절차가 원활하게 작동하도록 하려면 몇 가지 사전 계획 및 준비 작업을 수행하세요. AWS CLI 스크립트 또는 RDS API 기반 애플리케이션에 대해 업데이트하려는 관리 코드의 종류를 확인하려면 현재 위치 업그레이드가 클러스터의 파라미터 그룹에 미치는 영향 섹션을 참조하세요. Aurora MySQL 버전 간의 클러스터 속성 변경 섹션도 참조하세요.

업그레이드 중에 발생할 수 있는 문제를 알아보려면 Aurora MySQL 현재 위치 업그레이드를 위한 문제 해결 섹션을 참조하세요. 업그레이드 시간이 오래 걸릴 수 있는 문제를 대상으로 이러한 조건을 미리 테스트 및 수정할 수 있습니다.

> **Note**
>
> 현재 위치 업그레이드에는 작업이 진행되는 동안 DB 클러스터를 종료하는 작업이 포함됩니다. Aurora MySQL은 완전히 종료하고 제거 실행 취소와 같은 미결 작업을 완료합니다. 제거해야 할 실행 취소 레코드가 많으면 업그레이드 시간이 많이 소요될 수도 있습니다. 기록 목록 길이(HLL)가 줄어든 후에 한하여 업그레이드를 수행하는 것이 좋습니다. 일반적으로 허용되는 HLL 값은 100,000 이하입니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.

DB 클러스터를 복제하여 업그레이드 시뮬레이션

업그레이드된 클러스터에 대한 애플리케이션 호환성, 성능, 유지 관리 절차 및 이와 유사한 고려 사항을 확인할 수 있습니다. 이를 위해 실제 업그레이드를 수행하기 전에 업그레이드 시뮬레이션을 수행할 수 있습니다. 이 기술은 생산 클러스터에 특히 유용할 수 있습니다. 여기서 가동 중지 시간을 최소화하고, 업그레이드가 완료되는 즉시 업그레이드된 클러스터를 준비하는 것이 중요합니다.

다음 단계를 사용합니다.

1. 기존 클러스터의 클론을 생성합니다. Aurora DB 클러스터에 대한 볼륨 복제의 절차를 따르십시오.
2. 기존 클러스터에서처럼 유사한 라이터 및 리더 DB 인스턴스 세트를 설정합니다.
3. 클론 클러스터의 현재 위치 업그레이드 수행 현재 위치 업그레이드 수행 방법의 절차를 따르십시오.

   클론을 생성한 후 즉시 업그레이드를 시작합니다. 이러면 클러스터 볼륨이 기존 클러스터의 상태와 여전히 동일합니다. 업그레이드를 수행하기 전에 클론이 유휴 상태인 경우 Aurora 는 백그라운드에서 데이터베이스 정리 프로세스를 수행합니다. 이 경우 클론 업그레이드는 기존 클러스터 업그레이드의 정확한 시뮬레이션이 아닙니다.
4. 복제된 클러스터를 사용하여 응용 프로그램 호환성, 성능, 관리 절차 등을 테스트합니다.
5. 문제가 발생하면 업그레이드 계획을 조정하여 해당 문제를 해결하세요. 예를 들어 모든 애플리케이션 코드를 상위 버전의 기능 세트와 호환되도록 조정합니다. 클러스터의 데이터 양에 따라 업그레이드에 걸리는 시간을 예측합니다. 클러스터를 사용하지 않을 때 업그레이드를 예약하도록 선택할 수도 있습니다.
6. 애플리케이션 및 워크로드가 테스트 클러스터에서 제대로 작동하면, 생산 클러스터에 현재 위치 업그레이드를 수행할 수 있습니다.
7. 메이저 버전 업그레이드 중 클러스터의 총 가동 중지 시간을 최소화하기 위한 노력을 기울이세요. 그러기 위해서는 업그레이드 시 클러스터의 워크로드가 낮거나 0인지 확인해야 합니다. 특히 업그레이드를 시작할 때 진행 중인 장기 실행 트랜잭션이 없는지 확인하십시오.

블루/그린 배포

경우에 따라서는 오래된 클러스터에서 업그레이드된 클러스터로 즉시 전환하는 것이 가장 중요한 우선순위일 수 있습니다. 또는 이전 클러스터와 새 클러스터를 나란히 실행하는 다단계 프로세스를 따를 수도 있습니다. 이 경우 새 클러스터가 인수할 준비가 될 때까지 이전 클러스터의 데이터를 새 클러스터로 복제합니다. 세부 정보는 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용을 참조하세요.

Aurora MySQL에 대한 메이저 버전 업그레이드 사전 확인

MySQL 5.7에서 MySQL 8.0으로 전환하는 등 MySQL을 하나의 메이저 버전에서 다른 메이저 버전으로 업그레이드하려면 신중한 계획 및 준비가 필요한 몇 가지 중요한 아키텍처 변경이 필요합니다. 주로 데이터베이스 엔진 소프트웨어 업데이트와 경우에 따라 시스템 테이블 업데이트에 중점을 두는 마이너 버전 업그레이드와 달리, 메이저 MySQL 업그레이드는 데이터베이스가 메타데이터를 저장하고 관리하는 방식에 근본적인 변화를 가져오는 경우가 많습니다.

이러한 비호환성을 식별하는 데 도움이 되도록 Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 때 Aurora는 업그레이드 호환성 검사(사전 확인)를 자동으로 실행하여 데이터베이스 클러스터의 객체를 검사하고 업그레이드를 계속하지 못하도록 차단할 수 있는 알려진 비호환성을 식별합니다. Aurora MySQL 사전 확인에 대한 자세한 내용은 Aurora MySQL에 대한 사전 확인 설명 참조 섹션을 참조하세요. Aurora 사전 검사는 Community MySQL 업그레이드 확인기 유틸리티에서 실행하는 검사 외에 추가로 실행됩니다.

이러한 사전 점검은 필수입니다. 건너뛸 수 없습니다. 사전 점검은 다음과 같은 이점을 제공합니다.

- 업그레이드 실패로 가동 중지 시간이 길어질 가능성을 줄일 수 있습니다.
- 비호환성이 있는 경우 Amazon Aurora가 업그레이드 진행을 차단하고 이에 대해 알 수 있는 로그를 제공합니다. 그러면 로그를 사용해 비호환성을 해결함으로써 버전 3으로 업그레이드하기 위한 데이터베이스 준비를 마칠 수 있습니다. 비호환성 문제를 해결하는 방법에 대한 자세한 내용은 MySQL 설명서의 업그레이드를 위한 설치 준비 및 MySQL 8.0으로 업그레이드할 때를 참조하세요. 알아야 할 내용을 참조하세요.

  MySQL 8.0으로 업그레이드하는 방법에 대한 자세한 내용은 MySQL 설명서에서 MySQL 업그레이드를 참조하세요.

사전 확인은 메이저 버전 업그레이드를 위해 DB 클러스터를 오프라인으로 전환하기 전에 실행됩니다. 사전 확인에서 비호환성이 발견되면 Aurora는 DB 인스턴스가 중지되기 전에 자동으로 업그레이드를 취소합니다. 또한 Aurora는 비호환성에 대한 이벤트를 생성합니다. Amazon Aurora 이벤트에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 섹션을 참조하세요.

사전 확인이 완료되면 Aurora는 `upgrade-prechecks.log` 파일의 각 비호환성에 대한 세부 정보를 기록합니다. 대부분의 경우 로그 항목에는 비호환성 문제를 해결하기 위한 MySQL 설명서 링크가 포함되어 있습니다. 로그 파일 보기에 대한 자세한 내용은 데이터베이스 로그 파일 보기 및 나열 단원을 참조하십시오.

> **Note**
>
> 사전 점검의 특성으로 인해 데이터베이스의 객체를 분석합니다. 이 분석은 리소스를 소비하고 업그레이드가 완료되는 시간을 늘립니다. 사전 확인 성능 고려 사항에 대한 자세한 내용은 Aurora MySQL 사전 확인 프로세스 섹션을 참조하세요.

목차

Aurora MySQL 사전 확인 프로세스

앞서 설명한 대로 Aurora MySQL 업그레이드 프로세스에는 메이저 버전 업그레이드를 진행하기 전에 데이터베이스에서 호환성 검사(사전 확인)를 실행하는 작업이 포함됩니다.

현재 위치 업그레이드의 경우 사전 확인은 온라인 상태인 동안 라이터 DB 인스턴스에서 실행됩니다. 사전 확인에 성공하면 업그레이드가 진행됩니다. 오류가 발견되면 `upgrade-prechecks.log` 파일에 기록되고 업그레이드가 취소됩니다. 업그레이드를 다시 시도하기 전에 `upgrade-prechecks.log` 파일에 반환된 오류를 해결합니다.

스냅샷 복원 업그레이드의 경우 복원 프로세스 중에 사전 확인이 실행됩니다. 성공하면 데이터베이스가 새 Aurora MySQL 버전으로 업그레이드됩니다. 오류가 발견되면 `upgrade-prechecks.log` 파일에 기록되고 업그레이드가 취소됩니다. 업그레이드를 다시 시도하기 전에 `upgrade-prechecks.log` 파일에 반환된 오류를 해결합니다.

자세한 내용은 [Aurora MySQL 주 버전 업그레이드 실패 원인 조사](#) 및 [Aurora MySQL에 대한 사전 확인 설명 참조](#) 단원을 참조하세요.

사전 확인 상태를 모니터링하려면 DB 클러스터에서 다음 이벤트를 볼 수 있습니다.

사전 확인 상태	이벤트 메시지	작업
시작됨	업그레이드 준비 진행 중: 온라인 업그레이드 사전 확인 시작.	None
Failed	데이터베이스 클러스터를 업그레이드할 수 없는 상태: 업그레이드 사전 확인 실패. 자세한 내용은 upgrade-prechecks.log 파일을 참조하세요. 업그레이드 실패의 원인 해결에 대한 자세한 내용은 다음을 참조하세요. https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Upgrading.Troubleshooting.html	`upgrade-prechecks.log` 에 오류가 있는지 검토합니다. 오류를 수정합니다. 업그레이드를 다시 시도합니다.
성공	업그레이드 준비 진행 중: 온라인 업그레이드 사전 확인 완료.	오류가 반환되지 않고 사전 확인이 성공했습니다. `upgrade-prechecks.log` 에 경고 및 알림이 있는지 검토합니다.

이벤트 보기에 대한 자세한 내용은 Amazon RDS 이벤트 보기 섹션을 참조하세요.

Aurora MySQL의 사전 확인 로그 형식

업그레이드 호환성 확인(사전 확인)이 완료되면 `upgrade-prechecks.log` 파일을 검토할 수 있습니다. 로그 파일에는 각 사전 확인에 대한 결과, 영향을 받는 객체 및 수정 정보가 포함됩니다.

오류로 인해 업그레이드가 차단되었습니다. 업그레이드를 다시 시도하기 전에 문제를 해결해야 합니다.

경고 및 알림은 덜 중요하지만 애플리케이션 워크로드와의 호환성 문제가 없는지 주의 깊게 검토하는 것이 좋습니다. 식별된 문제를 이른 시일 내에 해결합니다.

로그 파일의 형식은 다음과 같습니다.

- `targetVersion` – Aurora MySQL 업그레이드의 MySQL 호환 버전입니다.
- `auroraServerVersion` – 사전 확인이 실행된 Aurora MySQL 버전입니다.
- `auroraTargetVersion` - 업그레이드하려는 Aurora MySQL 버전입니다.
- `checksPerformed` - 수행된 사전 확인 목록을 포함합니다.
- `id` - 실행 중인 사전 확인의 이름입니다.
- `title` – 실행 중인 사전 확인에 대한 설명입니다.
- `status` - 사전 확인이 성공했는지 실패했는지는 표시하지 않지만 다음과 같은 사전 확인 쿼리의 상태를 보여 줍니다.
  - `OK` – 사전 확인 쿼리가 성공적으로 실행되고 완료되었습니다.
  - `ERROR` – 사전 확인 쿼리를 실행하지 못했습니다. 이는 리소스 제약, 예기치 않은 인스턴스 재시작 또는 호환성 사전 확인 쿼리 중단과 같은 문제로 인해 발생할 수 있습니다.

    자세한 내용은 이 예제를 참조하세요.
- `description` - 비호환성에 대한 일반적인 설명 및 문제 해결 방법입니다.
- `documentationLink` - 해당하는 경우 관련 Aurora MySQL 또는 MySQL 설명서 링크가 여기에 표시됩니다. 자세한 내용은 Aurora MySQL에 대한 사전 확인 설명 참조 단원을 참조하십시오.
- `detectedProblems` - 사전 확인에서 오류, 경고 또는 알림이 반환되면 비호환성 및 해당하는 경우 비호환 객체에 대한 세부 정보가 표시됩니다.
  - `level` - 사전 확인에서 감지한 비호환성 수준입니다. 유효한 수준은 다음과 같습니다.
    - `Error` – 비호환성을 해결할 때까지 업그레이드를 진행할 수 없습니다.

    - Warning - 업그레이드를 진행할 수 있지만 더 이상 사용되지 않는 객체, 구문 또는 구성이 감지되었습니다. 경고를 주의 깊게 검토하고 향후 릴리스에서 문제가 발생하지 않도록 조만간 해결하세요.
    - Notice - 업그레이드를 진행할 수 있지만 더 이상 사용되지 않는 객체, 구문 또는 구성이 감지되었습니다. 알림을 주의 깊게 검토하고 향후 릴리스에서 문제가 발생하지 않도록 조만간 해결하세요.
  - dbObject - 비호환성이 감지된 데이터베이스 객체의 이름입니다.
  - description - 비호환성에 대한 세부 설명 및 문제 해결 방법입니다.
- errorCount – 감지된 비호환성 오류 수입니다. 이렇게 하면 업그레이드가 차단됩니다.
- warningCount – 감지된 비호환성 경고 수입니다. 업그레이드는 차단되지 않지만 향후 릴리스에서 문제를 방지하기 위해 이른 시일 내에 해결하세요.
- noticeCount – 감지된 비호환성 알림 수입니다. 업그레이드는 차단되지 않지만 향후 릴리스에서 문제를 방지하기 위해 이른 시일 내에 해결하세요.
- Summary – 사전 확인 호환성 오류, 경고 및 알림 수에 대한 요약입니다.

Aurora MySQL에 대한 사전 확인 로그 출력 예제

다음 예제에서는 표시될 수 있는 사전 확인 로그 출력을 보여줍니다. 실행 중인 사전 검사에 대한 자세한 내용은 Aurora MySQL에 대한 사전 확인 설명 참조 섹션을 참조하세요.

사전 확인 상태 정상, 비호환성이 감지되지 않음.

사전 확인 쿼리가 성공적으로 완료됨. 비호환성이 감지되지 않음.

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinytext",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny text columns",
  "status": "OK",
  "detectedProblems": []
},
```

사전 확인 상태 정상, 오류 감지됨

사전 확인 쿼리가 성공적으로 완료됨. 한 가지 오류가 감지됨.

```
{
```

```
 "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
 "title": "Check for geometry columns on prefix indexes",
 "status": "OK",
 "description": "Consider dropping the prefix indexes of geometry columns and
restart the upgrade.",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test25.sbtest1",
       "description": "Table `test25`.`sbtest1` has an index `idx_t1` on geometry
column/s. Mysql 8.0 does not support this type of index on a geometry column
https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run 'DROP
INDEX `idx_t1` ON `test25`.`sbtest1`;"
     },
}
```

사전 확인 상태 정상, 경고 감지됨

사전 확인에 성공하거나 실패하면 경고를 반환할 수 있음.

여기서 사전 확인 쿼리가 성공적으로 완료됨. 두 개의 경고가 감지됨.

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
in SQL_MODE by default. These modes should be used with strict mode as they will be
merged with strict mode in a future release. If you do not include these modes in
your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
contain zeros. It is strongly advised to replace zero values with valid ones, as
they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
```

```
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
    ]
}
```

사전 확인 상태 오류, 비호환성이 보고되지 않음

오류로 인해 사전 확인 쿼리가 실패했으므로 비호환성을 확인할 수 없음.

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "ERROR",
  "description": "Can't connect to MySQL server on 'localhost:3306' (111) at
 13/08/2024 12:22:20 UTC. This failure can occur due to low memory available on the
 instance for executing upgrade prechecks. Please check 'FreeableMemory' Cloudwatch
 metric to verify the available memory on the instance while executing prechecks. If
 instance ran out of memory, we recommend to retry the upgrade on a higher instance
 class."
}
```

이 오류는 예기치 않은 인스턴스 재시작 또는 실행 중에 데이터베이스에서 호환성 사전 확인 쿼리가 중단되어 발생할 수 있습니다. 예를 들어 더 작은 DB 인스턴스 클래스에서는 인스턴스에서 사용 가능한 메모리가 부족할 때 이러한 현상이 발생할 수 있습니다.

`FreeableMemory` Amazon CloudWatch 지표를 사용하여 사전 확인을 실행하는 동안 인스턴스에서 사용 가능한 메모리를 확인할 수 있습니다. 인스턴스의 메모리가 부족하면 더 큰 DB 인스턴스 클래스에서 업그레이드를 다시 시도하는 것이 좋습니다. 경우에 따라 블루/그린 배포를 사용할 수 있습니다. 이를 통해 프로덕션 워크로드와 관계없이 '그린' DB 클러스터에서 사전 확인 및 업그레이드를 실행할 수 있으며, 시스템 리소스도 사용합니다.

자세한 내용은 Aurora MySQL 데이터베이스의 메모리 사용량 문제 해결 단원을 참조하십시오.

사전 확인 요약, 오류 1개 및 경고 3개 감지됨

호환성 사전 확인에는 소스 및 대상 Aurora MySQL 버전에 대한 정보와 사전 확인 출력 끝부분의 오류, 경고 및 알림 수 요약도 포함됩니다.

예를 들어 다음 출력은 Aurora MySQL 2.11.6에서 Aurora MySQL 3.07.1로 업그레이드하려고 시도했음을 보여줍니다. 업그레이드에서 오류 1개, 경고 3개가 반환되었으며 알림은 반환되지 않았습니

다. 오류가 반환되면 업그레이드를 진행할 수 없으므로 [routineSyntaxCheck](#) 호환성 문제를 해결하고 업그레이드를 다시 시도해야 합니다.

```
{
  "serverAddress": "/tmp%2Fmysql.sock",
  "serverVersion": "5.7.12 - MySQL Community Server (GPL)",
  "targetVersion": "8.0.36",
  "auroraServerVersion": "2.11.6",
  "auroraTargetVersion": "3.07.1",
  "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
  "checksPerformed": [{
      ... output for each individual precheck ...
      .
      .
      {
        "id": "oldTemporalCheck",
        "title": "Usage of old temporal type",
        "status": "OK",
          "detectedProblems": []
      },
      {
        "id": "routinesSyntaxCheck",
        "title": "MySQL 8.0 syntax check for routine-like objects",
        "status": "OK",
        "description": "The following objects did not pass a syntax check with
 the latest MySQL 8.0 grammar. A common reason is that they reference names that
 conflict with new reserved keywords. You must update these routine definitions and
 `quote` any such references before upgrading.",
        "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
        "detectedProblems": [{
            "level": "Error",
            "dbObject": "test.select_res_word",
            "description": "at line 2,18: unexpected token 'except'"
        }]
      },
      .
      .
      .
      {
        "id": "zeroDatesCheck",
        "title": "Zero Date, Datetime, and Timestamp values",
        "status": "OK",
```

```
        "description": "Warning: By default zero date/datetime/timestamp values
 are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are
 included in SQL_MODE by default. These modes should be used with strict mode as
 they will be merged with strict mode in a future release. If you do not include
 these modes in your SQL_MODE setting, you are able to insert date/datetime/
timestamp values that contain zeros. It is strongly advised to replace zero values
 with valid ones, as they may not work correctly in the future.",
        "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
        "detectedProblems": [{
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
            },
            {
            "level": "Warning",
            "dbObject": "session.sql_mode",
            "description": " of 8 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
            }
          ]
       },
       .
       .
       .
  }],
  "errorCount": 1,
  "warningCount": 3,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
}
```


Aurora MySQL의 사전 확인 성능

호환성 사전 검사는 업그레이드를 위해 DB 인스턴스를 오프라인으로 전환하기 전에 실행되므로, 일반적인 상황에서는 실행 중에 DB 인스턴스 가동 중지가 발생하지 않습니다. 그러나 라이터 DB 인스턴스에서 실행되는 애플리케이션 워크로드에 영향을 미칠 수 있습니다. 사전 확인은 information_schema 테이블을 통해 데이터 딕셔너리에 액세스하며, 데이터베이스 객체가 많은 경우 속도가 느릴 수 있습니다. 다음 요인을 고려하세요.

- 사전 확인 시간은 테이블, 열, 루틴 및 제약 조건과 같은 데이터베이스 객체 수에 따라 달라집니다. 객체 수가 많은 DB 클러스터를 실행하는 데 시간이 더 걸릴 수 있습니다.

  예를 들어 removedFunctionsCheck는 더 오래 걸리고 저장된 객체 수에 따라 더 많은 리소스를 사용할 수 있습니다.
- 현재 위치 업그레이드의 경우 더 큰 DB 인스턴스 클래스(예: db.r5.24xlarge 또는 db.r6g.16xlarge)를 사용하면 더 많은 CPU를 사용하여 업그레이드가 더 빠르게 완료될 수 있습니다. 업그레이드 후 크기를 줄일 수 있습니다.
- 여러 데이터베이스의 `information_schema`에 대한 쿼리는 느릴 수 있습니다. 특히 많은 객체와 더 작은 DB 인스턴스의 경우 더욱 그렇습니다. 이러한 경우 업그레이드에 복제, 스냅샷 복원 또는 블루/그린 배포를 사용하는 것이 좋습니다.
- 사전 확인 리소스 사용량(CPU, 메모리)은 객체가 많을수록 증가할 수 있으므로 더 작은 DB 인스턴스에서 실행 시간이 길어집니다. 이러한 경우 업그레이드에 복제, 스냅샷 복원 또는 블루/그린 배포를 사용하여 테스트하는 것이 좋습니다.

  리소스 부족으로 인해 사전 확인이 실패하는 경우 상태 출력을 사용하여 사전 확인 로그에서 이를 감지할 수 있습니다.

  ```
  "status": "ERROR",
  ```

자세한 내용은 Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식 및 Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획 단원을 참조하세요.

커뮤니티 MySQL 업그레이드 사전 확인 요약

MySQL 버전 5.7 및 8.0 간의 일반적인 비호환 목록은 다음과 같습니다.

- MySQL 8.0에 지원되지 않는 기능을 MySQL 5.7 호환 DB 클러스터에서 사용해서는 안 됩니다.

  자세한 내용은 MySQL 설명서의 MySQL 8.0에서 제거된 기능을 참조하십시오.
- 키워드 또는 예약된 단어 위반이 없어야 합니다. 이전에 예약되지 않은 일부 키워드는 MySQL 8.0에서 예약할 수 있습니다.

  자세한 내용은 MySQL 설명서의 키워드 및 예약어를 참조하십시오.
- 유니코드 지원을 개선하기 위해 `utf8mb3` charset을 사용하는 객체가 `utf8mb4` charset을 사용하도록 변환하는 것을 고려하십시오. `utf8mb3` 문자 집합은 사용되지 않습니다. 또한, 현재 `utf8mb4`은 `utf8` charset의 별칭이므로 `utf8` 대신 문자 집합 참조를 위한 `utf8mb3` 사용을 고려하십시오.

자세한 내용은 MySQL 설명서의 utf8mb3 문자 집합(3바이트 UTF-8 유니코드 인코딩)을 참조하십시오.

- 기본값이 아닌 행 형식을 가진 InnoDB 테이블은 없어야 합니다.
- `ZEROFILL` 또는 `display` 길이 유형 속성이 없어야 합니다.
- 기본 파티셔닝 지원이 없는 스토리지 엔진을 사용하는 분할된 테이블이 없어야 합니다.
- MySQL 5.7 `mysql` 시스템 데이터베이스에는 MySQL 8.0 데이터 딕셔너리가 사용하는 테이블과 동일한 이름의 테이블이 없어야 합니다.
- 사용되지 않는 데이터 형식이나 함수를 사용하는 테이블이 없어야 합니다.
- 64자를 초과하는 외래 키 제약 조건 이름이 없어야 합니다.
- `sql_mode` 시스템 변수 설정에 정의된 사용되지 않는 SQL 모드가 없어야 합니다.
- 255자 길이를 초과하는 개별 `ENUM` 또는 `SET` 열 요소가 있는 테이블이나 저장 프로시저가 없어야 합니다.
- 공유 InnoDB 테이블스페이스에 있는 테이블 파티션이 없어야 합니다.
- 테이블스페이스 데이터 파일 경로에는 순환 참조가 없어야 합니다.
- `GROUP BY` 절에서 `ASC` 또는 `DESC` 한정자를 사용하는 쿼리 및 저장 프로그램 정의가 없어야 합니다.
- 제거된 시스템 변수가 없어야 하며, 시스템 변수는 MySQL 8.0의 새 기본값을 사용해야 합니다.
- 날짜, 날짜/시간 또는 타임스탬프 값이 `0`(영)이면 안 됩니다.
- 파일 제거 또는 손상으로 인한 스키마 불일치가 없어야 합니다.
- `FTS` 문자열을 포함하는 테이블 이름이 없어야 합니다.
- 다른 엔진에 속하는 InnoDB 테이블이 없어야 합니다.
- MySQL 5.7에 유효하지 않은 테이블 또는 스키마 이름이 없어야 합니다.

실행 중인 사전 검사에 대한 자세한 내용은 Aurora MySQL에 대한 사전 확인 설명 참조 섹션을 참조하세요.

MySQL 8.0으로 업그레이드하는 방법에 대한 자세한 내용은 MySQL 설명서에서 MySQL 업그레이드를 참조하세요. MySQL 8.0의 변경 사항에 대한 일반적인 설명은 MySQL 설명서의 MySQL 8.0의 새로운 기능을 참조하세요.

Aurora MySQL 업그레이드 사전 확인

Aurora MySQL에는 버전 2에서 버전 3으로 업그레이드하는 데 필요한 다음과 같은 고유한 요구 사항이 있습니다.

- 뷰, 루틴, 트리거 및 이벤트에는 SQL_CACHE, SQL_NO_CACHE, QUERY_CACHE와 같이 더 이상 사용되지 않는 SQL 구문이 없어야 합니다.
- FTS 인덱스가 없는 테이블에는 FTS_DOC_ID 열이 없어야 합니다.
- InnoDB 데이터 사전과 실제 테이블 정의 간에 열 정의 불일치가 없어야 합니다.
- lower_case_table_names 파라미터를 1로 설정하는 경우 모든 데이터베이스 및 테이블 이름은 소문자여야 합니다.
- 이벤트 및 트리거에는 누락되었거나 빈 definer 또는 잘못된 생성 컨텍스트가 없어야 합니다.
- 데이터베이스의 모든 트리거 이름은 고유해야 합니다.
- Aurora MySQL 버전 3에서는 DDL 복구 및 빠른 DDL이 지원되지 않습니다. 데이터베이스에는 이러한 기능과 관련된 아티팩트가 없어야 합니다.
- REDUNDANT 또는 COMPACT 행 형식의 테이블은 767바이트보다 큰 인덱스를 가질 수 없습니다.
- tiny 텍스트 열에 정의된 인덱스의 접두사 길이는 255바이트를 초과할 수 없습니다. utf8mb4 문자 집합의 경우 지원되는 접두사 길이가 63자로 제한됩니다.

  MySQL 5.7에서는 innodb_large_prefix 파라미터를 사용하여 더 큰 접두사 길이를 허용했습니다. 이 파라미터는 MySQL 8.0에서 더 이상 사용되지 않습니다.
- mysql.host 테이블에 InnoDB 메타데이터 불일치가 없어야 합니다.
- 시스템 테이블에 열 데이터 유형 불일치가 없어야 합니다.
- prepared 상태의 XA 트랜잭션이 없어야 합니다.
- 뷰의 열 이름은 64자를 초과할 수 없습니다.
- 저장 프로시저의 특수 문자 불일치가 없어야 합니다.
- 테이블에 데이터 파일 경로 불일치가 없어야 합니다.

실행 중인 사전 검사에 대한 자세한 내용은 Aurora MySQL에 대한 사전 확인 설명 참조 섹션을 참조하세요.

Aurora MySQL에 대한 사전 확인 설명 참조

Aurora MySQL 업그레이드 사전 확인은 여기에 자세히 설명되어 있습니다.

목차

- [경고]
  - [경고를 보고하는 MySQL 사전 확인]
  - [경고를 보고하는 Aurora MySQL 사전 확인]
- [고지 사항]
- [오류, 경고 또는 알림]

오류

다음 사전 확인은 사전 확인이 실패하고 업그레이드를 진행할 수 없을 때 오류를 생성합니다.

주제

- [오류를 보고하는 MySQL 사전 확인]
- [오류를 보고하는 Aurora MySQL 사전 확인]

오류를 보고하는 MySQL 사전 확인

다음 사전 확인은 Community MySQL에서 가져온 것입니다.

- checkTableMysqlSchema
- circularDirectoryCheck
- columnsWhichCannotHaveDefaultsCheck
- depreciatedSyntaxCheck
- engineMixupCheck
- enumSetElementLengthCheck
- foreignKeyLengthCheck
- getDuplicateTriggers
- getEventsWithNullDefiner
- getMismatchedMetadata
- getTriggersWithNullDefiner
- getValueOfVariablelower_case_table_names
- groupByAscSyntaxCheck
- mysqlEmptyDotTableSyntaxCheck
- mysqlIndexTooLargeCheck

- mysqlInvalid57NamesCheck
- mysqlOrphanedRoutinesCheck
- mysqlSchemaCheck
- nonNativePartitioningCheck
- oldTemporalCheck
- partitionedTablesInSharedTablespaceCheck
- removedFunctionsCheck
- routineSyntaxCheck
- schemaInconsistencyCheck

checkTableMysqlSchema

사전 확인 수준: 오류

`mysql` 스키마에 대해 `check table x for upgrade` 명령에서 보고되는 문제

Aurora MySQL 버전 3으로 업그레이드를 시작하기 전에 `check table for upgrade`는 DB 인스턴스의 `mysql` 스키마에 있는 각 테이블에서 실행됩니다. `check table for upgrade` 명령은 최신 버전의 MySQL로 업그레이드하는 동안 발생할 수 있는 잠재적 문제가 있는지 테이블을 검사합니다. 업그레이드를 시도하기 전에 이 명령을 실행하면 비호환성을 미리 식별하고 해결하는 데 도움이 되므로 실제 업그레이드 프로세스가 더 원활해집니다.

이 명령은 각 테이블에 대해 다음과 같은 다양한 검사를 수행합니다.

- 테이블 구조 및 메타데이터가 대상 MySQL 버전과 호환되는지 확인
- 더 이상 사용되지 않거나 제거된 기능이 테이블에서 사용되고 있는지 확인
- 데이터 손실 없이 테이블을 올바르게 업그레이드할 수 있는지 확인

자세한 내용은 MySQL 설명서의 CHECK TABLE 문을 참조하세요.

출력 예제:

```
{
  "id": "checkTableMysqlSchema",
  "title": "Issues reported by 'check table x for upgrade' command for mysql
 schema.",
  "status": "OK",
```

```
  "detectedProblems": []
}
```

이 사전 확인의 출력은 `check table for upgrade`가 여러 검사를 수행하기 때문에 발생한 오류와 발생한 시간에 따라 달라집니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

circularDirectoryCheck

사전 확인 수준: 오류

테이블스페이스 데이터 파일 경로의 원형 디렉터리 참조

MySQL 8.0.17부터 이 `CREATE TABLESPACE ... ADD DATAFILE` 절에서는 더 이상 원형 디렉터리 참조를 허용하지 않습니다. 업그레이드 문제를 방지하려면 Aurora MySQL 버전 3으로 업그레이드하기 전에 테이블스페이스 데이터 파일 경로에서 원형 디렉터리 참조를 제거합니다.

출력 예제:

```
{
  "id": "circularDirectory",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "description": "Error: Following tablespaces contain circular directory references
 (e.g. the reference '/../') in data file paths which as of MySQL 8.0.17 are not
 permitted by the CREATE TABLESPACE ... ADD DATAFILE clause. An exception to the
 restriction exists on Linux, where a circular directory reference is permitted if
 the preceding directory is a symbolic link. To avoid upgrade issues, remove any
 circular directory references from tablespace data file paths before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-innodb-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "ts2",
        "description": "circular reference in datafile path: '/home/ec2-user/dbdata/
mysql_5_7_44/../ts2.ibd'",
        "dbObjectType": "Tablespace"
      }
  ]
```

```
}
```

이 오류가 발생하면 file-per-table tablespace를 사용하여 테이블을 다시 빌드합니다. 모든 테이블스페이스 및 테이블 정의에 기본 파일 경로를 사용합니다.

> **Note**
>
> Aurora MySQL은 일반 테이블스페이스 또는 CREATE TABLESPACE 명령을 지원하지 않습니다.
> 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 온라인 DDL 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해하세요.

다시 빌드한 후 사전 확인이 통과되어 업그레이드를 진행할 수 있습니다.

```
{
  "id": "circularDirectoryCheck",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "detectedProblems": []
},
```

columnsWhichCannotHaveDefaultsCheck

사전 확인 수준: 오류

기본값을 가질 수 없는 열

MySQL 8.0.13 이전에는 BLOB, TEXT, GEOMETRY 및 JSON 열에 기본값이 있을 수 없습니다. Aurora MySQL 버전 3으로 업그레이드하기 전에 이러한 열에서 기본 절을 제거합니다. 이러한 데이터 유형의 기본 처리 변경 사항에 대한 자세한 내용은 MySQL 설명서의 데이터 유형 기본값을 참조하세요.

출력 예제:

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "description": "Error: The following columns are defined as either BLOB, TEXT,
 GEOMETRY or JSON and have a default value set. These data types cannot have default
```

```
  values in MySQL versions prior to 8.0.13, while starting with 8.0.13, the default
  value must be specified as an expression. In order to fix this issue, please use
  the ALTER TABLE ... ALTER COLUMN ... DROP DEFAULT statement.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/data-type-
defaults.html#data-type-defaults-explicit",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test_blob_default.geo_col",
        "description": "geometry"
      }
  ]
},
```

사전 확인은 `test.test_blob_default` 테이블의 `geo_col` 열이 기본값이 지정된 BLOB, TEXT, GEOMETRY 또는 JSON 데이터 유형을 사용하므로 오류를 반환합니다.

테이블 정의를 보면 `geo_col` 열이 `geo_col geometry NOT NULL default ''`로 정의되어 있음을 알 수 있습니다.

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

이 기본 절을 제거하여 사전 확인을 통과할 수 있도록 합니다.

> **Note**
>
> ALTER TABLE 문을 실행하거나 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 온라인 DDL 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해하세요.

```
mysql> ALTER TABLE test_blob_default modify COLUMN geo_col geometry NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table test_blob_default\G
```

```
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

사전 확인이 통과하면 업그레이드를 다시 시도할 수 있습니다.

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "detectedProblems": []
},
```

depreciatedSyntaxCheck

사전 확인 수준: 오류

정의에서 사용되지 않는 키워드 사용

MySQL 8.0에서 쿼리 캐시가 제거되었습니다. 따라서 일부 쿼리 캐시별 SQL 구문이 제거되었습니다. 데이터베이스 객체에 QUERY CACHE, SQL_CACHE 또는 SQL_NO_CACHE 키워드가 포함된 경우 사전 확인 오류가 반환됩니다. 이 문제를 해결하려면 언급된 키워드를 제거하여 이러한 객체를 다시 생성합니다.

출력 예제:

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "description": "Error: The following DB objects contain keywords like 'QUERY
 CACHE', 'SQL_CACHE', 'SQL_NO_CACHE' which are not supported in major version 8.0.
 It is recommended to drop these DB objects or rebuild without any of the above
 keywords before upgrade.",
  "detectedProblems": [
      {
"level": "Error",
"dbObject": "test.no_query_cache_check",
"description": "PROCEDURE uses depreciated words in definition"
      }
```

```
  ]
}
```

사전 확인은 `test.no_query_cache_check` 저장된 프로시저가 제거된 키워드 중 하나를 사용하고 있음을 보고합니다. 프로시저 정의를 보면 `SQL_NO_CACHE`를 사용하는 것을 알 수 있습니다.

```
mysql> show create procedure test.no_query_cache_check\G
*************************** 1. row ***************************
           Procedure: no_query_cache_check
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `no_query_cache_check`()
BEGIN
    SELECT SQL_NO_CACHE k from sbtest1 where id > 10 and id < 20 group by k asc;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

키워드를 제거합니다.

```
mysql> drop procedure test.no_query_cache_check;
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter //

mysql> CREATE DEFINER=`reinvent`@`%` PROCEDURE `no_query_cache_check`() BEGIN
 SELECT k from sbtest1 where id > 10 and id < 20 group by k asc; END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

키워드를 제거하면 사전 확인이 성공적으로 완료됩니다.

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "detectedProblems": []
}
```

engineMixupCheck

사전 확인 수준: 오류

다른 엔진에 속하는 InnoDB에서 인식하는 테이블

schemaInconsistencyCheck와 마찬가지로 이 사전 확인은 업그레이드를 진행하기 전에 MySQL의 테이블 메타데이터가 일관적인지 확인합니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

출력 예제:

```
{
  "id": "engineMixupCheck",
  "title": "Tables recognized by InnoDB that belong to a different engine",
  "status": "OK",
  "description": "Error: Following tables are recognized by InnoDB engine while the
 SQL layer believes they belong to a different engine. Such situation may happen
 when one removes InnoDB table files manually from the disk and creates e.g. a
 MyISAM table with the same name.\n\nA possible way to solve this situation is to
 e.g. in case of MyISAM table:\n\n1. Rename the MyISAM table to a temporary name
 (RENAME TABLE).\n2. Create some dummy InnoDB table (its definition does not need to
 match), then copy (copy, not move) and rename the dummy .frm and .ibd files to the
 orphan name using OS file commands.\n3. The orphan table can be then dropped (DROP
 TABLE), as well as the dummy table.\n4. Finally the MyISAM table can be renamed
 back to its original name.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.general_log_backup",
        "description": "recognized by the InnoDB engine but belongs to CSV"
      }
  ]
}
```

enumSetElementLengthCheck

사전 확인 수준: 오류

255자보다 긴 요소를 포함하는 **ENUM** 및 **SET** 열 정의

테이블 및 저장된 프로시저에는 255자 또는 1,020바이트를 초과하는 ENUM 또는 SET 열 요소가 없어야 합니다. MySQL 8.0 이전에는 최대 결합 길이가 64K였지만 8.0은 개별 요소를 255자 또는 1,020바이트(멀티바이트 지원)로 제한합니다. `enumSetElementLengthCheck`에 대한 사전 확인 실패가 발생하면 업그레이드를 다시 시도하기 전에 이러한 새 제한을 초과하는 요소를 수정합니다.

출력 예제:

```
{
  "id": "enumSetElementLengthCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "description": "Error: The following columns are defined as either ENUM or SET and
 contain at least one element longer that 255 characters. They need to be altered so
 that all elements fit into the 255 characters limit.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/string-type-
overview.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.large_set.s",
        "description": "SET contains element longer than 255 characters"
      }
  ]
},
```

`test.large_set` 테이블의 s 열에 255자보다 긴 SET 요소가 포함되어 있기 때문에 사전 확인은 오류를 보고합니다.

이 열의 SET 크기를 줄이면 사전 확인이 통과되어 업그레이드를 진행할 수 있습니다.

```
{
  "id": "enumSetElementLenghtCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "detectedProblems": []
},
```

foreignKeyLengthCheck

사전 확인 수준: 오류

64자보다 긴 외래 키 제약 조건 이름

MySQL 설명서에 설명된 대로 MySQL에서 식별자 길이는 64자로 제한됩니다. 외래 키 길이가 이 값 이상이어서 업그레이드 실패가 발생하는 문제가 확인되어 이 사전 확인이 구현되었습니다. 이 사전 확인에 오류가 발생하면 업그레이드를 다시 시도하기 전에 제약 조건을 64자 미만으로 변경하거나 이름을 변경해야 합니다.

출력 예제:

```
{
  "id": "foreignKeyLength",
  "title": "Foreign key constraint names longer than 64 characters",
  "status": "OK",
  "detectedProblems": []
}
```

getDuplicateTriggers

사전 확인 수준: 오류

데이터베이스의 모든 트리거 이름은 고유해야 합니다.

데이터 딕셔너리 구현의 변경으로 인해 MySQL 8.0은 데이터베이스 내에서 대/소문자를 구분하는 트리거를 지원하지 않습니다. 이 사전 확인은 DB 클러스터에 중복 트리거가 포함된 데이터베이스가 하나 이상 없는지 확인합니다. 자세한 내용은 MySQL 설명서의 식별자 대/소문자 구분을 참조하세요.

출력 예제:

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "description": "Error: You have one or more database containing duplicate
 triggers. Mysql 8.0 does not support case sensitive triggers within a database
 https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html. To
 upgrade to MySQL 8.0, drop the triggers with case-insensitive duplicate names and
 recreate with distinct names.",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "test",
        "description": "before_insert_product"
      },
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_PRODUCT"
      }
  ]
}
```

사전 확인은 데이터베이스 클러스터에 이름이 같지만 대/소문자가 다른 두 개의 트리거 (`test.before_insert_product` 및 `test.before_insert_PRODUCT`)가 있다는 오류를 보고합니다.

업그레이드하기 전에 트리거의 이름을 바꾸거나 삭제하고 새 이름으로 다시 생성합니다.

`test.before_insert_PRODUCT`의 이름을 `test.before_insert_product_2`로 변경하면 사전 확인이 성공합니다.

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "detectedProblems": []
}
```

getEventsWithNullDefiner

사전 확인 수준: 오류

`mysql.event`에 대한 정의자 열은 Null이거나 공백일 수 없습니다.

`DEFINER` 속성은 트리거, 저장된 프로시저 또는 이벤트와 같은 저장된 객체 정의를 소유하는 MySQL 계정을 지정합니다. 이 속성은 저장된 객체가 실행되는 보안 컨텍스트를 제어하려는 상황에서 특히 유용합니다. 저장된 객체를 생성할 때 `DEFINER`가 지정되지 않은 경우 기본값은 객체를 생성한 사용자입니다.

MySQL 8.0으로 업그레이드할 때는 MySQL 데이터 딕셔너리에서 `null` 또는 빈 정의자가 있는 저장된 객체를 가질 수 없습니다. 이러한 저장된 객체가 있는 경우 사전 확인 오류가 발생합니다. 업그레이드를 진행하려면 먼저 문제를 해결해야 합니다.

오류 메시지 예:


```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "description": "Error: Set definer column in mysql.event to a valid non-null
 definer.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.get_version",
        "description": "Set definer for event get_version in Schema test"
      }
  ]
}
```


사전 확인은 null 정의자가 있기 때문에 `test.get_version` 이벤트에 대한 오류를 반환합니다.

이를 해결하려면 이벤트 정의를 확인할 수 있습니다. 보시다시피 정의자는 `null`이거나 비어 있습니다.

```
mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+---------+
| db   | name        | definer |
+------+-------------+---------+
| test | get_version |         |
+------+-------------+---------+
1 row in set (0.00 sec)
```

유효한 정의자를 사용하여 이벤트를 삭제하거나 다시 생성합니다.

> **Note**
>
> DEFINER를 삭제하거나 재정의하기 전에 애플리케이션 및 권한 요구 사항을 주의 깊게 검토하고 확인합니다. 자세한 내용은 MySQL 설명서의 저장된 객체 액세스 제어를 참조하세요.

```
mysql> drop event test.get_version;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELIMITER ;
mysql> delimiter $$
mysql> CREATE EVENT get_version
    ->     ON SCHEDULE
    ->       EVERY 1 DAY
    ->     DO
    ->      ///DO SOMETHING //
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+------------+
| db   | name        | definer    |
+------+-------------+------------+
| test | get_version | reinvent@% |
+------+-------------+------------+
1 row in set (0.00 sec)
```

이제 사전 확인이 통과됩니다.

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "detectedProblems": []},
```

getMismatchedMetadata

사전 확인 수준: 오류

InnoDB 데이터 딕셔너리와 실제 테이블 정의 간의 열 정의 불일치

schemaInconsistencyCheck와 마찬가지로 이 사전 확인은 업그레이드를 진행하기 전에 MySQL의 테이블 메타데이터가 일관적인지 확인합니다. 이 경우 사전 확인은 열 정의가 InnoDB 데이터 딕셔너리와 MySQL 테이블 정의 간에 일치하는지 확인합니다. 불일치가 감지되면 업그레이드가 진행되지 않습니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

출력 예제:

```
{
  "id": "getMismatchedMetadata",
  "title": "Column definition mismatch between InnoDB Data Dictionary and actual
 table definition.",
  "status": "OK",
  "description": "Error: Your database has mismatched metadata. The upgrade to mysql
 8.0 will not succeed until this is fixed.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.mismatchTable",
        "description": "Table `test/mismatchTable` column names mismatch with InnoDb
 dictionary column names: iD <> id"
      }
  ]
}
```

사전 확인은 `test.mismatchTable` 테이블의 id 열에 대한 메타데이터의 불일치를 보고합니다. 특히 MySQL 메타데이터의 열 이름은 `iD`이고 InnoDB의 열 이름은 `id`입니다.

getTriggersWithNullDefiner

사전 확인 수준: 오류

`information_schema.triggers`에 대한 정의자 열은 `null`이거나 공백일 수 없습니다.

사전 확인은 데이터베이스에 `null` 또는 빈 정의자로 정의된 트리거가 없는지 확인합니다. 저장된 객체의 정의자 요구 사항에 대한 자세한 내용은 getEventsWithNullDefiner를 참조하세요.

출력 예제:

```
{
  "id": "getTriggersWithNullDefiner",
  "title": "The definer column for information_schema.triggers cannot be null or
 blank.",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.example_trigger",
        "description": "Set definer for trigger example_trigger in Schema test"
```

```
        }
    ]
}
```

`test` 스키마의 `example_trigger` 트리거에 `null` 정의자가 있으므로 사전 확인은 오류를 반환합니다. 이 문제를 해결하려면 유효한 사용자로 트리거를 다시 생성하거나 트리거를 삭제하여 정의자를 수정합니다. 자세한 내용은 getEventsWithNullDefiner의 예제를 참조하세요.

> **Note**
>
> `DEFINER`를 삭제하거나 재정의하기 전에 애플리케이션 및 권한 요구 사항을 주의 깊게 검토하고 확인합니다. 자세한 내용은 MySQL 설명서의 저장된 객체 액세스 제어를 참조하세요.

getValueOfVariablelower_case_table_names

사전 확인 수준: 오류

`lower_case_table_names` 파라미터를 **1**로 설정하는 경우 모든 데이터베이스 또는 테이블 이름은 소문자여야 합니다.

MySQL 8.0 이전에는 데이터베이스 이름, 테이블 이름 및 기타 객체가 파일 기반 메타데이터(.frm)와 같은 데이터 디렉터리의 파일에 상응했습니다. lower_case_table_names 시스템 변수를 사용하면 서버가 데이터베이스 객체의 식별자 대/소문자 구분 여부를 처리하는 방식과 이러한 메타데이터 객체의 스토리지를 제어할 수 있습니다. 재부팅 후 이미 초기화된 서버에서 이 파라미터를 변경할 수 있습니다.

그러나 MySQL 8.0에서 이 파라미터는 서버가 식별자 대/소문자 구분 여부를 처리하는 방식을 여전히 제어하지만 데이터 딕셔너리가 초기화된 후에는 변경할 수 없습니다. MySQL 8.0 데이터베이스를 업그레이드하거나 생성할 때 데이터 딕셔너리가 MySQL 에서 처음 시작될 때 `lower_case_table_names`에 대해 설정된 값은 해당 데이터베이스의 수명 동안 사용됩니다. 이 제한은 데이터베이스 객체가 파일 기반 메타데이터에서 `mysql` 스키마의 내부 InnoDB 테이블로 마이그레이션되는 원자성 데이터 딕셔너리 구현의 일부로 적용되었습니다.

자세한 내용은 MySQL 설명서의 데이터 딕셔너리 변경 사항을 참조하세요.

파일 기반 메타데이터를 새 원자성 데이터 딕셔너리로 업데이트할 때 업그레이드 중에 문제가 발생하지 않도록 하기 위해 이 사전 확인은 `lower_case_table_names = 1`일 때 모든 테이블이 디

스크에 소문자로 저장되는지 확인합니다. 그러지 않으면 사전 확인 오류가 반환되며 업그레이드를 진행하기 전에 메타데이터를 수정해야 합니다.

출력 예제:

```
{
  "id": "getValueOfVariablelower_case_table_names",
  "title": "MySQL pre-checks that all database or table names are lowercase when the
 lower_case_table_names parameter is set to 1.",
  "status": "OK",
  "description": "Error: You have one or more databases or tables with uppercase
 letters in the names, but the lower_case_table_names parameter is set to 1. To
 upgrade to MySQL 8.0, either change all database or table names to lowercase, or
 set the parameter to 0.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.TEST",
        "description": "Table test.TEST contains one or more capital letters in name
 while lower_case_table_names = 1"
      }
  ]
}
```

`test.TEST` 테이블에 대문자가 포함되어 있지만 `lower_case_table_names`가 1로 설정되어 있기 때문에 오류가 반환됩니다.

이 문제를 해결하려면 업그레이드를 시작하기 전에 소문자를 사용하도록 테이블의 이름을 바꾸거나 DB 클러스터의 `lower_case_table_names` 파라미터를 수정할 수 있습니다.

> **Note**
>
> MySQL의 [대/소문자 구분 여부]에 대한 설명서와 이러한 변경 사항이 애플리케이션에 미치는 영향을 주의 깊게 테스트하고 검토합니다.
> 또한 MySQL 8.0에서 [lower_case_table_names]를 다르게 처리하는 방법에 대한 MySQL 8.0 설명서를 검토합니다.

groupByAscSyntaxCheck

사전 확인 수준: 오류

제거된 **GROUP BY ASC/DESC** 구문 사용

MySQL 8.0.13부터 GROUP BY 절의 더 이상 사용되지 않는 ASC 또는 DESC 구문이 제거되었습니다. GROUP BY 정렬에 의존하는 쿼리는 이제 다른 결과를 생성할 수 있습니다. 특정 정렬 순서를 가져오려면 대신 ORDER BY 절을 사용합니다. 이 구문을 사용하는 데이터베이스에 객체가 있는 경우 업그레이드를 다시 시도하기 전에 ORDER BY 절을 사용하여 객체를 다시 생성해야 합니다. 자세한 내용은 MySQL 설명서에서 [SQL 변경 사항](https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-13.html#mysqld-8-0-13-sql-syntax)을 참조하세요.

출력 예제:

```
{
  "id": "groupbyAscSyntaxCheck",
  "title": "Usage of removed GROUP BY ASC/DESC syntax",
  "status": "OK",
  "description": "Error: The following DB objects use removed GROUP BY ASC/DESC
 syntax. They need to be altered so that ASC/DESC keyword is removed from GROUP BY
 clause and placed in appropriate ORDER BY clause.",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-sql-syntax",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.groupbyasc",
        "description": "PROCEDURE uses removed GROUP BY ASC syntax",
        "dbObjectType": "Routine"
      }
  ]
}
```

mysqlEmptyDotTableSyntaxCheck

사전 확인 수준: 오류

루틴에 더 이상 사용되지 않는 **.<table>** 구문이 있는지 확인합니다.

MySQL 8.0에서 루틴에는 더 이상 사용되지 않는 식별자 구문(`\".<table>\"`)이 포함될 수 없습니다. 저장된 루틴 또는 트리거에 이러한 식별자가 포함된 경우 업그레이드가 실패합니다. 예를 들어, 다음 `.dot_table` 참조는 더 이상 허용되지 않습니다.

```
mysql> show create procedure incorrect_procedure\G
*************************** 1. row ***************************
           Procedure: incorrect_procedure
            sql_mode:
```

```
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `incorrect_procedure`()
BEGIN delete FROM .dot_table; select * from .dot_table where 1=1; END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

올바른 식별자 구문과 이스케이핑을 사용하도록 루틴과 트리거를 다시 생성한 후 사전 확인이 통과되고 업그레이드를 진행할 수 있습니다. 자세한 내용은 MySQL 설명서의 스키마 객체 이름을 참조하세요.

출력 예제:

```
{
  "id": "mysqlEmptyDotTableSyntaxCheck",
  "title": "Check for deprecated '.<table>' syntax used in routines.",
  "status": "OK",
  "description": "Error: The following routines contain identifiers in deprecated
 identifier syntax (\".<table>\"), and should be corrected before upgrade:\n",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.incorrect_procedure",
        "description": " routine body contains deprecated identifiers."
      }
  ]
}
```

사전 확인은 더 이상 사용되지 않는 구문이 포함되어 있으므로 `test` 데이터베이스의 `incorrect_procedure` 루틴에 대한 오류를 반환합니다.

루틴을 수정하면 사전 확인이 성공하고 업그레이드를 다시 시도할 수 있습니다.

mysqlIndexTooLargeCheck

사전 확인 수준: 오류

5.7보다 높은 MySQL 버전에서 작업하기에 너무 큰 인덱스가 있는지 확인합니다.

컴팩트하거나 중복된 행 형식의 경우 접두사가 767바이트보다 큰 인덱스를 생성할 수 없습니다. 그러나 MySQL 버전 5.7.35 이전에는 이 작업이 가능했습니다. 자세한 내용은 MySQL 5.7.35 릴리스 정보를 참조하세요.

이 버그의 영향을 받은 인덱스는 MySQL 8.0으로 업그레이드한 후 액세스할 수 없게 됩니다. 이 사전 확인은 업그레이드를 진행할 수 있게 되기 전에 다시 빌드해야 하는 잘못된 인덱스를 식별합니다.

```
{
 "id": "mysqlIndexTooLargeCheck",
 "title": "Check for indexes that are too large to work on higher versions of MySQL
Server than 5.7",
 "status": "OK",
 "description": "Error: The following indexes ware made too large for their format
in an older version of MySQL (older than 5.7.34). Normally those indexes within
tables with compact or redundant row formats shouldn't be larger than 767 bytes. To
fix this problem those indexes should be dropped before upgrading or those tables
will be inaccessible.",
 "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_large_idx",
        "description": "IDX_2"
      }
  ]
}
```

`test.table_with_large_idx` 테이블에는 767바이트보다 크고 컴팩트하거나 중복된 행 형식을 사용하는 테이블의 인덱스가 포함되어 있기 때문에 사전 확인은 오류를 반환합니다. MySQL 8.0으로 업그레이드한 후에는 이러한 테이블에 액세스할 수 없게 됩니다. 업그레이드를 진행하기 전에 다음 중 하나를 수행합니다.

- 사전 확인에 언급된 인덱스를 삭제합니다.
- 사전 확인에 언급된 인덱스를 추가합니다.
- 테이블에서 사용하는 행 형식을 변경합니다.

여기서 사전 확인 실패를 해결하기 위해 테이블을 다시 빌드합니다. 테이블을 다시 빌드하기 전에 innodb_file_format이 `Barracuda`로 설정되어 있고 innodb_default_row_format이 `dynamic`으로 설정되어 있는지 확인합니다. 이는 MySQL 5.7의 기본값입니다. 자세한 내용은 MySQL 설명서의 InnoDB 행 형식 및 InnoDB 파일 형식 관리를 참조하세요.

> **Note**
>
> 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 [온라인 DDL 작업]을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해하세요.

```
mysql > select @@innodb_file_format,@@innodb_default_row_format;
+----------------------+-----------------------------+
| @@innodb_file_format | @@innodb_default_row_format |
+----------------------+-----------------------------+
| Barracuda            | dynamic                     |
+----------------------+-----------------------------+
1 row in set (0.00 sec)

mysql> optimize table table_with_large_idx;
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| Table                     | Op       | Msg_type | Msg_text
                                   |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| test.table_with_large_idx | optimize | note     | Table does not support optimize,
 doing recreate + analyze instead |
| test.table_with_large_idx | optimize | status   | OK
                                   |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.02 sec)

# Verify FILE_FORMAT and ROW_FORMAT
mysql>  select * from information_schema.innodb_sys_tables where name like 'test/
table_with_large_idx';
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
| TABLE_ID | NAME                      | FLAG | N_COLS | SPACE | FILE_FORMAT |
 ROW_FORMAT | ZIP_PAGE_SIZE | SPACE_TYPE |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
|       43 | test/table_with_large_idx |   33 |      4 |    26 | Barracuda   |
 Dynamic    |             0 | Single     |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
```

```
1 row in set (0.00 sec)
```

테이블을 다시 빌드한 후 사전 확인이 통과되고 업그레이드를 진행할 수 있습니다.

```
{
  "id": "mysqlIndexTooLargeCheck",
  "title": "Check for indexes that are too large to work on higher versions of MySQL
 Server than 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlInvalid57NamesCheck

사전 확인 수준: 오류

MySQL 5.7에 사용된 테이블 및 스키마 이름이 유효하지 않은지 확인

MySQL 8.0에서 새 데이터 딕셔너리로 마이그레이션할 때 데이터베이스 인스턴스에 `#mysql50#` 접두사가 붙은 스키마 또는 테이블이 포함될 수 없습니다. 이러한 객체가 있으면 업그레이드가 실패합니다. 이 문제를 해결하려면 반환된 스키마 및 테이블에 대해 [mysqlcheck](https://dev.mysql.com/doc/refman/5.7/en/mysqlcheck.html)를 실행합니다.

> **Note**
>
> [--fix-db-names](https://dev.mysql.com/doc/refman/5.7/en/mysqlcheck.html#option_mysqlcheck_fix-db-names) 및 [--fix-table-names](https://dev.mysql.com/doc/refman/5.7/en/mysqlcheck.html#option_mysqlcheck_fix-table-names)가 [MySQL 8.0](https://dev.mysql.com/doc/refman/8.0/en/mysqlcheck.html)에서 제거되었으므로 `mysqlcheck`의 [MySQL 5.7 버전](https://dev.mysql.com/doc/refman/5.7/en/mysqlcheck.html)을 사용해야 합니다.

출력 예제:

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "description": "The following tables and/or schemas have invalid names. In order
 to fix them use the mysqlcheck utility as follows:\n\n  $ mysqlcheck --check-
upgrade --all-databases\n  $ mysqlcheck --fix-db-names --fix-table-names --all-
databases\n\nOR via mysql client, for eg:\n\n  ALTER DATABASE `#mysql50#lost+found`
 UPGRADE DATA DIRECTORY NAME;",
  "documentationLink": "https://dev.mysql.com/doc/refman/5.7/en/identifier-
mapping.html https://dev.mysql.com/doc/refman/5.7/en/alter-database.html https://
dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html#mysql-nutshell-removals",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "#mysql50#fix_db_names",
        "description": "Schema name"
      }
  ]
}
```

사전 확인은 `#mysql50#fix_db_names` 스키마의 이름이 유효하지 않다고 보고합니다.

스키마 이름을 수정한 후 사전 확인이 통과되어 업그레이드를 진행할 수 있습니다.

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlOrphanedRoutinesCheck

사전 확인 수준: 오류

5.7에서 분리된 루틴 확인

MySQL 8.0의 새 데이터 딕셔너리로 마이그레이션할 때 스키마가 더 이상 존재하지 않는 데이터베이스에 저장된 프로시저가 있는 경우 업그레이드가 실패합니다. 이 사전 확인은 DB 인스턴스의 저장된 프로시저에 참조된 모든 스키마가 여전히 존재하는지 확인합니다. 업그레이드를 진행하려면 다음 저장 프로시저를 삭제합니다.

출력 예제:

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "description": "Error: The following routines have been orphaned. Schemas that
 they are referencing no longer exists.\nThey have to be cleaned up or the upgrade
 will fail.",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "dropped_db.get_version",
        "description": "is orphaned"
      }
    ]
},
```

사전 확인은 dropped_db 데이터베이스의 get_version 저장된 프로시저가 분리되었음을 보고합니다.

이 프로시저를 정리하려면 누락된 스키마를 다시 생성할 수 있습니다.

```
mysql> create database dropped_db;
Query OK, 1 row affected (0.01 sec)
```

스키마를 다시 생성한 후 프로시저를 삭제하여 업그레이드를 진행할 수 있습니다.

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlSchemaCheck

사전 확인 수준: 오류

mysql 스키마의 테이블 이름이 MySQL 8.0의 새 테이블과 충돌함

MySQL 8.0에 도입된 새로운 원자성 데이터 딕셔너리는 mysql 스키마의 내부 InnoDB 테이블 세트에 모든 메타데이터를 저장합니다. 업그레이드 중에 새 내부 데이터 딕셔너리 테이블이 mysql 스키마에 생성됩니다. 업그레이드 실패를 일으키는 이름 지정 충돌을 방지하기 위해 사전 확인은 mysql 스키마의 모든 테이블 이름을 검사하여 새 테이블 이름 중에 이미 사용 중인 것이 없는지 확인합니다. 있다면 오류가 반환되고 업그레이드를 진행할 수 없습니다.

출력 예제:

```
{
  "id": "mysqlSchema",
  "title": "Table names in the mysql schema conflicting with new tables in the
 latest MySQL.",
```

```
  "status": "OK",
  "description": "Error: The following tables in mysql schema have names that will
 conflict with the ones introduced in the latest version. They must be renamed or
 removed before upgrading (use RENAME TABLE command). This may also entail changes
 to applications that use the affected tables.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrade-before-you-
begin.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.tablespaces",
        "description": "Table name used in mysql schema.",
        "dbObjectType": "Table"
      }
  ]
}
```

mysql 스키마에 이름이 지정된 테이블 tablespaces가 있으므로 오류가 반환됩니다. MySQL 8.0의 새로운 내부 데이터 딕셔너리 테이블 이름 중 하나입니다. RENAME TABLE 명령을 사용하여 업그레이드하기 전에 이러한 테이블의 이름을 바꾸거나 제거해야 합니다.

nonNativePartitioningCheck

사전 확인 수준: 오류

기본이 아닌 파티셔닝이 있는 엔진을 사용하는 파티셔닝된 테이블

MySQL 8.0 설명서에 따르면 현재 두 개의 스토리지 엔진 InnoDB 및 NDB가 기본 파티셔닝 지원을 제공합니다. 이 중에서 Aurora MySQL 버전 3에서는 MySQL 8.0과 호환되는 InnoDB만 지원됩니다. 다른 스토리지 엔진을 사용하여 MySQL 8.0에서 파티셔닝된 테이블을 생성하려는 시도는 실패합니다. 이 사전 확인은 DB 클러스터에서 기본이 아닌 파티셔닝을 사용하는 테이블을 찾습니다. 반환되는 것이 있다면 파티셔닝을 제거하거나 스토리지 엔진을 InnoDB로 변환해야 합니다.

출력 예제:

```
{
  "id": "nonNativePartitioning",
  "title": "Partitioned tables using engines with non native partitioning",
  "status": "OK",
  "description": "Error: In the latest MySQL storage engine is responsible for
 providing its own partitioning handler, and the MySQL server no longer provides
 generic partitioning support. InnoDB and NDB are the only storage engines that
 provide a native partitioning handler that is supported in the latest MySQL. A
```

```
 partitioned table using any other storage engine must be altered—either to convert
 it to InnoDB or NDB, or to remove its partitioning—before upgrading the server,
 else it cannot be used afterwards.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-configuration-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partMyisamTable",
         "description": "MyISAM engine does not support native partitioning",
         "dbObjectType": "Table"
      }
  ]
}
```

여기서 MyISAM 테이블은 파티셔닝을 사용하며, 업그레이드를 진행하기 전에 작업이 필요합니다.

oldTemporalCheck

사전 확인 수준: 오류

이전 시간 유형 사용

'이전 시간'은 MySQL 버전 5.5 이하에서 생성된 시간 유형 열(예: `TIMESTAMP` 및 `DATETIME`)을 나타냅니다. MySQL 8.0에서는 이러한 이전 시간 데이터 유형에 대한 지원이 제거되므로 데이터베이스에 이러한 이전 시간 유형이 포함된 경우 MySQL 5.7에서 8.0으로의 현재 위치 업그레이드가 불가능합니다. 이 문제를 해결하려면 업그레이드를 진행하기 전에 이전 시간 날짜 유형이 포함된 테이블을 다시 빌드해야 합니다.

MySQL 5.7의 이전 시간 데이터 유형 사용 중단에 대한 자세한 내용은 이 블로그를 참조하세요. MySQL 8.0의 이전 시간 데이터 유형 제거에 대한 자세한 내용은 이 블로그를 참조하세요.

> **Note**
>
> 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 온라인 DDL 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해하세요.

출력 예제:

```
{
  "id": "oldTemporalCheck",
```

```
  "title": "Usage of old temporal type",
  "status": "OK",
  "description": "Error: Following table columns use a deprecated and no longer
 supported temporal disk storage format. They must be converted to the new format
 before upgrading. It can by done by rebuilding the table using 'ALTER TABLE
 <table_name> FORCE' command",
  "documentationLink": "https://dev.mysql.com/blog-archive/mysql-8-0-removing-
support-for-old-temporal-datatypes/",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.55_temporal_table.timestamp_column",
        "description": "timestamp /* 5.5 binary format */",
        "dbObjectType": "Column"
      }
  ]
},
```

더 이상 지원되지 않는 이전 임시 디스크 스토리지 형식을 사용하기 때문에 `test.55_temporal_table` 테이블의 `timestamp_column` 열에 오류가 보고됩니다.

이 문제를 해결하고 업그레이드를 진행하려면 테이블을 다시 빌드하여 이전 시간 디스크 스토리지 형식을 MySQL 5.6에 도입된 새 스토리지 형식으로 변환합니다. 자세한 내용과 사전 요구 사항은 MySQL 설명서의 3바이트와 4바이트 유니코드 문자 세트 간 변환을 참조하세요.

다음 명령을 실행하여 이 사전 확인에 언급된 테이블을 다시 빌드하면 이전 시간 유형이 소수점 이하의 자릿수까지 정밀한 초 단위의 최신 형식으로 변환됩니다.

```
ALTER TABLE ... ENGINE=InnoDB;
```

테이블을 다시 빌드하는 것에 대한 자세한 내용은 MySQL 설명서의 ALTER TABLE 문을 참조하세요.

해당 테이블을 다시 빌드하고 업그레이드를 다시 시작한 후 호환성 검사를 통과하여 업그레이드를 진행할 수 있습니다.

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

partitionedTablesInSharedTablespaceCheck

사전 확인 수준: 오류

공유 테이블스페이스에서 파티셔닝된 테이블 사용

MySQL 8.0.13부터 공유 테이블스페이스에 테이블 파티션을 적용하는 것이 더 이상 지원되지 않습니다. 업그레이드하기 전에 이러한 테이블을 공유 테이블스페이스에서 테이블당 파일 테이블스페이스로 이동합니다.

> **Note**
>
> 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 파티셔닝 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해합니다.

출력 예제:

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "description": "Error: The following tables have partitions in shared tablespaces.
 They need to be moved to file-per-table tablespace before upgrading. You can do
 this by running query like 'ALTER TABLE table_name REORGANIZE PARTITION X INTO
 (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partInnoDBTable",
        "description": "Partition p1 is in shared tablespace innodb",
        "dbObjectType": "Table"
      }
  ]
}
```

테이블 `test.partInnoDBTable`의 파티션 `p1`이 시스템 테이블스페이스에 있으므로 사전 확인이 실패합니다.

이 문제를 해결하려면 `test.partInnodbTable` 테이블을 다시 빌드하여 문제가 되는 파티션 p1을 테이블당 파일 테이블스페이스에 배치합니다.

```
mysql > ALTER TABLE partInnodbTable REORGANIZE PARTITION p1
    ->   INTO (PARTITION p1 VALUES LESS THAN ('2014-01-01')
 TABLESPACE=innodb_file_per_table);
Query OK, 0 rows affected, 1 warning (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

이렇게 하면 사전 확인이 통과됩니다.

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "detectedProblems": []
}
```

removedFunctionsCheck

사전 확인 수준: 오류

최신 MySQL 버전에서 제거된 함수 사용

MySQL 8.0에서는 여러 내장 함수가 제거되었습니다. 이 사전 확인은 이러한 함수를 사용할 수 있는 객체가 있는지 데이터베이스를 검사합니다. 발견된 경우 오류가 반환됩니다. 업그레이드를 다시 시도하기 전에 문제를 해결해야 합니다.

제거된 대부분의 함수는 공간 함수이며, 이는 동등한 ST_* 함수로 대체되었습니다. 이러한 경우 새 프로시저 이름 지정을 사용하도록 데이터베이스 객체를 수정합니다. 자세한 내용은 MySQL 설명서의 MySQL 8.0에서 제거된 기능을 참조하십시오.

출력 예제:

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "description": "Error: The following DB objects use functions that were removed
 in the latest MySQL version. Please make sure to update them to use supported
 alternatives before upgrade.",
```

```
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.GetLocationsInPolygon",
        "description": "PROCEDURE uses removed function POLYGONFROMTEXT (consider
 using ST_POLYGONFROMTEXT instead)",
        "dbObjectType": "Routine"
      },
      {
        "level": "Error",
        "dbObject": "test.InsertLocation",
        "description": "PROCEDURE uses removed function POINTFROMTEXT (consider
 using ST_POINTFROMTEXT instead)",
        "dbObjectType": "Routine"
      }
  ]
},
```

사전 확인은 `test.GetLocationsInPolygon` 저장된 프로시저가 POLYGONFROMTEXT 및 POINTFROMTEXT의 제거된 두 가지 함수를 사용하고 있음을 보고합니다. 또한 새 ST_POLYGONFROMTEXT 및 ST_POINTFROMTEXT를 대체용으로 사용할 것을 제안합니다. 제안을 사용하여 프로시저를 다시 생성하면 사전 확인이 성공적으로 완료됩니다.

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "detectedProblems": []
},
```

> **Note**
>
> 대부분의 경우 더 이상 사용되지 않는 함수는 직접적인 대체 항목이 있지만, 애플리케이션을 테스트하고 변경으로 인한 동작 변화가 있는지 설명서를 검토해야 합니다.

routineSyntaxCheck

사전 확인 수준: 오류

루틴과 유사한 객체에 대한 MySQL 구문 확인

MySQL 8.0에서는 이전에 예약되지 않은 [예약된 키워드](https://dev.mysql.com/doc/refman/8.0/en/keywords.html)가 도입되었습니다. 업그레이드 사전 확인기는 데이터베이스 객체의 이름과 정의 및 본문에서 예약된 키워드의 사용을 평가합니다. 저장된 프로시저, 함수, 이벤트 및 트리거와 같은 데이터베이스 객체에 사용되는 예약된 키워드를 감지하면 업그레이드가 실패하고 오류가 `upgrade-prechecks.log` 파일에 게시됩니다. 문제를 해결하려면 업그레이드하기 전에 이러한 객체 정의를 업데이트하고 이러한 참조를 작은따옴표(')로 묶어야 합니다. MySQL 에서 예약된 단어를 이스케이프하는 방법에 대한 자세한 내용은 MySQL 설명서의 [문자열 리터럴](https://dev.mysql.com/doc/refman/8.0/en/string-literals.html)을 참조하세요.

또는 이름을 다른 이름으로 변경할 수 있으며, 이로 인해 애플리케이션을 변경해야 할 수 있습니다.

출력 예제:

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "description": "The following objects did not pass a syntax check with the latest
 MySQL grammar. A common reason is that they reference names that conflict with new
 reserved keywords. You must update these routine definitions and `quote` any such
 references before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
         "level": "Error",
         "dbObject": "test.select_res_word",
         "description": "at line 2,18: unexpected token 'except'",
         "dbObjectType": "Routine"
      }
  ]
}
```

이 문제를 해결하려면 루틴 정의를 확인하세요.

```
SHOW CREATE PROCEDURE test.select_res_word\G

*************************** 1. row ***************************
           Procedure: select_res_word
            sql_mode:
 ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_Z
    Create Procedure: CREATE PROCEDURE 'select_res_word'()
```

```
BEGIN
    SELECT * FROM except;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

이 프로시저에서는 MySQL 8.0의 새 키워드인 `except`라는 테이블을 사용합니다. 문자열 리터럴을 이스케이프하여 프로시저를 다시 생성합니다.

```
> drop procedure if exists select_res_word;
Query OK, 0 rows affected (0.00 sec)

> DELIMITER $$
 > CREATE PROCEDURE select_res_word()
    -> BEGIN
    ->     SELECT * FROM 'except';
    -> END$$
Query OK, 0 rows affected (0.00 sec)

 > DELIMITER ;
```

이제 사전 확인이 통과됩니다.

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "detectedProblems": []
}
```

schemaInconsistencyCheck

사전 확인 수준: 오류

파일 제거 또는 손상으로 인한 스키마 불일치

앞서 설명한 대로 MySQL 8.0은 모든 메타데이터를 `mysql` 스키마의 내부 InnoDB 테이블 세트에 저장하는 원자성 데이터 딕셔너리를 도입했습니다. 이 새로운 아키텍처는 트랜잭션 ACID 규정을 준수하는 방식으로 데이터베이스 메타데이터를 관리하여 이전 파일 기반 접근 방식의 '원자성 DDL' 문제를 해결합니다. MySQL 8.0 이전에는 DDL 작업이 예기치 않게 중단된 경우 스키마 객체가 분

리될 수 있었습니다. 업그레이드 중에 파일 기반 메타데이터를 새 원자성 데이터 딕셔너리 테이블로 마이그레이션하면 DB 인스턴스에 분리된 스키마 객체가 없습니다. 분리된 객체가 발생하면 업그레이드가 실패합니다.

업그레이드를 시작하기 전에 이러한 분리된 객체를 감지하는 데 도움이 되도록 `schemaInconsistencyCheck` 사전 확인을 실행하여 모든 데이터 딕셔너리 메타데이터 객체가 동기화되어 있는지 확인합니다. 분리된 메타데이터 객체가 감지되면 업그레이드가 진행되지 않습니다. 업그레이드를 진행하려면 분리된 메타데이터 객체를 정리합니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

출력 예제:

```
{
  "id": "schemaInconsistencyCheck",
  "title": "Schema inconsistencies resulting from file removal or corruption",
  "status": "OK",
  "description": "Error: Following tables show signs that either table datadir
 directory or frm file was removed/corrupted. Please check server logs, examine
 datadir to detect the issue and fix it before upgrade",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.schemaInconsistencyCheck_failure",
        "description": "present in INFORMATION_SCHEMA's INNODB_SYS_TABLES table but
 missing from TABLES table"
      }
  ]
}
```

사전 확인은 `test.schemaInconsistencyCheck_failure` 테이블에 불일치하는 메타데이터가 있음을 보고합니다. 이 경우 테이블은 InnoDB 스토리지 엔진 메타데이터(`information_schema.INNODB_SYS_TABLES`)에 존재하지만 MySQL 메타데이터(`information_schema.TABLES`)에는 존재하지 않습니다.

오류를 보고하는 Aurora MySQL 사전 확인

다음 사전 확인은 Aurora MySQL에만 적용됩니다.

- auroraCheckDDLRecovery
- auroraCheckRdsUpgradePrechecksTable
- auroraFODUpgradeCheck
- auroraGetDanglingFulltextIndex
- auroraUpgradeCheckForDatafilePathInconsistency
- auroraUpgradeCheckForFtsSpaceIdZero
- auroraUpgradeCheckForIncompleteXATransactions
- auroraUpgradeCheckForInstanceLimit
- auroraUpgradeCheckForInternalUsers
- auroraUpgradeCheckForMasterUser
- auroraUpgradeCheckForPrefixIndexOnGeometryColumns
- auroraUpgradeCheckForSpecialCharactersInProcedures
- auroraUpgradeCheckForSysSchemaObjectTypeMismatch
- auroraUpgradeCheckForViewColumnNameLength
- auroraUpgradeCheckIndexLengthLimitOnTinyColumns
- auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable
- auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

auroraCheckDDLRecovery

사전 확인 수준: 오류

Aurora DDL 복구 기능과 관련된 아티팩트 확인

Aurora MySQL 의 데이터 정의 언어(DDL) 복구 구현의 일환으로, 진행 중인 DDL 문의 메타데이터는 `mysql` 스키마의 `ddl_log_md_table` 및 `ddl_log_table` 테이블에 유지됩니다. 이 기능은 MySQL 8.0의 새로운 원자성 데이터 딕셔너리 구현의 일부이므로 버전 3 이상에서는 Aurora의 DDL 복구 구현이 지원되지 않습니다. 호환성 검사 중에 실행 중인 DDL 문이 있는 경우 이 사전 검사가 실패할 수 있습니다. DDL 문이 실행되지 않는 동안 업그레이드를 시도하는 것이 좋습니다.

DDL 문을 실행하지 않았는데 이 사전 확인이 실패하면 AWS 지원에서 사례를 개설하여 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

DDL 문이 실행 중인 경우 사전 확인 출력은 다음 메시지를 표시합니다.

```
“There are DDL statements in process. Please allow DDL statements to finish before
 upgrading.”
```

출력 예제:

```
{
  "id": "auroraCheckDDLRecovery",
  "title": "Check for artifacts related to Aurora DDL recovery feature",
  "status": "OK",
  "description": "Aurora implementation of DDL recovery is not supported from 3.x
 onwards. This check verifies that the database do not have artifacts realted to the
 feature",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_md_table",
        "description": "Table mysql.ddl_log_md_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_table",
        "description": "Table mysql.ddl_log_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "information_schema.processlist",
        "description": "There are DDL statements in process. Please allow DDL
 statements to finish before upgrading."
      }
  ]
}
```

사전 확인이 호환성 확인과 동시에 실행되는 진행 중인 DDL로 인해 오류를 반환했습니다. DDL을 실행하지 않고 업그레이드를 다시 시도하는 것이 좋습니다.

auroraCheckRdsUpgradePrechecksTable

사전 확인 수준: 오류

mysql.rds_upgrade_prechecks 테이블의 존재 확인

RDS 서비스에서 수행하는 내부 전용 사전 확인입니다. 모든 오류는 업그레이드 시 자동으로 처리되며 안전하게 무시할 수 있습니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

```
{
  "id": "auroraCheckRdsUpgradePrechecksTable",
  "title": "Check existence of mysql.rds_upgrade_prechecks table",
  "status": "OK",
  "detectedProblems": []
}
```

auroraFODUpgradeCheck

사전 확인 수준: 오류

Aurora 빠른 DDL 기능과 관련된 아티팩트 확인

일부 DDL 작업의 효율성을 개선하기 위해 Aurora MySQL 버전 2의 랩 모드에서 빠른 DDL 최적화가 도입되었습니다. Aurora MySQL 버전 3에서는 랩 모드가 제거되었으며, 빠른 DDL 구현이 인스턴트 DDL이라는 MySQL 8.0 기능으로 대체되었습니다.

Aurora MySQL 버전 3으로 업그레이드하기 전에 랩 모드에서 빠른 DDL을 사용하는 모든 테이블은 Aurora MySQL 버전 3과의 호환성을 보장하기 위해 `OPTIMIZE TABLE` 또는 `ALTER TABLE ... ENGINE=InnoDB` 명령을 실행하여 다시 빌드해야 합니다.

이 사전 확인은 이러한 테이블의 목록을 반환합니다. 반환된 테이블을 다시 빌드한 후 업그레이드를 다시 시도할 수 있습니다.

출력 예제:

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "description": "Aurora fast DDL is not supported from 3.x onwards. This check
 verifies that the database does not have artifacts related to the feature",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/AuroraMySQL.Managing.FastDDL.html#AuroraMySQL.Managing.FastDDL-v2",
  "detectedProblems": [
```

```
      {
        "level": "Error",
        "dbObject": "test.test",
        "description": "Your table has pending Aurora fast DDL operations. Run
 'OPTIMIZE TABLE <table name>' for the table to apply all the pending DDL updates.
 Then try the upgrade again."
      }
   ]
}
```

사전 확인은 테이블 test.test에 보류 중인 빠른 DDL 작업이 있음을 보고합니다.

업그레이드를 계속하려면 테이블을 다시 빌드한 다음 업그레이드를 다시 시도할 수 있습니다.

> **Note**
>
> 테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 온라인 DDL 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해하세요.

```
mysql> optimize table test.test;
+-----------+----------+----------
+-------------------------------------------------------------------+
| Table     | Op       | Msg_type | Msg_text
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
| test.test | optimize | note     | Table does not support optimize, doing recreate
 + analyze instead |
| test.test | optimize | status   | OK
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.04 sec)
```

테이블을 다시 빌드한 후 사전 확인이 성공합니다.

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

auroraGetDanglingFulltextIndex

사전 확인 수준: 오류

누락된 **FULLTEXT** 인덱스 참조가 있는 테이블

MySQL 5.6.26 이전에는 전체 텍스트 검색 인덱스를 삭제한 후 숨겨진 FTS_DOC_ID 및 FTS_DOC_ID_INDEX 열이 분리될 수 있었습니다. 자세한 내용은 버그 #76012를 참조하세요.

이러한 문제가 발생한 이전 버전의 MySQL에서 테이블이 생성된 경우 Aurora MySQL 버전 3으로의 업그레이드가 실패할 수 있습니다. 이 사전 확인은 MySQL 8.0으로 업그레이드하기 전에 DB 클러스터에 이렇게 분리되었거나 누락된 전체 텍스트 인덱스가 없는지 확인합니다. 이 사전 확인이 실패하면 이러한 누락된 전체 텍스트 인덱스가 포함된 테이블을 다시 빌드합니다.

출력 예제:

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which
 is not supported. It is recommended to rebuild the table before upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_fts_index",
        "description": "Table `test.table_with_fts_index` contains dangling FULLTEXT
 index. Kindly recreate the table before upgrade."
      }
  ]
},
```

test.table_with_fts_index 테이블에 누락된 전체 텍스트 인덱스가 포함되어 있으므로 사전 확인은 테이블에 대한 오류를 보고합니다. 업그레이드를 진행하려면 테이블을 다시 빌드하여 전체 텍스트 인덱스 보조 테이블을 정리합니다. OPTIMIZE TABLE test.table_with_fts_index 또는 ALTER TABLE test.table_with_fts_index, ENGINE=INNODB를 사용합니다.

테이블을 다시 빌드한 후 사전 확인이 통과됩니다.

```
{
  "id": "auroraGetDanglingFulltextIndex",
```

```
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForDatafilePathInconsistency

사전 확인 수준: 오류

ibd 파일 경로와 관련된 불일치 확인

이 사전 확인은 Aurora MySQL 버전 3.03.0 이하에만 적용됩니다. 이 사전 확인에 오류가 발생하면 Aurora MySQL 버전 3.04 이상으로 업그레이드하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForFtsSpaceIdZero

사전 확인 수준: 오류

스페이스 ID가 0인 전체 텍스트 인덱스 확인

MySQL에서 InnoDB 테이블에 전체 텍스트 인덱스를 추가하면 여러 보조 인덱스 테이블스페이스가 생성됩니다. 이전 버전의 MySQL에 버그(버전 5.6.20에서 수정됨)가 있어 이러한 보조 인덱스 테이블이 자체 InnoDB 테이블스페이스가 아닌 시스템 테이블스페이스에 생성되었을 수 있습니다.

이러한 보조 테이블스페이스가 있는 경우 업그레이드가 실패합니다. 사전 확인 오류에 언급된 전체 텍스트 인덱스를 다시 생성한 다음 업그레이드를 다시 시도합니다.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForFtsSpaceIdZero",
  "title": "Check for fulltext index with space id as zero",
  "status": "OK",
  "description": "The auxiliary tables of FTS indexes on the table are created
 in system table-space. Due to this DDL queries executed on MySQL8.0 shall cause
```

```
database unavailability. To avoid that, drop and recreate all the FTS indexes on
the table or rebuild the table using ALTER TABLE query before the upgrade.",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test.fts_space_zero_check",
       "description": " The auxiliary tables of FTS indexes on the table
'test.fts_space_zero_check' are created in system table-space due to https://
bugs.mysql.com/bug.php?id=72132. In MySQL8.0, DDL queries executed on this table
shall cause database unavailability. To avoid that, drop and recreate all the
FTS indexes on the table or rebuild the table using ALTER TABLE query before the
upgrade."
     }
 ]
},
```

사전 확인은 시스템 테이블스페이스에 보조 전체 텍스트 검색(FTS) 테이블이 있으므로 `test.fts_space_zero_check` 테이블에 대한 오류를 보고합니다.

이 테이블과 연결된 FTS 인덱스를 삭제하고 다시 생성하면 사전 확인이 성공합니다.

Note

테이블스페이스를 다시 빌드하기 전에 MySQL 설명서의 파티셔닝 작업을 참조하여 잠금 및 데이터 이동이 전경 트랜잭션에 미치는 영향을 이해합니다.

```
{
 "id": "auroraUpgradeCheckForFtsSpaceIdZero",
 "title": "Check for fulltext index with space id as zero",
 "status": "OK",
 "detectedProblems": []
}
```

auroraUpgradeCheckForIncompleteXATransactions

사전 확인 수준: 오류

준비된 상태의 XA 트랜잭션 확인

메이저 버전 업그레이드 프로세스를 실행하는 동안 Aurora MySQL 버전 2 DB 인스턴스가 완전히 종료되는 것이 중요합니다. 이렇게 하면 모든 트랜잭션이 커밋되거나 롤백되고 InnoDB가 모든 실

행 취소 로그 레코드를 제거하게 됩니다. 트랜잭션 롤백이 필요하기 때문에 데이터베이스에 준비된 상태의 XA 트랜잭션이 있는 경우 완전한 종료가 계속되는 것을 차단할 수 있습니다. 따라서 준비된 XA 트랜잭션이 감지되면 커밋 또는 롤백 작업을 수행할 때까지 업그레이드를 진행할 수 없습니다.

`XA RECOVER`를 사용하여 준비된 상태의 XA 트랜잭션을 찾는 방법에 대한 자세한 내용은 MySQL 설명서의 XA 트랜잭션 SQL 문을 참조하세요. XA 트랜잭션 상태에 대한 자세한 내용은 MySQL 설명서의 XA 트랜잭션 상태를 참조하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "description": "Your cluster currently has XA transactions in the prepared state.
 To proceed with the upgrade, commit or rollback these transactions.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your cluster currently has XA transactions in the prepared
 state. To proceed with the upgrade, commit or rollback these transactions."
      }
  ]
}
```

이 사전 확인은 커밋하거나 롤백해야 하는 준비된 상태의 트랜잭션이 있기 때문에 오류를 보고합니다.

데이터베이스에 로그인한 후 information_schema.innodb_trx 테이블과 `XA RECOVER` 출력에서 자세한 내용을 확인할 수 있습니다.

> **Important**
>
> 트랜잭션을 커밋하거나 롤백하기 전에 MySQL 설명서와 애플리케이션 요구 사항을 검토하는 것이 좋습니다.

```
mysql> select trx_started,
    trx_mysql_thread_id,
```

```
    trx_id,trx_state,
    trx_operation_state,
    trx_rows_modified,
    trx_rows_locked
from
    information_schema.innodb_trx;
+---------------------+---------------------+---------+-----------
+---------------------+------------------+-----------------+
| trx_started         | trx_mysql_thread_id | trx_id  | trx_state |
 trx_operation_state | trx_rows_modified | trx_rows_locked |
+---------------------+---------------------+---------+-----------
+---------------------+------------------+-----------------+
| 2024-08-12 01:09:39 |                   0 | 2849470 | RUNNING   | NULL
    |                 1 |               0 |
+---------------------+---------------------+---------+-----------
+---------------------+------------------+-----------------+
1 row in set (0.00 sec)

mysql> xa recover;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        1 |            6 |            0 | xatest |
+----------+--------------+--------------+--------+
1 row in set (0.00 sec)
```

이 경우 준비된 트랜잭션을 롤백합니다.

```
mysql> XA ROLLBACK 'xatest';
Query OK, 0 rows affected (0.00 sec)
v
mysql> xa recover;
Empty set (0.00 sec)
```

XA 트랜잭션이 롤백되면 사전 확인이 성공합니다.

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInstanceLimit

사전 확인 수준: 오류

현재 인스턴스 클래스에서 업그레이드가 지원되는지 확인

라이터 DB 인스턴스 클래스가 db.r6i.32xlarge인 Aurora MySQL 버전 2.12.0 또는 2.12.1에서 현재 위치 업그레이드를 실행하는 것은 현재 지원되지 않습니다. 이 경우 사전 확인은 오류를 반환합니다. 업그레이드를 계속하려면 DB 인스턴스 클래스를 db.r6i.24xlarge 이하로 변경할 수 있습니다. 또는 Aurora MySQL 버전 2.12.2 이상으로 업그레이드할 수 있습니다. 여기서 Aurora MySQL 버전 3으로의 현재 위치 업그레이드는 db.r6i.32xlarge에서 지원됩니다.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForInstanceLimit",
  "title": "Checks if upgrade is supported on the current instance class",
  "status": "OK",
  "description": "Upgrade from Aurora Version 2.12.0 and 2.12.1 may fail for 32.xl
 and above instance class.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Upgrade is not supported on this instance size for Aurora
 MySql Version 2.12.1. Before upgrading to Aurora MySql 3, please consider either:
 1. Changing the instance class to 24.xl or lower. -or- 2. Upgrading to patch
 version 2.12.2 or higher."
      }
  ]
},
```

사전 확인은 라이터 DB 인스턴스가 db.r6i.32xlarge 인스턴스 클래스를 사용하고 Aurora MySQL 버전 2.12.1에서 실행 중이므로 오류를 반환합니다.

auroraUpgradeCheckForInternalUsers

사전 확인 수준: 오류

8.0 내부 사용자 확인

이 사전 확인은 Aurora MySQL 버전 3.03.0 이하에만 적용됩니다. 이 사전 확인에 오류가 발생하면 Aurora MySQL 버전 3.04 이상으로 업그레이드하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForInternalUsers",
  "title": "Check for 8.0 internal users.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForMasterUser

사전 확인 수준: 오류

RDS 마스터 사용자가 존재하는지 확인

MySQL 8에는 권한 관리를 더 쉽고 세밀하게 만들 수 있도록 역할과 동적 권한을 지원하는 새로운 권한 모델이 추가되었습니다. 이 변경 사항의 일환으로 Aurora MySQL은 Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 때 데이터베이스의 마스터 사용자에게 자동으로 부여되는 새로운 `rds_superuser_role`을 도입했습니다.

Aurora MySQL에서 마스터 사용자에게 할당되는 역할 및 권한에 대한 자세한 내용은 마스터 사용자 계정 권한 섹션을 참조하세요. Aurora MySQL 버전 3의 역할 기반 권한 모델에 대한 자세한 내용은 역할 기반 권한 모델 섹션을 참조하세요.

이 사전 확인은 마스터 사용자가 데이터베이스에 있는지 확인합니다. 마스터 사용자가 없는 경우 사전 확인이 실패합니다. 업그레이드를 계속하려면 마스터 사용자 암호를 재설정하거나 사용자를 수동으로 생성하여 마스터 사용자를 다시 생성합니다. 그런 다음 업그레이드를 다시 시도하세요. 마스터 사용자 암호 재설정에 대한 자세한 내용은 데이터베이스 마스터 사용자의 암호 변경 섹션을 참조하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  "title": "Check if master user exists",
  "status": "OK",
  "description": "Throws error if master user has been dropped!",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/UsingWithRDS.MasterAccounts.html",
  "detectedProblems": [
      {
        "level": "Error",
```

```
        "dbObject": "all",
        "description": "Your Master User on host '%' has been dropped. To proceed
 with the upgrade, recreate the master user `reinvent` on default host '%'"
      }
    ]
}
```

마스터 사용자 암호를 재설정하면 사전 확인이 통과되고 업그레이드를 다시 시도할 수 있습니다.

다음 예제에서는 AWS CLI를 사용하여 암호를 재설정합니다. 암호 변경 사항이 바로 적용됩니다.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --master-user-password my-new-password
```

그러면 사전 확인이 성공합니다.

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  title": "Check if master user exists",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForPrefixIndexOnGeometryColumns

사전 확인 수준: 오류

접두사 인덱스의 지오메트리 열 확인

MySQL 8.0.12부터는 GEOMETRY 데이터 유형을 사용하여 열에 접두사 인덱스를 더 이상 생성할 수 없습니다. 자세한 내용은 WL#11808을 참조하세요.

이러한 인덱스가 있으면 업그레이드가 실패합니다. 문제를 해결하려면 사전 확인 실패에 언급된 테이블에 접두사가 붙은 `GEOMETRY` 인덱스를 삭제합니다.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
```

```
  "description": "Consider dropping the prefix Indexes of geometry columns and
restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.geom_index_prefix",
        "description": "Table `test`.`geom_index_prefix` has an index `LatLon` on
geometry column/s. Mysql 8.0 does not support this type of index on a geometry
column https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run
'DROP INDEX `LatLon` ON `test`.`geom_index_prefix`;"
      }
  ]
}
```

test.geom_index_prefix 테이블에 GEOMETRY 열에 접두사가 있는 인덱스가 포함되어 있기 때문에 사전 확인은 오류를 보고합니다.

이 인덱스를 삭제하면 사전 확인이 성공합니다.

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForSpecialCharactersInProcedures

사전 확인 수준: 오류

저장된 프로시저의 특수 문자와 관련된 불일치 확인

MySQL 8.0 이전에는 데이터베이스 이름, 테이블 이름 및 기타 객체가 데이터 디렉터리의 파일, 즉 파일 기반 메타데이터에 상응했습니다. MySQL 8.0으로 업그레이드하는 과정에서 모든 데이터베이스 객체가 mysql 스키마에 저장된 새 내부 데이터 딕셔너리 테이블로 마이그레이션되어 새로 구현된 원자성 데이터 딕셔너리를 지원합니다. 저장된 프로시저를 마이그레이션하는 과정에서 각 프로시저의 프로시저 정의와 본문은 새 데이터 딕셔너리로 수집될 때 검증됩니다.

MySQL 8 이전에는 경우에 따라 저장된 루틴을 생성하거나 특수 문자가 포함된 프로시저인 mysql.proc 테이블에 직접 삽입할 수 있었습니다. 예를 들어 규정을 준수하지 않는 줄 바꿈하지 않는 공백 \xa0이 있는 주석이 포함된 저장 프로시저를 생성할 수 있습니다. 이러한 프로시저가 발생하면 업그레이드가 실패합니다.

이 사전 확인은 저장된 프로시저의 본문과 정의에 이러한 문자가 포함되어 있지 않은지 확인합니다. 업그레이드를 진행하려면 숨겨진 문자나 특수 문자 없이 이러한 저장된 프로시저를 다시 생성합니다.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "description": "Following procedure(s) has special characters inconsistency.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "information_schema.routines",
        "description": "Data Dictionary Metadata is inconsistent for the procedure
 `get_version_proc` in the database `test` due to usage of special characters in
 procedure body. To avoid that, drop and recreate the procedure without any special
 characters before proceeding with the Upgrade."
      }
  ]
}
```

사전 확인은 DB 클러스터에 프로시저 본문에 특수 문자가 포함된 `test` 데이터베이스에 `get_version_proc`이라는 프로시저가 포함되어 있음을 보고합니다.

저장된 프로시저를 삭제하고 다시 생성한 후 사전 확인이 성공하여 업그레이드를 진행할 수 있습니다.

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForSysSchemaObjectTypeMismatch

사전 확인 수준: 오류

sys 스키마에 대한 객체 유형 불일치 확인

sys 스키마는 사용자가 DB 인스턴스의 문제를 해결하고 최적화하고 모니터링하는 데 도움이 되는 MySQL 데이터베이스의 객체 및 뷰 세트입니다. Aurora MySQL 버전 2에서 버전 3으로 메이저 버전 업그레이드를 수행할 때 sys 스키마 뷰가 다시 생성되고 새 Aurora MySQL 버전 3 정의로 업데이트됩니다.

업그레이드의 일환으로 sys 스키마의 모든 객체가 뷰가 아닌 스토리지 엔진 (INFORMATION_SCHEMA.TABLES의 sys_config/BASE TABLE)을 사용하여 정의된 경우 업그레이드가 실패합니다. 이러한 테이블은 information_schema.tables 테이블에서 찾을 수 있습니다. 이는 예상되는 동작은 아니지만 경우에 따라 사용자 오류로 인해 발생할 수 있습니다.

이 사전 확인은 스키마 객체가 올바른 테이블 정의를 사용하고 뷰가 InnoDB 또는 MyISAM 테이블로 잘못 정의되지 않았는지 확인하기 위해 모든 sys 스키마 객체를 검증합니다. 문제를 해결하려면 반환된 객체의 이름을 바꾸거나 삭제하여 수동으로 수정합니다. 그런 다음 업그레이드를 다시 시도하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "description": "Database contains objects with type mismatch for sys schema.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "sys.waits_global_by_latency",
        "description": "Your object sys.waits_global_by_latency has a type mismatch.
 To fix the inconsistency we recommend to rename or remove the object before
 upgrading (use RENAME TABLE command). "
      }
  ]
}
```

사전 확인은 sys 스키마의 sys.waits_global_by_latency 뷰에 유형 불일치가 있어 업그레이드가 진행되지 않는다고 보고합니다.

DB 인스턴스에 로그인한 후 이 객체가 뷰여야 하지만 InnoDB 테이블로 정의되어 있음을 확인할 수 있습니다.

```
mysql> show create table sys.waits_global_by_latency\G
*************************** 1. row ***************************
```

```
       Table: waits_global_by_latency
Create Table: CREATE TABLE `waits_global_by_latency` (
  `events` varchar(128) DEFAULT NULL,
  `total` bigint(20) unsigned DEFAULT NULL,
  `total_latency` text,
  `avg_latency` text,
  `max_latency` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

이 문제를 해결하기 위해 뷰를 삭제하고 올바른 정의로 다시 생성하거나 이름을 바꿀 수 있습니다. 업그레이드 프로세스 중에 올바른 테이블 정의로 자동으로 생성됩니다.

```
mysql> RENAME TABLE sys.waits_global_by_latency to sys.waits_global_by_latency_old;
Query OK, 0 rows affected (0.01 sec)
```

이렇게 하면 사전 확인이 통과됩니다.

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForViewColumnNameLength

사전 확인 수준: 오류

뷰에서 열 이름의 상한 확인

MySQL에서 열 이름의 최대 허용 길이는 64자입니다. MySQL 8.0 이전에는 열 이름이 64자보다 큰 뷰를 생성할 수 있었습니다. 데이터베이스 인스턴스에 이러한 뷰가 있는 경우 사전 확인 오류가 반환되고 업그레이드가 실패합니다. 업그레이드를 계속하려면 열 길이가 64자 미만으로 해당 뷰를 다시 생성해야 합니다. 그런 다음 업그레이드를 다시 시도하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
```

```
  "description": "Following view(s) has column(s) with length greater than 64.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject":
 "test.colname_view_test.col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pa
        "description": "View `test`.`colname_view_test`has column
 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` with
 invalid column name length. To avoid Upgrade errors, view should be altered by
 renaming the column name so that its length is not 0 and doesn't exceed 64."
      }
  ]
}
```

사전 확인은 `test.colname_view_test` 뷰에 허용되는 최대 열 길이인 64자를 초과하는 열 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad`가 포함되어 있음을 보고합니다.

뷰 정의에서 문제가 되는 열을 볼 수 있습니다.

```
mysql> desc `test`.`colname_view_test`;
+------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| Field                                                            | Type        |
 Null | Key | Default | Extra |
+------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| col1                                                             | varchar(20) |
 YES  |     | NULL    |       |
| col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad | int(11)     |
 YES  |     | NULL    |       |
+------------------------------------------------------------------+-------------
+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

업그레이드를 계속하려면 열 길이가 64자를 초과하지 않도록 뷰를 다시 생성하세요.

```
mysql> drop view `test`.`colname_view_test`;
Query OK, 0 rows affected (0.01 sec)

mysql> create view `test`.`colname_view_test`(col1, col2_nopad) as select inf,
 fodcol from test;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> desc `test`.`colname_view_test`;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| col1       | varchar(20) | YES  |     | NULL    |       |
| col2_nopad | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

이렇게 하면 사전 확인이 성공합니다.

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckIndexLengthLimitOnTinyColumns

사전 확인 수준: 오류

작은 열에서 접두사 길이가 255바이트를 초과하는 인덱스가 정의된 테이블 확인

MySQL에서 바이너리 데이터 유형을 사용하여 열에 인덱스를 생성할 때는 인덱스 정의에 접두사 길이를 추가해야 합니다. MySQL 8.0 이전에는 경우에 따라 이러한 데이터 유형의 최대 허용 크기보다 큰 접두사 길이를 지정할 수 있었습니다. 허용되는 최대 데이터 크기가 255바이트이지만 이보다 큰 인덱스 접두사가 허용된 TINYTEXT 및 TINYBLOB 열을 예로 들 수 있습니다. 자세한 내용은 MySQL 설명서의 InnoDB 한도를 참조하세요.

이 사전 확인이 실패하면 문제가 되는 인덱스를 삭제하거나 인덱스에서 TINYTEXT 및 TINYBLOB 열의 접두사 길이를 255바이트 미만으로 줄입니다. 그런 다음 업그레이드를 다시 시도하세요.

출력 예제:

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "description": "Prefix length of the indexes defined on tiny columns cannot
 exceed 255 bytes. With utf8mb4 char set, this limits the prefix length supported
```

```
upto 63 characters only. A larger prefix length was allowed in MySQL5.7 using
innodb_large_prefix parameter. This parameter is deprecated in MySQL 8.0.",
 "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/innodb-limits.html,
https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test.tintxt_prefixed_index.col1",
       "description": "Index 'PRIMARY' on tinytext/tinyblob column `col1` of table
`test.tintxt_prefixed_index` is defined with prefix length exceeding 255 bytes.
Reduce the prefix length to <=255 bytes depending on character set used. For
utf8mb4, it should be <=63."
     }
  ]
}
```

사전 확인은 TINYTEXT 또는 TINYBLOB 열에 접두사가 255바이트보다 큰 PRIMARY 인덱스가 있기 때문에 `test.tintxt_prefixed_index` 테이블에 대한 오류를 보고합니다.

이 테이블의 정의를 보면 `col1` 열의 TINYTEXT에서 프라이머리 키의 접두사가 65임을 알 수 있습니다. 테이블은 문자당 4바이트를 저장하는 `utf8mb4` 문자 세트를 사용하여 정의되므로 접두사가 255바이트 제한을 초과합니다.

```
mysql> show create table `test`.`tintxt_prefixed_index`\G
*************************** 1. row ***************************
       Table: tintxt_prefixed_index
Create Table: CREATE TABLE `tintxt_prefixed_index` (
  `col1` tinytext NOT NULL,
  `col2` tinytext,
  `col_id` tinytext,
  PRIMARY KEY (`col1`(65))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```

`utf8mb4` 문자 세트를 사용하는 동안 인덱스 접두사를 63으로 수정하면 업그레이드를 진행할 수 있습니다.

```
mysql> alter table `test`.`tintxt_prefixed_index` drop PRIMARY KEY, ADD  PRIMARY KEY
 (`col1`(63));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

이렇게 하면 사전 확인이 성공합니다.

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

사전 확인 수준: 오류

mysql.host 테이블에 누락된 InnoDB 메타데이터 불일치 확인

RDS 서비스에서 수행하는 내부 전용 사전 확인입니다. 모든 오류는 업그레이드 시 자동으로 처리되며 안전하게 무시할 수 있습니다.

이 사전 확인에 오류가 발생하면 AWS 지원에서 사례를 열어 메타데이터 불일치를 해결하도록 요청합니다. 또는 논리적 덤프를 수행한 다음 새 Aurora MySQL 버전 3 DB 클러스터로 복원하여 업그레이드를 다시 시도할 수 있습니다.

auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

사전 확인 수준: 오류

잘못된 utf8mb3 열 주석 확인

이 사전 확인은 잘못된 utf8mb3 문자 인코딩이 있는 열 주석이 포함된 테이블을 식별합니다. MySQL 8.0에서는 열 주석을 포함하여 메타데이터의 문자 인코딩에 더 엄격한 검증이 적용됩니다. utf8mb3 문자 집합에 유효하지 않은 문자가 포함된 열 주석이 있는 경우 업그레이드가 실패합니다.

이 문제의 가장 일반적인 원인은 열 주석에 기본 다국어 평면(BMP) 이외의 문자가 있는 것입니다. BMP 이외 문자에는 4바이트 UTF-8 인코딩(utf8mb4)이 필요한데 utf8mb3는 3바이트 시퀀스만 지원하여 이러한 문자를 나타낼 수 없습니다.

이 문제를 해결하려면 업그레이드를 시도하기 전에 열 주석을 수정하여 BMP 이외 문자를 제거하거나 바꿔야 합니다. `ALTER TABLE` 문을 사용하여 열 주석을 업데이트할 수 있습니다.

출력 예제:

```
{
```

```
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "description": "Following table(s) has/have invalid utf8mb3 comments on the
 column/columns.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.t2",
        "description": "Table test.t2 has invalid utf8mb3 comments in it's column/
columns. This is due to non-BMP characters in the comment field. To fix the
 inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
      }
  ]
}
```

사전 확인 결과, 특히 BMP 이외 문자로 인해 `test.t2` 테이블의 하나 이상의 열 주석에 잘못된 utf8mb3 문자가 포함되어 있는 것으로 나타났습니다.

이 문제를 해결하려면 문제가 있는 열을 식별하고 주석을 업데이트해야 합니다. 먼저 테이블 구조를 조사하여 주석이 있는 열을 식별합니다.

```
mysql> SHOW CREATE TABLE test.t2\G
```

문제가 있는 주석이 포함된 열을 식별한 후에는 `ALTER TABLE` 문을 사용하여 해당 열을 업데이트합니다. 예시:

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT 'Updated
 comment without non-BMP characters';
```

또는 주석을 완전히 제거할 수도 있습니다.

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT '';
```

문제가 있는 열 주석을 모두 업데이트하면 사전 확인이 통과되고 업그레이드를 진행할 수 있습니다.

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
```

```
  "status": "OK",
  "detectedProblems": []
}
```

> **Note**
> 열 주석을 수정하기 전에 이러한 주석에 의존하는 모든 애플리케이션 코드 또는 문서를 그에 따라 업데이트해야 합니다. 애플리케이션에 BMP 이외 문자가 필요한 경우 더 나은 유니코드 지원을 위해 utf8mb4 문자 집합으로 마이그레이션하는 것이 좋습니다.

경고

다음 사전 확인은 사전 확인이 실패할 때 경고를 생성하지만 업그레이드를 진행할 수 있습니다.

주제

- 경고를 보고하는 MySQL 사전 확인
- 경고를 보고하는 Aurora MySQL 사전 확인

경고를 보고하는 MySQL 사전 확인

다음 사전 확인은 Community MySQL에서 가져온 것입니다.

- defaultAuthenticationPlugin
- maxdbFlagCheck
- mysqlDollarSignNameCheck
- reservedKeywordsCheck
- utf8mb3Check
- zeroDatesCheck

defaultAuthenticationPlugin

사전 확인 수준: 경고

새로운 기본 인증 플러그인 고려 사항

MySQL 8.0에는 `caching_sha2_password` 인증 플러그인이 도입되어 더 이상 사용되지 않는 `mysql_native_password` 플러그인보다 더 안전한 암호 암호화와 더 나은 성능을 제공합

니다. Aurora MySQL 버전 3의 경우 데이터베이스 사용자에게 사용되는 기본 인증 플러그인은 mysql_native_password 플러그인입니다.

이 사전 확인은 이 플러그인이 제거되고 향후 메이저 버전 릴리스에서 기본값이 변경된다는 것을 경고합니다. 이 변경 전에 애플리케이션 클라이언트와 사용자의 호환성을 평가하는 것이 좋습니다.

자세한 내용은 MySQL 설명서의 caching_sha2_password 호환성 문제 및 해결 방법을 참조하세요.

출력 예제:

```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
 'caching_sha2_password' offers more secure password hashing than previously
 used 'mysql_native_password' (and consequent improved client connection
 authentication). However, it also has compatibility implications that
 may affect existing MySQL installations. If your MySQL installation
 must serve pre-8.0 clients and you encounter compatibility issues after
 upgrading, the simplest way to address those issues is to reconfigure
 the server to revert to the previous default authentication plugin
 (mysql_native_password). For example, use these lines in the server option file:
\n\n[mysqld]\ndefault_authentication_plugin=mysql_native_password\n\nHowever, the
 setting should be viewed as temporary, not as a long term or permanent solution,
 because it causes new accounts created with the setting in effect to forego the
 improved authentication security.\nIf you are using replication please take time to
 understand how the authentication plugin changes may impact you.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
},
```

maxdbFlagCheck

사전 확인 수준: 경고

더 이상 사용되지 않는 **MAXDB sql_mode** 플래그 사용

MySQL 8.0에서는 더 이상 사용되지 않는 여러 sql_mode 시스템 변수 옵션이 제거되었으며, 그 중 하나는 MAXDB였습니다. 이 사전 확인은 루틴 및 트리거와 함께 현재 연결된 모든 세션을 검사하여 MAXDB를 포함하는 조합으로 sql_mode가 설정된 세션이 없는지 확인합니다.

출력 예제:

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "description": "Warning: The following DB objects have the obsolete MAXDB
 option persisted for sql_mode, which will be cleared during the upgrade. It can
 potentially change the datatype DATETIME into TIMESTAMP if it is used inside
 object's definition, and this in turn can change the behavior in case of dates
 earlier than 1970 or later than 2037. If this is a concern, please redefine these
 objects so that they do not rely on the MAXDB flag before running the upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.maxdb_stored_routine",
        "description": "PROCEDURE uses obsolete MAXDB sql_mode",
        "dbObjectType": "Routine"
      }
  ]
}
```

사전 확인은 `test.maxdb_stored_routine` 루틴에 지원되지 않는 `sql_mode` 옵션이 포함되어 있음을 보고합니다.

데이터베이스에 로그인한 후 루틴 정의에서 `sql_mode`에 `MAXDB`가 포함되어 있는 것을 볼 수 있습니다.

```
 > SHOW CREATE PROCEDURE test.maxdb_stored_routine\G

*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode:
 PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,MAXDB,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPT
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
```

```
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

문제를 해결하려면 클라이언트에서 올바른 sql_mode를 설정한 후 프로시저를 다시 생성합니다.

> **Note**
>
> MySQL 설명서에 따라 MySQL은 루틴이 생성되거나 변경될 때 적용되는 sql_mode 설정을 저장합니다. 루틴이 시작될 때의 sql_mode 설정에 관계없이 항상 이 설정으로 루틴을 실행합니다.
> sql_mode를 변경하기 전에 MySQL 설명서의 서버 SQL 모드를 참조하세요. 이 작업이 애플리케이션에 미치는 잠재적 영향을 신중하게 평가합니다.

지원되지 않는 sql_mode를 사용하지 않고 프로시저를 다시 생성합니다.

```
mysql > set sql_mode='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql > DROP PROCEDURE test.maxdb_stored_routine\G
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER $$
mysql >
mysql > CREATE PROCEDURE test.maxdb_stored_routine()
    -> SQL SECURITY DEFINER
    -> BEGIN
    ->     SELECT * FROM test;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER ;
mysql > show create procedure test.maxdb_stored_routine\G
*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
```

```
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

사전 확인에 성공합니다.

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "detectedProblems": []
}
```

mysqlDollarSignNameCheck

사전 확인 수준: 경고

객체 이름에서 더 이상 사용되지 않는 단일 달러 기호 사용 확인

MySQL 8.0.32부터 인용되지 않은 식별자의 첫 번째 문자로 달러 기호($)를 사용하는 것은 더 이상 지원되지 않습니다. $ 기호를 첫 번째 문자로 포함하는 스키마, 테이블, 뷰, 열 또는 루틴이 있는 경우 이 사전 확인은 경고를 반환합니다. 이 경고로 인해 업그레이드가 차단되지는 않지만 이 문제를 해결하기 위해 조만간 조치를 취하는 것이 좋습니다. MySQL 8.4에서 이러한 식별자는 경고가 아닌 구문 오류를 반환합니다.

출력 예제:

```
{
  "id": "mysqlDollarSignNameCheck",
  "title": "Check for deprecated usage of single dollar signs in object names",
  "status": "OK",
  "description": "Warning: The following objects have names with deprecated usage
 of dollar sign ($) at the begining of the identifier. To correct this warning,
 ensure, that names starting with dollar sign, also end with it, similary to quotes
 ($example$). ",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.$deprecated_syntx",
```

```
        "description": " name starts with $ sign."
      }
  ]
},
```

test 스키마의 $deprecated_syntx 테이블에는 $ 기호가 첫 번째 문자로 포함되어 있으므로 사전 확인은 경고를 보고합니다.

reservedKeywordsCheck

사전 확인 수준: 경고

이름이 예약된 새 키워드와 충돌하는 데이터베이스 객체 사용

이 확인은 이름이 예약된 새 키워드와 충돌하는 데이터베이스 객체의 사용을 확인한다는 점에서 routineSyntaxCheck와 유사합니다. 업그레이드를 차단하지는 않지만 경고를 신중하게 평가해야 합니다.

출력 예제:

테이블 이름이 except인 이전 예제를 사용하면 사전 확인이 경고를 반환합니다.

```
{
  "id": "reservedKeywordsCheck",
  "title": "Usage of db objects with names conflicting with new reserved keywords",
  "status": "OK",
  "description": "Warning: The following objects have names that conflict with
 new reserved keywords. Ensure queries sent by your applications use `quotes` when
 referring to them or they will result in errors.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.except",
        "description": "Table name",
        "dbObjectType": "Table"
      }
  ]
}
```

이 경고를 통해 검토할 몇 가지 애플리케이션 쿼리가 있을 수 있음을 알 수 있습니다. 애플리케이션 쿼리가 문자열 리터럴을 올바르게 이스케이프하지 않는 경우 MySQL 8.0으로 업그레이드한 후 오

류가 발생할 수 있습니다. 애플리케이션을 검토하여 버전 3에서 실행되는 Aurora MySQL DB 클러스터의 복제본 또는 스냅샷을 기준으로 테스트하는 방법으로 확인합니다.

업그레이드 후 실패하는 이스케이프되지 않은 애플리케이션 쿼리의 예:

```
SELECT * FROM escape;
```

업그레이드 후 성공하는 올바르게 이스케이프된 애플리케이션 쿼리의 예:

```
SELECT * FROM 'escape';
```

utf8mb3Check

사전 확인 수준: 경고

utf8mb3 문자 세트 사용

MySQL 8.0에서는 `utf8mb3` 문자 세트가 더 이상 사용되지 않으며 향후 MySQL 메이저 버전에서 제거됩니다. 이 사전 확인은 이 문자 세트를 사용하는 데이터베이스 객체가 감지되는 경우 경고를 표시하기 위해 구현됩니다. 이렇게 해도 업그레이드가 차단되지는 않지만 테이블을 MySQL 8.0의 기본값인 `utf8mb4` 문자 세트로 마이그레이션하는 것이 좋습니다. utf8mb3 및 utf8mb4에 대한 자세한 내용은 MySQL 설명서의 3바이트와 4바이트 유니코드 문자 세트 간 변환을 참조하세요.

출력 예제:

```
{
  "id": "utf8mb3",
  "title": "Usage of utf8mb3 charset",
  "status": "OK",
  "description": "Warning: The following objects use the deprecated utf8mb3
 character set. It is recommended to convert them to use utf8mb4 instead, for
 improved Unicode support. The utf8mb3 character is subject to removal in the
 future.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-
utf8mb3.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.t1.col1",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
```

```
        },
        {
          "level": "Warning",
          "dbObject": "test.t1.col2",
          "description": "column's default character set: utf8",
          "dbObjectType": "Column"
        }
    ]
}
```

이 문제를 해결하기 위해 참조된 객체와 테이블을 다시 빌드합니다. 자세한 내용과 사전 요구 사항은 MySQL 설명서의 3바이트와 4바이트 유니코드 문자 세트 간 변환을 참조하세요.

zeroDatesCheck

사전 확인 수준: 경고

날짜, 날짜/시간 및 타임스탬프 값이 0임

MySQL은 이제 날짜, 날짜/시간 및 타임스탬프 열에 0 값을 사용하는 것과 관련하여 더 엄격한 규칙을 적용합니다. `NO_ZERO_IN_DATE` 및 `NO_ZERO_DATE` `SQL` 모드는 향후 MySQL 릴리스에서 `strict` 모드와 병합되므로 `strict` 모드와 함께 사용하는 것이 좋습니다.

사전 확인을 실행할 때 데이터베이스 연결에 대한 `sql_mode` 설정에 이러한 모드가 포함되지 않으면 사전 확인에 경고가 발생합니다. 사용자는 여전히 값이 0인 날짜, 날짜/시간 및 타임스탬프 값을 삽입할 수 있습니다. 하지만 향후 동작이 변경되어 제대로 작동하지 않을 수 있으므로 0 값을 유효한 값으로 바꾸는 것이 좋습니다. 이는 경고이므로 업그레이드를 차단하지는 않지만 조치를 취하기 위한 계획을 시작하는 것이 좋습니다.

출력 예제:

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
```

```
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
  ]
}
```

경고를 보고하는 Aurora MySQL 사전 확인

다음 사전 확인은 Aurora MySQL에만 적용됩니다.

- auroraUpgradeCheckForRollbackSegmentHistoryLength
- auroraUpgradeCheckForUncommittedRowModifications

auroraUpgradeCheckForRollbackSegmentHistoryLength

사전 확인 수준: 경고

클러스터의 롤백 세그먼트 기록 목록 길이가 높은지 확인합니다.

auroraUpgradeCheckForIncompleteXATransactions에서 언급한 바와 같이 메이저 버전 업그레이드 프로세스를 실행하는 동안 Aurora MySQL 버전 2 DB 인스턴스가 완전히 종료되는 것이 중요합니다. 이렇게 하면 모든 트랜잭션이 커밋되거나 롤백되고 InnoDB가 모든 실행 취소 로그 레코드를 제거하게 됩니다.

DB 클러스터의 롤백 세그먼트 기록 목록 길이(HLL)가 길면 InnoDB가 실행 취소 로그 레코드 제거를 완료하는 데 걸리는 시간이 길어져 메이저 버전 업그레이드 프로세스 중에 가동 중지 시간이 길어질 수 있습니다. 사전 확인에서 DB 클러스터의 HLL이 높음을 감지하면 경고가 발생합니다. 이렇게 해도 업그레이드가 차단되지는 않지만 DB 클러스터의 HLL을 면밀히 모니터링하는 것이 좋습니다. 이를 낮은 수준으로 유지하면 메이저 버전 업그레이드 중에 필요한 가동 중지 시간이 줄어듭니다. HLL 모니터링에 대한 자세한 내용은 InnoDB 기록 목록 길이가 크게 늘어남 섹션을 참조하세요.

출력 예제:


```
{
  "id": "auroraUpgradeCheckForRollbackSegmentHistoryLength",
  "title": "Checks if the rollback segment history length for the cluster is high",
  "status": "OK",
  "description": "Rollback Segment History length is greater than 1M. Upgrade may
 take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_metrics",
        "description": "The InnoDB undo history list length('trx_rseg_history_len')
 is 82989114. Upgrade may take longer due to purging of undo information for old row
 versions."
      }
  ]
}
```


사전 확인은 데이터베이스 클러스터(82989114)에서 InnoDB 실행 취소 HLL이 높음을 감지했기 때문에 경고를 반환합니다. 업그레이드는 진행되지만 제거를 위한 실행 취소의 양에 따라 업그레이드 프로세스 중에 필요한 가동 중지 시간이 연장될 수 있습니다.

프로덕션에서 업그레이드를 실행하기 전에 DB 클러스터에서 열린 트랜잭션을 조사하여 HLL이 관리 가능한 크기로 유지되도록 하는 것이 좋습니다.

auroraUpgradeCheckForUncommittedRowModifications

사전 확인 수준: 경고

커밋되지 않은 행 수정이 많은지 확인

auroraUpgradeCheckForIncompleteXATransactions에서 언급한 바와 같이 메이저 버전 업그레이드 프로세스를 실행하는 동안 Aurora MySQL 버전 2 DB 인스턴스가 완전히 종료되는 것이 중요합니다. 이렇게 하면 모든 트랜잭션이 커밋되거나 롤백되고 InnoDB가 모든 실행 취소 로그 레코드를 제거하게 됩니다.

DB 클러스터에 많은 수의 행을 수정한 트랜잭션이 있는 경우 완전한 종료 프로세스의 일부로 InnoDB이 트랜잭션의 롤백을 완료하는 데 걸리는 시간이 연장될 수 있습니다. 사전 확인에서 DB 클러스터에 많은 수의 수정된 행이 있는 장기 실행 트랜잭션을 찾으면 경고가 발생합니다. 이렇게 해도 업그레이드가 차단되지는 않지만 DB 클러스터의 활성 트랜잭션 크기를 면밀히 모니터링하는

것이 좋습니다. 이를 낮은 수준으로 유지하면 메이저 버전 업그레이드 중에 필요한 가동 중지 시간이 줄어듭니다.

출력 예제:

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "description": "Database contains uncommitted row changes greater than 10M.
 Upgrade may take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_trx",
        "description": "The database contains 11000000 uncommitted row change(s) in
 1 transaction(s). Upgrade may take longer due to transaction rollback."
      }
  ]
},
```

사전 확인은 DB 클러스터에 11,000,000개의 커밋되지 않은 행 변경 사항이 있는 트랜잭션이 포함되어 있음을 보고합니다. 이 트랜잭션은 완전한 종료 프로세스 중에 롤백해야 합니다. 업그레이드가 진행되지만 업그레이드 프로세스 중 가동 중지 시간을 줄이려면 프로덕션 클러스터에서 업그레이드를 실행하기 전에 이를 모니터링하고 조사하는 것이 좋습니다.

라이터 DB 인스턴스에서 활성 트랜잭션을 보려면 information_schema.innodb_trx 테이블을 사용할 수 있습니다. 라이터 DB 인스턴스에 대한 다음 쿼리는 DB 클러스터의 현재 트랜잭션, 실행 시간, 상태 및 수정된 행을 보여줍니다.

```
# Example of uncommitted transaction
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+------------------------------
+--------------------------------+----------+-----------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state | rows_modified |
```

```
+---------------------+-----------------------------
+------------------------------+-----------+-----------+---------------+
| 2024-08-12 18:32:52 |                         1592 |
 20041 | 52866130 | RUNNING   |      11000000 |
+---------------------+-----------------------------
+------------------------------+-----------+-----------+---------------+
1 row in set (0.01 sec)

# Example of transaction rolling back
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+--------------------------------+-----------+--------------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id    | trx_state    | rows_modified |
+---------------------+-----------------------------
+--------------------------------+-----------+--------------+---------------+
| 2024-08-12 18:32:52 |                         1719 |
 20041 | 52866130 | ROLLING BACK |      10680479 |
+---------------------+-----------------------------
+--------------------------------+-----------+--------------+---------------+
1 row in set (0.01 sec)
```

트랜잭션이 커밋되거나 롤백된 후에는 사전 확인이 더 이상 경고를 반환하지 않습니다. 대규모 트랜잭션을 롤백하기 전에 MySQL 설명서와 애플리케이션 팀에 문의하세요. 롤백이 완료되려면 트랜잭션 크기에 따라 시간이 걸릴 수 있습니다.

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "detectedProblems": []
},
```

InnoDB 트랜잭션 관리 최적화와 MySQL 데이터베이스 인스턴스에서 대규모 트랜잭션 실행 및 롤백의 잠재적 영향에 대한 자세한 내용은 MySQL 설명서의 InnoDB 트랜잭션 관리 최적화를 참조하세요.

고지 사항

다음 사전 확인은 사전 확인이 실패하면 알림을 생성하지만 업그레이드를 진행할 수 있습니다.

sqlModeFlagCheck

사전 확인 수준: 알림

더 이상 사용되지 않는 **sql_mode** 플래그 사용

MAXDB 외에도 DB2, MSSQL, MYSQL323, MYSQL40, ORACLE, POSTGRESQL, NO_FIELD_OPTIONS, NO_KEY_OPTIONS 및 NO_TABLE_OPTIONS 등의 다른 sql_mode 옵션이 제거되었습니다. MySQL 8.0부터 이러한 값 중 어느 것도 sql_mode 시스템 변수에 할당할 수 없습니다. 이 사전 확인에서 이러한 sql_mode 설정을 사용하여 열려 있는 세션을 찾으면 DB 인스턴스 및 DB 클러스터 파라미터 그룹과 클라이언트 애플리케이션 및 구성을 업데이트하여 비활성화해야 합니다. 자세한 내용은 MySQL 설명서를 참조하세요.

출력 예제:

```
{
  "id": "sqlModeFlagCheck",
  "title": "Usage of obsolete sql_mode flags",
  "status": "OK",
  "detectedProblems": []
}
```

이러한 사전 확인 실패를 해결하려면 maxdbFlagCheck를 참조하세요.

오류, 경고 또는 알림

다음 사전 확인은 사전 확인 출력에 따라 오류, 경고 또는 알림을 반환할 수 있습니다.

checkTableOutput

사전 확인 수준: 오류, 경고 또는 알림

check table x for upgrade 명령에서 보고되는 문제

Aurora MySQL 버전 3으로 업그레이드를 시작하기 전에 check table for upgrade는 DB 클러스터의 사용자 스키마에 있는 각 테이블에서 실행됩니다. 이 사전 확인은 checkTableMysqlSchema와 동일하지 않습니다.

`check table for upgrade` 명령은 최신 버전의 MySQL로 업그레이드하는 동안 발생할 수 있는 잠재적 문제가 있는지 테이블을 검사합니다. 업그레이드를 시도하기 전에 이 명령을 실행하면 비호환성을 미리 식별하고 해결하는 데 도움이 되므로 실제 업그레이드 프로세스가 더 원활해집니다.

이 명령은 각 테이블에 대해 다음과 같은 다양한 검사를 수행합니다.

- 테이블 구조 및 메타데이터가 대상 MySQL 버전과 호환되는지 확인
- 더 이상 사용되지 않거나 제거된 기능이 테이블에서 사용되고 있는지 확인
- 데이터 손실 없이 테이블을 올바르게 업그레이드할 수 있는지 확인

다른 사전 확인과 달리 `check table` 출력에 따라 오류, 경고 또는 알림을 반환할 수 있습니다. 이 사전 확인에서 테이블을 반환하는 경우 업그레이드를 시작하기 전에 반환 코드 및 메시지와 함께 테이블을 주의 깊게 검토합니다. 자세한 내용은 MySQL 설명서의 [CHECK TABLE 문]을 참조하세요.

여기서는 오류 예제와 경고 예제를 제공합니다.

오류 예제

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.parent",
        "description": "Table 'test.parent' doesn't exist"
      }
  ]
},
```

사전 확인은 `test.parent` 테이블이 존재하지 않는다는 오류를 보고합니다.

라이터 DB 인스턴스의 `mysql-error.log` 파일에 외래 키 오류가 있는 것으로 표시됩니다.

```
2024-08-13T15:32:10.676893Z 62 [Warning] InnoDB: Load table `test`.`parent` failed,
 the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try
 again.
```

```
2024-08-13T15:32:10.676905Z 62 [Warning] InnoDB: Cannot open table test/parent from
 the internal data dictionary of InnoDB though the .frm file for the table exists.
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
 for how to resolve the issue.
```

라이터 DB 인스턴스에 로그인하고 `show engine innodb status\G`를 실행하여 외래 키 오류에 대한 자세한 정보를 얻습니다.

```
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2024-08-13 15:33:33 0x14ef7b8a1700 INNODB MONITOR OUTPUT
=====================================
.
.
.
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2024-08-13 15:32:10 0x14ef6dbbb700 Error in foreign key constraint of table test/
child:
there is no index in referenced table which would contain
the columns as the first columns, or the data types in the
referenced table do not match the ones in table. Constraint:
,
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
The index in the foreign key in table is p_name_idx
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-
constraints.html for correct foreign key definition.
.
.
```

이 `LATEST FOREIGN KEY ERROR` 메시지는 `test.parent` 테이블을 참조하는 `test.child` 테이블의 fk_pname 외래 키 제약 조건에 누락된 인덱스 또는 데이터 유형 불일치가 있음을 보고합니다. [외래 키 제약 조건](http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-constraints.html)에 대한 MySQL 설명서에는 외래 키에서 참조되는 열에 연결된 인덱스가 있어야 하고 상위/하위 열은 동일한 데이터 유형을 사용해야 한다고 명시되어 있습니다.

누락된 인덱스 또는 데이터 유형 불일치와 관련이 있는지 확인하려면 데이터베이스에 로그인하고 세션 변수 foreign_key_checks를 일시적으로 비활성화하여 테이블 정의를 확인합니다. 이렇게 하면

문제의 하위 제약 조건(fk_pname)이 상위 테이블 name varchar(20) NOT NULL을 참조하는데 p_name varchar(20) CHARACTER SET latin1 DEFAULT NULL을 사용하는 것을 확인할 수 있습니다. 상위 테이블은 DEFAULT CHARSET=utf8을 사용하지만 하위 테이블의 p_name 열은 latin1을 사용하므로 데이터 유형 불일치 오류가 발생합니다.

```
mysql> show create table parent\G
ERROR 1146 (42S02): Table 'test.parent' doesn't exist

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> set foreign_key_checks=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show create table parent\G
*************************** 1. row ***************************
       Table: parent
Create Table: CREATE TABLE `parent` (
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

```
1 row in set (0.00 sec)
```

이 문제를 해결하기 위해 하위 테이블을 상위 테이블과 동일한 문자 세트를 사용하도록 변경하거나 상위 테이블을 하위 테이블과 동일한 문자 세트를 사용하도록 변경할 수 있습니다. 여기서 하위 테이블은 p_name 열 정의에서 명시적으로 `latin1`을 사용하기 때문에 `ALTER TABLE`을 실행하여 문자 세트를 `utf8`로 수정합니다.

```
mysql> alter table child modify p_name varchar(20) character set utf8 DEFAULT NULL;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)
```

이렇게 하면 사전 확인이 통과되고 업그레이드를 진행할 수 있습니다.

경고 예:

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.orders",
        "description": "Trigger test.orders.delete_audit_trigg does not have CREATED
 attribute."
      }
  ]
}
```

사전 확인은 CREATED 속성이 없기 때문에 `test.orders` 테이블에 `delete_audit_trigg` 트리거에 대한 경고를 보고합니다. MySQL 설명서의 버전 호환성 확인에 따르면 이 메시지는 정보 제공용이며 MySQL 5.7.2 이전에 생성된 트리거에 대해 표시됩니다.

이는 경고이므로 업그레이드를 계속 진행하지 못하도록 차단하지 않습니다. 그러나 문제를 해결하려면 해당 트리거를 다시 생성할 수 있으며 사전 확인이 성공하고 경고가 표시되지 않습니다.

```
{
  "id": "checkTableOutput",
```

```
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": []
},
```

현재 위치 업그레이드 수행 방법

Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식의 배경 자료를 검토하는 것이 좋습니다.

Aurora MySQL 클러스터에 대한 주 버전 업그레이드 계획에 설명된 대로 사전 업그레이드 계획 및 테스트를 수행합니다.

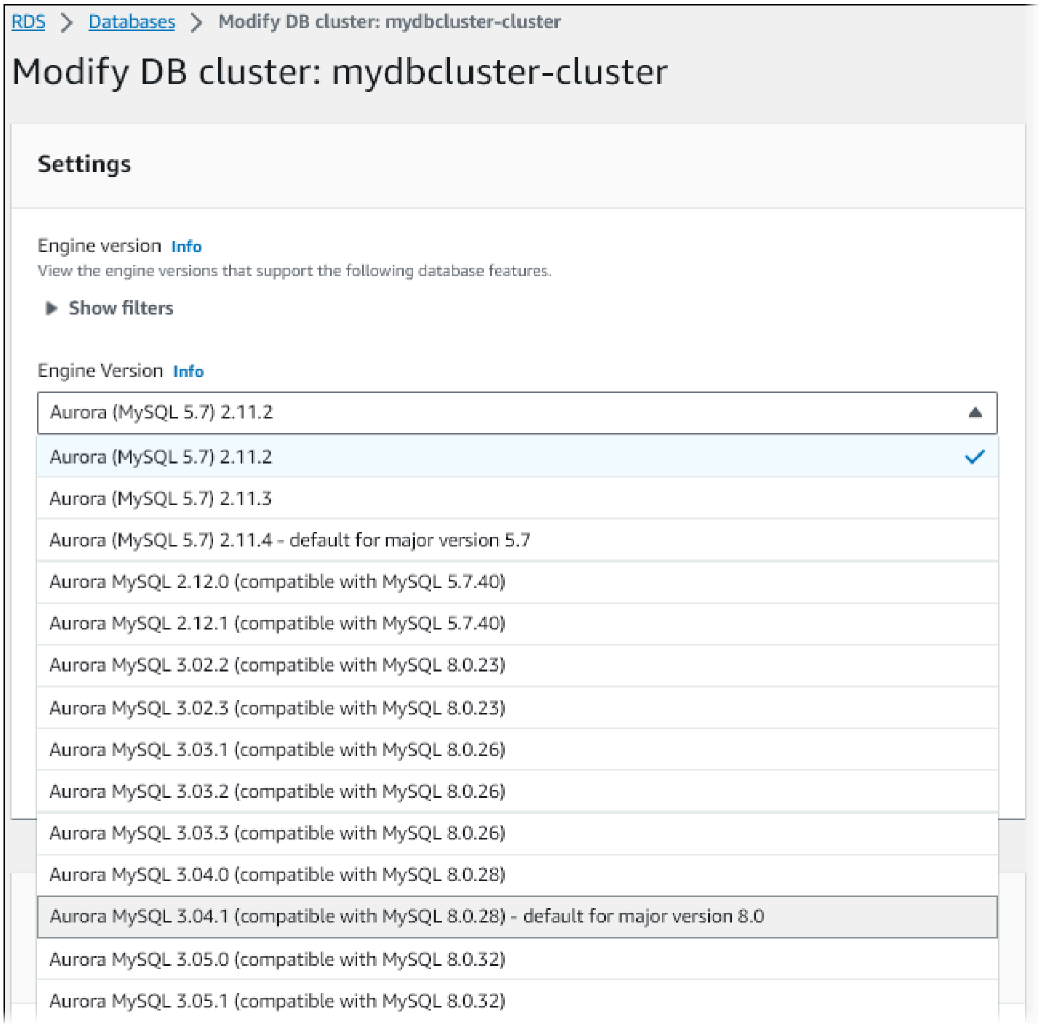
콘솔

다음 예제에서는 `mydbcluster-cluster` DB 클러스터를 Aurora MySQL 버전 3.04.1로 업그레이드합니다.

Aurora MySQL DB 클러스터의 주 버전을 업그레이드하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 원래 DB 클러스터에 사용자 지정 파라미터 그룹을 사용한 경우 새 메이저 버전과 호환되는 해당 파라미터 그룹을 생성합니다. 새 파라미터 그룹의 구성 파라미터를 필요에 따라 조정합니다. 자세한 내용은 현재 위치 업그레이드가 클러스터의 파라미터 그룹에 미치는 영향 섹션을 참조하세요.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 목록에서 수정할 DB 클러스터를 선택합니다.
5. 수정을 선택합니다.
6. 버전에 새 Aurora MySQL 메이저 버전을 선택합니다.

   일반적으로 메이저 버전의 최신 마이너 버전을 사용하는 것이 좋습니다. 여기에서는 현재의 기본 버전을 선택합니다.



7. [Continue]를 선택합니다.
8. 다음 페이지에서 업그레이드 수행 시기를 지정합니다. 다음 예약된 유지 관리 기간 중 또는 즉시를 선택합니다.
9. (선택 사항) 업그레이드하는 동안 RDS 콘솔에서 이벤트 페이지를 주기적으로 검사합니다. 이렇게 하면 업그레이드 진행 상황을 모니터링하고 모든 문제를 식별할 수 있습니다. 업그레이드에 문제가 발생하면 수행할 단계에 대한 Aurora MySQL 현재 위치 업그레이드를 위한 문제 해결를 참조하십시오.
10. 이 절차를 시작할 때 새 파라미터 그룹을 생성한 경우 사용자 지정 파라미터 그룹을 업그레이드된 클러스터와 연결합니다. 자세한 내용은 현재 위치 업그레이드가 클러스터의 파라미터 그룹에 미치는 영향 단원을 참조하십시오.

> **Note**
>
> 이 단계를 수행하려면 클러스터를 다시 시작하여 새 파라미터 그룹을 적용해야 합니다.

11. (선택 사항) 업그레이드 후 테스트를 완료한 후 Aurora이 업그레이드 시작 시 생성한 수동 스냅샷을 삭제합니다.

AWS CLI

Aurora MySQL DB 클러스터의 메이저 버전을 업그레이드하려면 다음 필수 파라미터와 함께 AWS CLI modify-db-cluster 명령을 사용합니다.

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` 또는 `--no-apply-immediately`

클러스터에서 사용자 지정 파라미터 그룹을 사용하는 경우 다음 옵션 중 하나 또는 모두를 포함합니다.

- `--db-cluster-parameter-group-name`클러스터가 사용자 지정 클러스터 파라미터 그룹을 사용하는 경우
- `--db-instance-parameter-group-name`클러스터의 인스턴스가 사용자 지정 DB 파라미터 그룹을 사용하는 경우

다음 예제에서는 `sample-cluster` DB 클러스터를 Aurora MySQL 버전 3.04.1로 업그레이드합니다. 업그레이드는 다음 유지 관리 기간을 기다리는 대신 즉시 수행됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
          --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
```

```
            --allow-major-version-upgrade ^
            --apply-immediately
```

다른 CLI 명령을 modify-db-cluster와 결합하여 업그레이드 수행 및 확인을 위한 자동화된 종단 간 프로세스를 생성할 수 있습니다. 자세한 정보와 지침은 Aurora MySQL 현재 위치 업그레이드 자습서 단원을 참조하십시오.

> **Note**
>
> 클러스터가 Aurora 글로벌 데이터베이스의 일부라면 현재 위치 업그레이드 절차가 약간 다릅니다. modify-global-cluster 명령 작업을 modify-db-cluster 대신 호출합니다. 자세한 내용은 글로벌 데이터베이스에 대한 현재 위치 메이저 업그레이드 섹션을 참조하세요.

RDS API

Aurora MySQL DB 클러스터의 메이저 버전을 업그레이드하려면 다음 필수 파라미터와 함께 RDS API 작업 ModifyDBCluster를 사용합니다.

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (`true` 또는 `false`로 설정)

> **Note**
>
> 클러스터가 Aurora 글로벌 데이터베이스의 일부라면 현재 위치 업그레이드 절차가 약간 다릅니다. ModifyGlobalCluster 작업을 `ModifyDBCluster` 대신 호출합니다. 자세한 내용은 글로벌 데이터베이스에 대한 현재 위치 메이저 업그레이드 섹션을 참조하세요.

현재 위치 업그레이드가 클러스터의 파라미터 그룹에 미치는 영향

Aurora 파라미터 그룹에는 MySQL 5.7 또는 8.0과 호환되는 클러스터에 대해 서로 다른 구성 설정 집합이 있습니다. 현재 위치 업그레이드를 수행할 때 업그레이드된 클러스터와 클러스터의 모든 인스턴스는 해당하는 클러스터 및 인스턴스 파라미터 그룹을 사용해야 합니다.

클러스터와 인스턴스는 기본 5.7 호환 파라미터 그룹을 사용할 수 있습니다. 그렇다면 업그레이드된 클러스터 및 인스턴스는 기본 8.0 호환 파라미터 그룹으로 시작합니다. 클러스터와 인스턴스가 사용자 지정 파라미터 그룹을 사용하는 경우 해당하는 8.0 호환 파라미터 그룹을 생성해야 합니다. 또한 업그레이드 프로세스 중에 이를 지정해야 합니다.

> **Note**
>
> 대부분의 파라미터 설정에서는 두 지점에서 사용자 정의 파라미터 그룹을 선택할 수 있습니다. 클러스터를 만들 때와 나중에 파라미터 그룹을 클러스터와 연결할 때입니다.
> 그러나 `lower_case_table_names` 파라미터에 대해 기본값이 아닌 설정을 사용하는 경우 미리 이 설정으로 사용자 정의 파라미터 그룹을 설정해야 합니다. 그런 다음, 클러스터를 생성하기 위해 스냅샷 복원을 수행하는 경우 파라미터 그룹을 지정합니다. `lower_case_table_names` 파라미터에 대한 모든 변경은 클러스터를 생성한 후에는 효력이 없습니다.
> Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 때 `lower_case_table_names`에 동일한 설정을 사용하는 것이 좋습니다.
> Aurora MySQL 기반 Aurora Global Database 사용 시 `lower_case_table_names` 파라미터가 활성화 되어 있는 경우 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수 없습니다. 사용할 수 있는 메서드에 대한 자세한 내용은 메이저 버전 업그레이드 섹션을 참조하세요.

> **Important**
>
> 업그레이드 프로세스 중에 사용자 지정 파라미터 그룹을 지정하는 경우 업그레이드가 완료된 후 클러스터를 수동으로 재부팅해야 합니다. 이렇게 하면 클러스터가 사용자 지정 파라미터 설정을 사용하기 시작합니다.

Aurora MySQL 버전 간의 클러스터 속성 변경

Aurora MySQL 버전 2에서 버전 3으로 업그레이드할 때는 Aurora MySQL 클러스터 및 DB 인스턴스를 설정하거나 관리하는 데 사용하는 애플리케이션 또는 스크립트를 변경해야 합니다.

또한 5.7 및 8.0 호환 클러스터에서 기본 파라미터 그룹 이름이 다르다는 사실을 설명하기 위해 파라미터 그룹을 조작하는 코드를 변경하세요. Aurora MySQL 버전 2 및 3 클러스터에 해당하는 기본 파라미터 그룹 이름은 각각 `default.aurora-mysql5.7` 및 `default.aurora-mysql8.0`입니다.

예를 들어 업그레이드 전에 클러스터에 적용되는 다음과 같은 코드가 있을 수 있습니다.

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

클러스터의 주 버전을 업그레이드한 후 다음과 같이 해당 코드를 수정합니다.

```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

글로벌 데이터베이스에 대한 현재 위치 메이저 업그레이드

Aurora 글로벌 데이터베이스의 경우 글로벌 데이터베이스 클러스터를 업그레이드합니다. Aurora는 자동으로 모든 클러스터를 동시에 업그레이드하고 모든 클러스터가 동일한 엔진 버전을 실행하도록 합니다. 이 요구 사항은 시스템 테이블, 데이터 파일 형식 등에 대한 변경 내용이 모든 보조 클러스터에 자동으로 복제되기 때문입니다.

Aurora MySQL 현재 위치 주 버전 업그레이드 작동 방식의 지침을 따르세요. 업그레이드할 대상을 지정할 때 포함된 클러스터 중에서 대신 글로벌 데이터베이스 클러스터를 선택해야 합니다.

AWS Management Console을 사용하는 경우 글로벌 데이터베이스(Global database) 역할이 있는 항목을 선택합니다.

DB identifier	Role	Engine	Engine version
global-cluster	Global database	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster1	Primary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-1	Writer instance	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster-2	Secondary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-2	Reader instance	Aurora MySQL	5.7.mysql_aurora.2.09.2

AWS CLI 또는 RDS API를 사용하는 경우에는 modify-global-cluster 명령 또는 ModifyGlobalCluster 작업을 호출하여 업그레이드 프로세스를 시작합니다. `modify-db-cluster` 또는 `ModifyDBCluster` 대신 이중 하나를 사용합니다.

> **Note**
>
> 해당 Aurora Global Database의 메이저 버전 업그레이드를 수행하는 동안에는 글로벌 데이터베이스 클러스터에 대한 사용자 지정 파라미터 그룹을 지정할 수 없습니다. 전역 클러스터의

각 리전에 사용자 지정 파라미터 그룹을 생성합니다. 그런 다음 업그레이드 후 리전 클러스터에 수동으로 적용합니다.

AWS CLI를 사용하여 Aurora MySQL 글로벌 데이터베이스 클러스터의 메이저 버전을 업그레이드하려면 다음 필수 파라미터와 함께 modify-db-cluster 명령을 사용합니다.

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

다음 예에서는 글로벌 데이터베이스 클러스터를 Aurora MySQL 버전 2.10.2로 업그레이드합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 5.7.mysql_aurora.2.10.2 \
          --allow-major-version-upgrade
```

Windows의 경우:

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 5.7.mysql_aurora.2.10.2 ^
          --allow-major-version-upgrade
```

역추적 고려 사항

업그레이드한 클러스터에서 역추적 기능을 사용하도록 설정한 경우 업그레이드된 클러스터를 업그레이드 전의 시간으로 되돌릴 수 없습니다.

Aurora MySQL 현재 위치 업그레이드 자습서

다음 Linux 예제에서는 AWS CLI를 사용하여 전체 업그레이드 절차의 일반적인 단계를 수행하는 방법을 보여 줍니다.

이 첫 번째 예시에서는 2.x 버전의 Aurora MySQL를 실행하는 Aurora DB 클러스터를 생성합니다. 클러스터에는 라이터 DB 인스턴스와 리더 DB 인스턴스가 포함됩니다. `wait db-instance-available` 명령은 라이터 DB 인스턴스를 사용할 수 있을 때까지 일시 중지됩니다. 이것이 바로 클러스터를 업그레이드할 준비가 된 시점입니다.

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.11.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

클러스터를 업그레이드할 수 있는 Aurora MySQL 3.x 버전은 클러스터가 현재 실행 중인 2.x 버전과 클러스터가 위치한 AWS 리전에 따라 다릅니다. `--output text`를 갖춘 첫 번째 명령은 사용 가능한 대상 버전만 표시합니다. 두 번째 명령은 응답의 전체 JSON 출력을 보여줍니다. 이 응답에서 `engine` 파라미터에 사용하는 `aurora-mysql` 값과 같은 세부 정보를 볼 수 있습니다. 또한 3.04.0으로의 업그레이드가 메이저 버전 업그레이드를 나타낸다는 사실을 알 수 있습니다.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.11.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.04.0",
    "Description": "Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
```

```
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false,
    "SupportsIntegrations": false
},
...
```

이 예에서는 클러스터에 유효한 업그레이드 대상이 아닌 대상 버전 번호를 입력하면 Aurora에서 업그레이드를 수행하지 않는 상황을 보여줍니다. Aurora는 또한 `--allow-major-version-upgrade` 파라미터를 포함하지 않으면 메이저 버전 업그레이드를 수행하지 않습니다. 이러면 애플리케이션 코드를 광범위하게 테스트하여 변경해야 할 가능성이 있는 업그레이드를 우연히 수행하게 될 수 없습니다.

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.11.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.11.2"
}
```

클러스터 및 관련 DB 인스턴스의 상태가 `upgrading`로 변경되려면 몇 분 정도 걸립니다. 클러스터와 DB 인스턴스의 버전 번호는 업그레이드가 완료된 경우에만 변경됩니다. 다시 말해 라이터 DB 인스턴스에 대한 `wait db-instance-available` 명령을 사용하여 업그레이드가 완료될 때까지 기다렸다가 계속 진행할 수 있습니다.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
```

```
upgrading 5.7.mysql_aurora.2.11.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

이 시점에서 클러스터의 버전 번호는 업그레이드에 지정된 번호와 일치합니다.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.04.0
```

앞의 예제에서는 `--apply-immediately` 파라미터를 지정하여 즉시 업그레이드를 수행했습니다. 클러스터가 사용 중일 것으로 예상되지 않는 편리한 시간에 업그레이드를 수행하도록 `--no-apply-immediately` 파라미터를 지정할 수 있습니다. 이러면 클러스터의 다음 유지 관리 기간 동안 업그레이드가 시작됩니다. 유지 관리 기간은 유지 관리 작업을 시작할 수 있는 기간을 정의합니다. 유지 관리 기간 동안 장기 실행 작업이 완료되지 않을 수 있습니다. 따라서 업그레이드에 시간이 오래 걸릴 것으로 예상되더라도 더 긴 유지 관리 시간을 정의할 필요는 없습니다.

다음 예시에서는 처음에 Aurora MySQL 버전 2.11.2를 실행하는 클러스터를 업그레이드합니다. `describe-db-engine-versions` 출력에서 `False` 및 `True` 값은 `IsMajorVersionUpgrade` 속성을 나타냅니다. 버전 2.11.2부터는 다른 2.* 버전으로 업그레이드할 수 있습니다. 이러한 업그레이드는 메이저 버전 업그레이드로 간주되지 않으므로 현재 위치 업그레이드가 필요하지 않습니다. 현재 위치 업그레이드는 목록에 표시된 3.* 버전으로 업그레이드할 때만 사용할 수 있습니다.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text
```

```
5.7.mysql_aurora.2.11.3 False
5.7.mysql_aurora.2.11.4 False
5.7.mysql_aurora.2.11.5 False
5.7.mysql_aurora.2.11.6 False
5.7.mysql_aurora.2.12.0 False
5.7.mysql_aurora.2.12.1 False
5.7.mysql_aurora.2.12.2 False
5.7.mysql_aurora.2.12.3 False
8.0.mysql_aurora.3.04.0 True
8.0.mysql_aurora.3.04.1 True
8.0.mysql_aurora.3.04.2 True
8.0.mysql_aurora.3.04.3 True
8.0.mysql_aurora.3.05.2 True
8.0.mysql_aurora.3.06.0 True
8.0.mysql_aurora.3.06.1 True
8.0.mysql_aurora.3.07.1 True

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

지정된 유지 관리 기간 없이 클러스터가 생성되면 Aurora에서 주중 임의의 요일을 선택합니다. 이 경우 `modify-db-cluster` 명령은 월요일에 제출됩니다. 따라서 유지 보수 기간을 화요일 아침으로 변경합니다. 모든 시간은 UTC 표준 시간대로 표시됩니다. `tue:10:00-tue:10:30` 기간은 태평양 표준시 오전 2시~2시 30분에 해당합니다. 유지 관리 기간의 변경 내용은 즉시 적용됩니다.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
```

```
]
```

다음 예시에서는 업그레이드로 생성된 이벤트에 대한 보고서를 가져오는 방법을 보여줍니다. `--duration` 인수는 이벤트 정보 검색 시간(분)을 나타냅니다. 기본적으로 `describe-events`는 마지막 시간의 이벤트만 반환하므로 이 인수가 필요합니다.

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
```

```
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
```

```
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
    ]
}
```

Aurora MySQL 주 버전 업그레이드 실패 원인 조사

자습서에서는 Aurora MySQL 버전 2에서 버전 3로 업그레이드하는 데 성공했습니다. 하지만 업그레이드가 실패했다면 그 이유를 알고 싶을 것입니다.

먼저 `describe-events` AWS CLI 명령을 사용하여 DB 클러스터 이벤트를 살펴볼 수 있습니다. 이 예제는 지난 10시간 동안의 `mydbcluster`에 대한 이벤트를 보여줍니다.

```
aws rds describe-events \
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

이 경우 업그레이드 사전 점검에 실패한 것이 원인입니다.

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
```

```
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Upgrading.Troubleshooting.html",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:24.373000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
    ]
}
```

문제의 정확한 원인을 진단하려면 라이터 DB 인스턴스에 대한 데이터베이스 로그를 검사합니다. Aurora MySQL 버전 3으로 업그레이드할 때 실패하면 라이터 인스턴스에 `upgrade-prechecks.log`라는 이름의 로그 파일이 포함됩니다. 이 예에서는 해당 로그의 존재를 감지한 다음 검사를 위해 로컬 파일로 다운로드하는 방법을 보여줍니다.

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text

error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

upgrade-prechecks.log 파일은 JSON 형식입니다. 다른 JSON 래퍼 내에서 JSON 출력을 인코딩하지 않도록 --output text 옵션을 사용하여 다운로드합니다. Aurora MySQL 버전 3 업그레이드의 경우 이 로그에는 항상 특정 정보 메시지지와 경고 메시지가 포함됩니다. 업그레이드가 실패하는 경우에만 오류 메시지가 포함됩니다. 업그레이드가 성공하면 로그 파일이 생성되지 않습니다.

이러한 모든 오류를 요약하고 관련 객체 및 설명 필드를 표시하려면 upgrade-prechecks.log 파일의 내용에 관한 grep -A 2 '"level": "Error"' 명령을 실행합니다. 그러기 위해 각 오류 줄과 그 뒤에 두 줄이 표시됩니다. 여기에는 해당 데이터베이스 객체의 이름과 문제 수정 방법에 대한 지침이 포함되어 있습니다.

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

이 예제에서는 문제가 되는 테이블에서 다음 SQL 명령을 실행하여 문제를 해결하거나, 누락된 인덱스 없이 테이블을 다시 생성할 수 있습니다.

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

그런 다음 업그레이드를 다시 시도하세요.

Aurora MySQL 현재 위치 업그레이드를 위한 문제 해결

다음 팁을 사용하면 Aurora MySQL 인플레이스 업그레이드와 관련된 문제를 해결하는 데 도움이 됩니다. 이러한 팁은 Aurora Serverless DB 클러스터에는 적용되지 않습니다.

현재 위치 업그레이드가 취소되거나 느린 이유	Effect	유지 관리 기간 내에 현재 위치 업그레이드를 완료할 수 있는 솔루션
관련 Aurora 크로스 리전 복제본이 아직 패치되지 않음	Aurora는 업그레이드를 취소합니다.	Aurora 크로스 리전 복제본을 업그레이드하고 다시 시도합니다.
클러스터에 준비된 상태의 XA 트랜잭션이 있습니다.	Aurora는 업그레이드를 취소합니다.	준비된 모든 XA 트랜잭션을 커밋하거나 롤백합니다.

현재 위치 업그레이드가 취소되거나 느린 이유	Effect	유지 관리 기간 내에 현재 위치 업그레이드를 완료할 수 있는 솔루션
클러스터가 데이터 정의 언어(DDL)문을 처리하고 있습니다.	Aurora는 업그레이드를 취소합니다.	모든 DDL문이 완료된 후 대기하고 업그레이드를 수행하기를 고려하십시오.
클러스터에 많은 행에 대해 커밋되지 않은 변경 사항이 있습니다.	업그레이드에 시간이 오래 걸릴 수 있습니다.	업그레이드 프로세스는 커밋되지 않은 변경 사항을 롤백합니다. 이 조건에 대한 표시기는 `TRX_ROWS_MODIFIED` 테이블 내 `INFORMATION_SCHEMA.INNODB_TRX` 의 값입니다. 모든 대규모 트랜잭션이 커밋되거나 롤백된 후에만 업그레이드를 수행하는 것이 좋습니다.
클러스터에 실행 취소 레코드 수가 많음	업그레이드에 시간이 오래 걸릴 수 있습니다.	커밋되지 않은 트랜잭션이 대량의 행에 영향을 미치지 않더라도 대량의 데이터가 포함될 수 있습니다. 예를 들어 대형 BLOB를 삽입할 수 있습니다. Aurora는 이러한 종류의 트랜잭션 활동에 대한 이벤트를 자동으로 감지하거나 생성하지 않습니다. 이 조건에 대한 지표는 HLL(기록 목록 길이)입니다. 업그레이드 프로세스는 커밋되지 않은 변경 사항을 롤백합니다. `SHOW ENGINE INNODB STATUS` SQL 명령의 출력에서 HLL을 확인하거나 다음 SQL 쿼리를 사용하여 직접 확인할 수 있습니다. `SELECT count FROM information_schema.innodb_metrics WHERE name = 'trx_rseg_history_len';` 또한 Amazon CloudWatch를 사용하여 `RollbackSegmentHistoryListLength` 지표를 모니터링할 수도 있습니다. HLL의 길이가 더 줄어든 후에 한하여 업그레이드 수행을 고려하세요.

<table>
<tr><th>현재 위치 업그레이드가 취소되거나 느린 이유</th><th>Effect</th><th>유지 관리 기간 내에 현재 위치 업그레이드를 완료할 수 있는 솔루션</th></tr>
<tr><td>클러스터가 큰 이진 로그 트랜잭션을 커밋하는 중입니다.</td><td>업그레이드에 시간이 오래 걸릴 수 있습니다.</td><td>업그레이드 프로세스는 이진 로그 변경 내용이 적용될 때까지 기다립니다. 이 기간 동안 더 많은 트랜잭션 또는 DDL문이 시작될 수 있으므로 업그레이드 프로세스가 느려질 수 있습니다.

클러스터가 이진 로그 복제 변경 내용을 생성하느라 바쁜 경우가 아니면 업그레이드 프로세스를 예약합니다. Aurora는 이 조건에 대한 이벤트를 자동으로 감지하거나 생성하지 않습니다.</td></tr>
<tr><td>파일 제거 또는 손상으로 인한 스키마 불일치</td><td>Aurora는 업그레이드를 취소합니다.</td><td>임시 테이블의 기본 스토리지 엔진을 MyISAM에서 InnoDB로 변경합니다. 다음 단계를 수행합니다.

1. <code>default_tmp_storage_engine</code> DB 파라미터를 <code>InnoDB</code>로 수정합니다.
2. DB 클러스터를 재부팅합니다.
3. 재부팅한 후 <code>default_tmp_storage_engine</code> DB 파라미터가 <code>InnoDB</code>로 설정되어 있는지 확인합니다. 다음 명령을 사용합니다.
<pre>show global variables like 'default_tmp_storage_engine';</pre>4. MyISAM 스토리지 엔진을 사용하는 임시 테이블을 만들지 마시기 바랍니다. 곧 업그레이드할 예정이므로 데이터베이스 워크로드를 일시 중지하고 새 데이터베이스 연결을 만들지 않는 것이 좋습니다.
5. 현재 위치 업그레이드를 다시 시도합니다.</td></tr>
</table>

<table>
<tr><th>현재 위치 업그레이드가 취소되거나 느린 이유</th><th>Effect</th><th>유지 관리 기간 내에 현재 위치 업그레이드를 완료할 수 있는 솔루션</th></tr>
<tr><td>마스터 사용자 삭제됨</td><td>Aurora는 업그레이드를 취소합니다.</td><td>

Important

마스터 사용자는 삭제하지 마세요.

하지만 어떤 이유로든 마스터 사용자를 삭제해야 하는 경우 다음 SQL 명령을 사용하여 복원하세요.

```
CREATE USER 'master_username '@'%'
 IDENTIFIED BY ' master_user_passwo
rd ' REQUIRE NONE PASSWORD EXPIRE
 DEFAULT ACCOUNT UNLOCK;

GRANT SELECT, INSERT, UPDATE, DELETE,
 CREATE, DROP, RELOAD, PROCESS,
 REFERENCES, INDEX, ALTER, SHOW
 DATABASES, CREATE TEMPORARY TABLES,
LOCK TABLES, EXECUTE, REPLICATION
 SLAVE, REPLICATION CLIENT, CREATE
 VIEW, SHOW VIEW, CREATE ROUTINE, ALTER
 ROUTINE, CREATE USER, EVENT,
TRIGGER, LOAD FROM S3, SELECT INTO
 S3, INVOKE LAMBDA, INVOKE SAGEMAKER
, INVOKE COMPREHEND ON *.* TO
 'master_username '@'%' WITH GRANT
 OPTION;
```

</td></tr>
</table>

업그레이드 사전 검사 실패를 유발하는 문제 해결에 대한 자세한 내용은 다음 블로그를 참조하세요.

- [Amazon Aurora MySQL 버전 2(MySQL 5.7 호환성 지원)에서 버전 3(MySQL 8.0 호환성 지원)으로 업그레이드할 때 필요한 체크리스트, 1부]
- [Amazon Aurora MySQL 버전 2(MySQL 5.7 호환성 지원)에서 버전 3(MySQL 8.0 호환성 지원)으로 업그레이드할 때 필요한 체크리스트, 2부]

다음 단계를 사용하여 위 테이블의 일부 조건에 대해 직접 검사를 수행할 수 있습니다. 이렇게 하면 데이터베이스가 업그레이드가 성공적으로 신속하게 완료될 수 있는 상태임을 알고 있을 때 업그레이드를 예약할 수 있습니다.

- XA RECOVER문을 실행하여 열린 XA 트랜잭션을 확인할 수 있습니다. 그런 다음 업그레이드를 시작하기 전에 XA 트랜잭션을 커밋하거나 롤백할 수 있습니다.
- SHOW PROCESSLIST문을 실행하고 출력에서 CREATE, DROP, ALTER, RENAME 및 TRUNCATE 문을 찾아 DDL 문을 확인할 수 있습니다. 업그레이드를 시작하기 전에 모든 DDL 문이 완료될 수 있습니다.
- INFORMATION_SCHEMA.INNODB_TRX 테이블을 쿼리하여 수행되지 않은 행의 총 수를 확인할 수 있습니다. 테이블은 각 트랜잭션에 대한 하나의 행을 포함합니다. TRX_ROWS_MODIFIED 열은 트랜잭션에 의해 수정되거나 삽입된 행 수를 포함합니다.
- SHOW ENGINE INNODB STATUS SQL문을 실행하고 출력에서 History list length를 찾아 InnoDB 히스토리 목록의 길이를 확인할 수 있습니다. 다음 쿼리를 실행하여 값을 직접 확인할 수도 있습니다.

```
SELECT count FROM information_schema.innodb_metrics WHERE name =
 'trx_rseg_history_len';
```

  기록 목록의 길이는 다중 버전 동시성 제어 (MVCC)를 구현하기 위해 데이터베이스에 저장된 실행 취소 정보의 양에 해당합니다.

Aurora MySQL 버전 3에 대한 업그레이드 후 정리

Aurora MySQL 버전 2 클러스터를 Aurora MySQL 버전 3으로 업그레이드한 후 다음과 같은 기타 정리 작업을 수행할 수 있습니다.

- 모든 사용자 지정 파라미터 그룹의 MySQL 8.0 호환 버전을 새로 만듭니다. 필요한 모든 사용자 파라미터 값을 새 파라미터 그룹에 적용합니다.
- CloudWatch 경보, 설정 스크립트 등을 업데이트하여 이름이 포괄적 언어 변경의 영향을 받은 모든 표지에 새 이름을 사용합니다. 이런 지표 목록은 Aurora MySQL 버전 3에 대한 포괄적 언어 변경 섹션을 참조하세요.
- 모든 AWS CloudFormation 템플릿을 업데이트하여 이름이 포괄적 언어 변경의 영향을 받은 모든 구성 파라미터에 새 이름을 사용합니다. 이런 파라미터 전체 목록은 Aurora MySQL 버전 3에 대한 포괄적 언어 변경 섹션을 참조하세요.

공간 인덱스

Aurora MySQL 버전 3으로 업그레이드한 후 공간 인덱스와 관련된 객체 및 인덱스를 삭제하거나 다시 생성해야 하는지 확인합니다. MySQL 8.0 이전에 Aurora는 공간 리소스 식별자(SRID)를 포함하지 않은 인덱스를 사용하여 공간 쿼리를 최적화할 수 있었습니다. Aurora MySQL 버전 3은 SRID를 포함하는 공간 인덱스만 사용합니다. 업그레이드 중에 Aurora는 SRID 없는 모든 공간 인덱스를 자동으로 삭제하고 데이터베이스 로그에 경고 메시지를 인쇄합니다. 이러한 경고 메시지가 표시되면 업그레이드 후 SRID를 사용하여 새 공간 인덱스를 생성합니다. MySQL 8.0의 공간 함수와 데이터 유형 변경에 대한 자세한 내용은 MySQL 참조 설명서의 MySQL 8.0의 변경 사항을 참조하세요.

Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 및 수정

Amazon Aurora MySQL 호환 버전 릴리스 정보에서 다음 내용을 확인할 수 있습니다.

- Amazon Aurora MySQL 버전 3에 대한 데이터베이스 엔진 업데이트
- Amazon Aurora MySQL 버전 2에 대한 데이터베이스 엔진 업데이트
- Amazon Aurora MySQL 버전 1에 대한 데이터베이스 엔진 업데이트(지원 중지)
- Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그
- Amazon Aurora MySQL에서 수정된 보안 취약성

Amazon Aurora PostgreSQL 작업

Amazon Aurora PostgreSQL은 Amazon Aurora의 속도, 안정성 및 관리 용이성과 오픈 소스 데이터베이스의 단순성 및 비용 효율성을 결합한 완전관리형 PostgreSQL 호환 및 ACID 호환 관계형 데이터베이스 엔진입니다. Aurora PostgreSQL은 PostgreSQL을 대신하는 제품으로, 새 PostgreSQL 및 기존 PostgreSQL 배포를 간편하고 비용 효율적으로 설정, 운영 및 확장할 수 있으므로 비즈니스 및 애플리케이션에 집중할 수 있습니다. 일반적인 Aurora에 대해 자세히 알아보려면 Amazon Aurora이란 무엇인가요? 단원을 참조하세요.

Aurora PostgreSQL은 Aurora의 이점 외에도 기존 RDS for PostgreSQL 애플리케이션을 Aurora PostgreSQL로 변환하는 푸시 버튼 마이그레이션 도구를 통해 Amazon RDS에서 Aurora로의 편리한 마이그레이션 경로를 제공합니다. 프로비저닝, 패치 적용, 백업, 복구, 장애 감지 및 복구와 같은 일상적인 데이터베이스 작업도 Aurora PostgreSQL로 쉽게 관리할 수 있습니다.

Aurora PostgreSQL는 다수의 산업 표준에서 작동할 수 있습니다. 예를 들어 Aurora PostgreSQL 데이터베이스를 사용하여 HIPAA 준수 애플리케이션을 구축하고 AWS와 체결한 비즈니스 제휴 계약(BAA)에 따라 보호 대상 건강 정보(PHI)를 비롯한 의료 관련 정보를 저장할 수 있습니다.

Aurora PostgreSQL은 FedRAMP HIGH 사용 자격이 있습니다. AWS 및 규정 준수 활동에 대한 자세한 내용은 규정 준수 프로그램 제공 AWS 범위 내 서비스 섹션을 참조하세요.

> **Important**
>
> Aurora PostgreSQL DB 클러스터에 문제가 발생하면 AWS Support 에이전트가 데이터베이스 상태에 대한 추가 정보가 필요할 수 있습니다. AWS Support는 가능한 한 가장 빠른 시간 내에 필요한 정보를 최대한 많이 확보하는 것이 목표입니다.
> PG 컬렉터를 사용하여 통합 HTML 파일에서 중요한 데이터베이스 정보를 수집할 수 있습니다. PG 컬렉터, 실행 방법 및 HTML 보고서 다운로드 방법에 대한 자세한 내용은 PG 컬렉터를 참조하세요.
> 스크립트가 성공적으로 완료되고 달리 명시되지 않는 한, 스크립트는 읽기 가능한 HTML 형식으로 출력을 반환합니다. 스크립트는 업무에 영향을 미칠 수 있는 모든 데이터 또는 보안 세부 정보를 HTML에서 제외하도록 설계되었습니다. 또한 스크립트는 데이터베이스나 해당 환경을 수정하지 않습니다. 하지만 HTML에서 공유하기에 부적합한 정보를 발견한 경우, HTML을 업로드하기 전에 문제가 있는 정보를 얼마든지 제거해도 됩니다. HTML이 허용되는 경우, 지원 사례의 사례 세부 정보에 있는 첨부 파일 섹션을 사용하여 업로드합니다.

주제

데이터베이스 미리 보기 환경 작업

PostgreSQL 커뮤니티는 매년 새로운 메이저 PostgreSQL 버전을 릴리스합니다. 마찬가지로 Amazon Aurora는 PostgreSQL 메이저 버전을 미리 보기 릴리스로 제공합니다. 이를 통해 사용자는 미리 보기 버전을 사용하여 DB 클러스터를 만들고 데이터베이스 미리 보기 환경에서 관련 기능을 테스트할 수 있습니다.

데이터베이스 미리 보기 환경의 Aurora PostgreSQL DB 클러스터는 다른 Aurora PostgreSQL DB 클러스터와 기능적으로 유사합니다. 그러나 프로덕션에는 미리 보기 버전을 사용할 수 없습니다.

이 방식을 사용하는 경우 다음과 같은 중요 제한 사항에 유의하십시오.

- 모든 DB 인스턴스 및 DB 클러스터는 생성 60일 후 백업 및 스냅샷과 함께 삭제됩니다.
- Amazon VPC 서비스 기반의 Virtual Private Cloud(VPC)에서만 DB 인스턴스를 생성할 수 있습니다.
- DB 인스턴스의 스냅샷을 프로덕션 환경으로 복제할 수 없습니다.

미리 보기에서 지원되는 옵션은 다음과 같습니다.

- DB 인스턴스는 r5, r6g, r6i, r7g, r7i, r8g, x2g, t3, t4g 인스턴스 유형으로만 만들 수 있습니다. 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하세요.
- 단일 AZ 배포와 다중 AZ 배포를 모두 사용할 수 있습니다.
- 표준 PostgreSQL 덤프 및 로드 함수를 사용하여 데이터베이스 미리 보기 환경에서 데이터베이스를 내보내거나 데이터베이스 미리 보기 환경으로 데이터베이스를 가져올 수 있습니다.

지원되는 DB 인스턴스 클래스 유형

Amazon Aurora PostgreSQL은 미리 보기 영역에서 다음 DB 인스턴스 클래스를 지원합니다.

메모리 최적화 클래스

- db.r5 – 메모리 최적화 인스턴스 클래스
- db.r6g – AWS Graviton2 프로세서로 구동되는 메모리 최적화 인스턴스 클래스
- db.r6i – 메모리 최적화 인스턴스 클래스
- db.r7g – AWS Graviton3 프로세서로 구동되는 메모리 최적화 인스턴스 클래스
- db.r7i – 메모리 최적화 인스턴스 클래스
- db.r8g – AWS Graviton4 프로세서로 구동되는 메모리 최적화 인스턴스 클래스
- db.x2g - AWS Graviton2 프로세서로 구동되는 메모리 최적화 인스턴스 클래스

> **Note**
> 인스턴스 클래스 목록에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

버스트 가능 클래스

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.xlarge

미리 보기 환경에서 지원하지 않는 기능

다음 기능은 미리 보기 환경에서 사용할 수 없습니다.

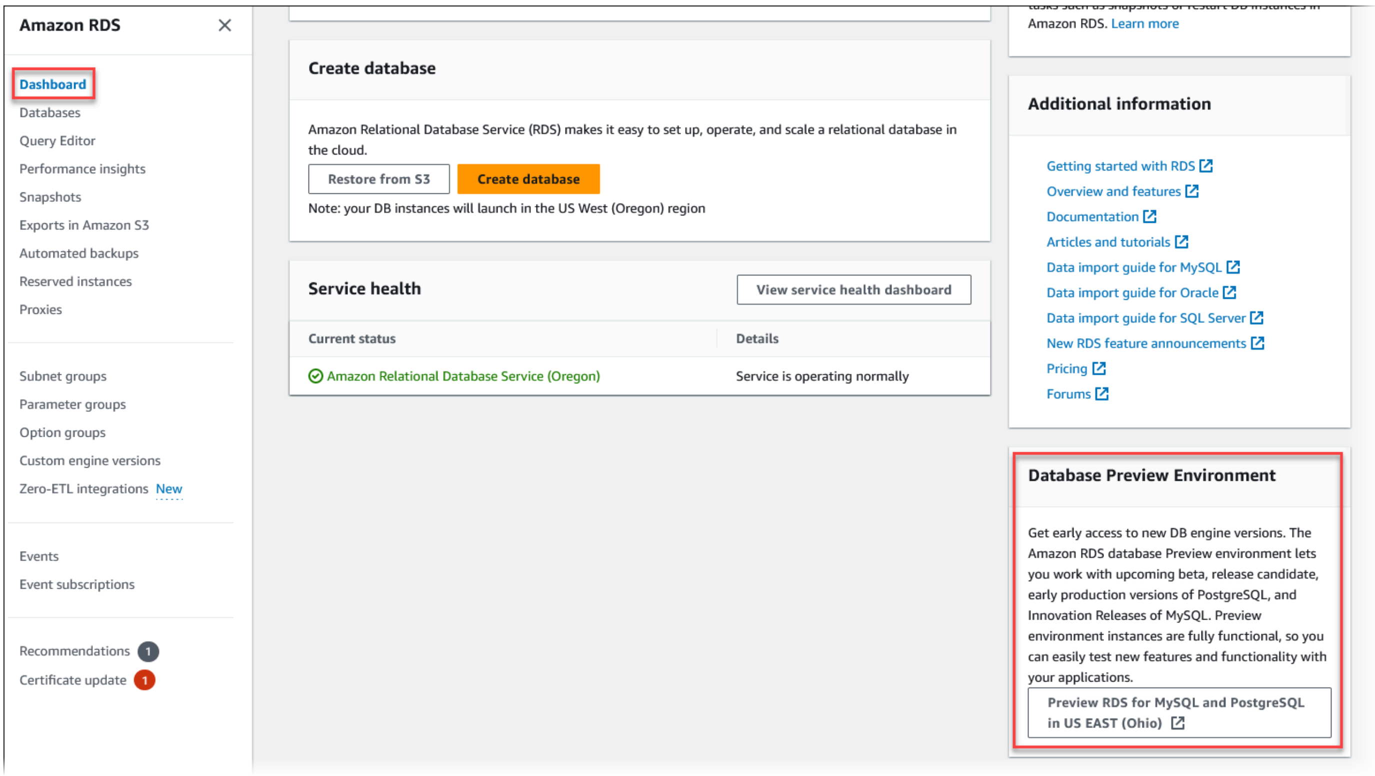
- Aurora Serverless v1 및 v2
- 메이저 버전 업그레이드(MVU)
- 미리 보기 영역에서는 새로운 마이너 버전이 릴리스되지 않음
- RDS for PostgreSQL에서 Aurora PostgreSQL으로의 인바운드 복제
- Amazon RDS 블루/그린 배포
- 리전 간 스냅샷 복제
- Aurora Global Database
- 데이터베이스 활동 스트림(DAS), RDS 프록시 및 AWS DMS
- Auto Scaling 읽기 전용 복제본
- AWS Bedrock
- RDS 내보내기
- 성능 개선 도우미
- 사용자 지정 엔드포인트
- 스냅샷 복사
- 제로 ETL
- Babelfish
- 모든 확장

미리 보기 환경에서 새 DB 클러스터 생성

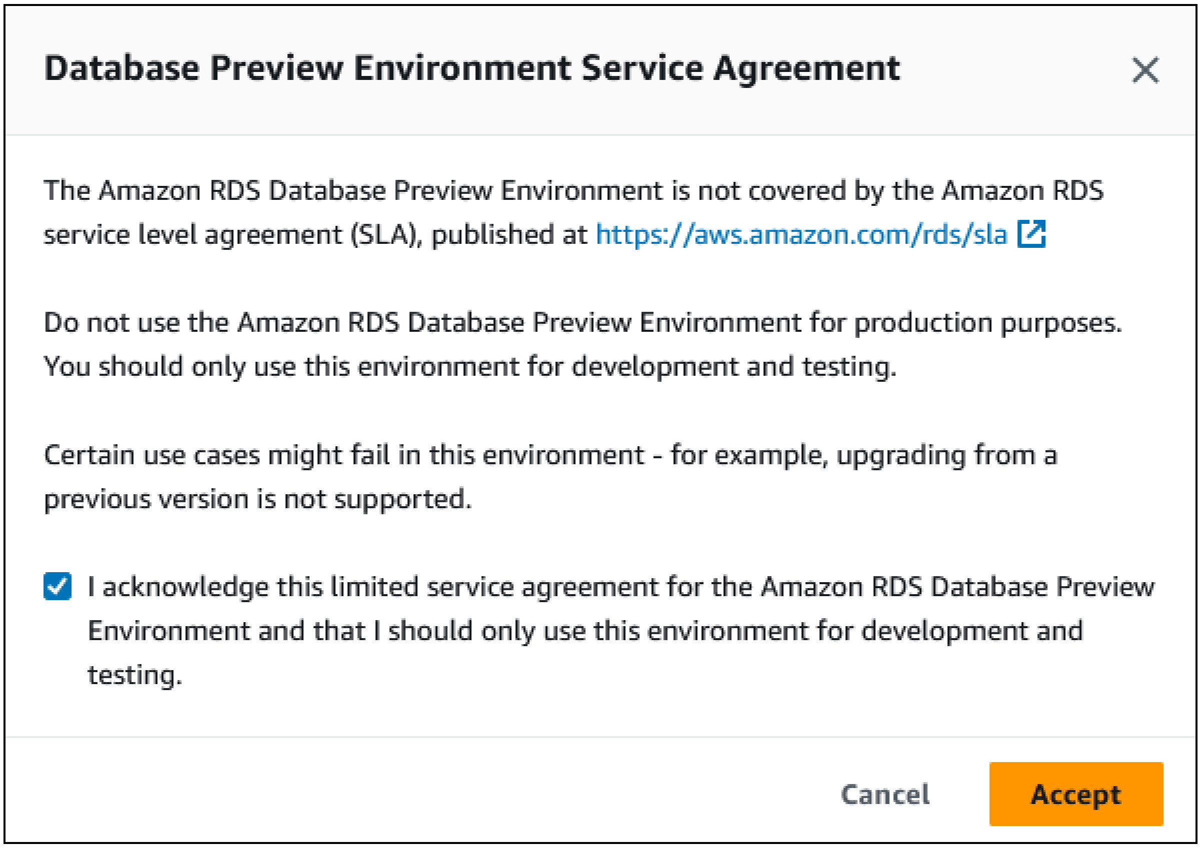
다음 절차를 이용하여 미리 보기 환경에서 DB 클러스터를 새로 만듭니다.

미리 보기 환경에서 DB 클러스터를 생성하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 대시보드를 선택합니다.
3. 다음 그림과 같이 Dashboard(대시보드) 페이지에서 Database Preview Environment(데이터베이스 미리 보기 환경) 섹션을 찾습니다.



Database preview environment(데이터베이스 미리 보기 환경)으로 바로 이동할 수도 있습니다. 계속하려면 먼저 제한 사항을 확인하고 수락해야 합니다.



4. Aurora PostgreSQL DB 클러스터를 생성하려면 Aurora DB 클러스터를 생성할 때와 동일한 프로세스를 따르면 됩니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

Aurora API 또는 AWS CLI를 사용하여 데이터베이스 미리 보기 환경에 인스턴스를 생성하려면 다음 엔드포인트를 사용합니다.

```
rds-preview.us-east-2.amazonaws.com
```

데이터베이스 미리 보기 환경의 PostgreSQL 버전 17

⚠ 이 문서는 Aurora PostgreSQL 버전 17의 미리 보기 설명서입니다. 내용은 변경될 수 있습니다.

이제 PostgreSQL 버전 17.0을 Amazon RDS Database Preview Environment에서 사용할 수 있습니다. PostgreSQL 버전 17에는 여러 개선 사항이 포함되어 있으며, 이에 관한 설명은 다음 PostgreSQL 설명서에 나와 있습니다.

- PostgreSQL 17 릴리스됨

데이터베이스 미리 보기 환경에 대한 자세한 내용은 데이터베이스 미리 보기 환경 작업 섹션을 참조하세요. 콘솔에서 미리 보기 환경에 액세스하려면 https://console.aws.amazon.com/rds-preview/를 선택합니다.

Amazon Aurora PostgreSQL를 사용한 보안

Aurora 보안에 대한 일반적인 개요는 Amazon Aurora의 보안 섹션을 참조하세요. 다음과 같은 몇 가지 수준에서 Amazon Aurora PostgreSQL의 보안을 관리할 수 있습니다.

- Aurora PostgreSQL DB 클러스터 및 DB 인스턴스에서 Amazon RDS 관리 작업을 수행할 수 있는 사용자를 제어하려면 AWS Identity and Access Management(IAM)를 사용합니다. IAM은 사용자가 서비스에 액세스하기 전에 사용자 자격 증명 인증을 처리합니다. 또한 권한 부여, 즉 사용자가 원하는 작업을 수행할 수 있는지도 처리합니다. IAM 데이터베이스 인증은 Aurora PostgreSQL DB 클러스터를 생성할 때 선택할 수 있는 추가 인증 방법입니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.

  Aurora PostgreSQL DB 클러스터에서 IAM을 사용하는 경우 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 열기 전에 먼저 IAM 보안 인증 정보를 사용하여 AWS Management Console에 로그인합니다.
- Aurora DB 클러스터는 Amazon VPC 서비스를 기반으로 Virtual Private Cloud(VPC)에 생성해야 합니다. Aurora DB 클러스터에서 DB 인스턴스의 엔드포인트 및 포트에 연결할 수 있는 디바이스 또는 Amazon EC2 인스턴스를 제어하려면 VPC 보안 그룹을 사용합니다. 이 엔드포인트와 포트는 보안 소켓 계층(SSL) 방식으로 연결할 수 있습니다. 그 밖에도 기업의 방화벽 규칙을 통해 기업에서 이용하는 디바이스의 DB 인스턴스 연결 여부를 제어하는 것도 가능합니다. VPC에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 단원을 참조하세요.

  지원되는 VPC 테넌시는 Aurora PostgreSQL DB 클러스터에서 사용하는 DB 인스턴스 클래스에 따라 다릅니다. `default` VPC 테넌시일 경우 DB 클러스터가 공유 하드웨어에서 실행됩니다. `dedicated` VPC 테넌시일 때는 DB 클러스터가 전용 하드웨어 인스턴스에서 실행됩니다. 확장 가능 성능 DB 인스턴스 클래스는 기본 VPC 테넌시만 지원합니다. 확장 성능 DB 인스턴스 클래스에는 db.t3 및 db.t4g DB 인스턴스 클래스가 포함됩니다. 다른 모든 Aurora PostgreSQL DB 인스턴스 클래스는 기본 및 전용 VPC 테넌시를 모두 지원합니다.

  인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하세요. `default` 및 `dedicated` VPC 테넌시에 대한 자세한 정보는 Amazon Elastic Compute Cloud 사용 설명서의 전용 인스턴스를 참조하세요.
- Amazon Aurora DB 클러스터에서 실행 중인 PostgreSQL 데이터베이스에 사용 권한을 부여하려면 PostgreSQL의 독립 실행형 인스턴스와 동일한 일반적인 방법을 사용할 수 있습니다. `CREATE`

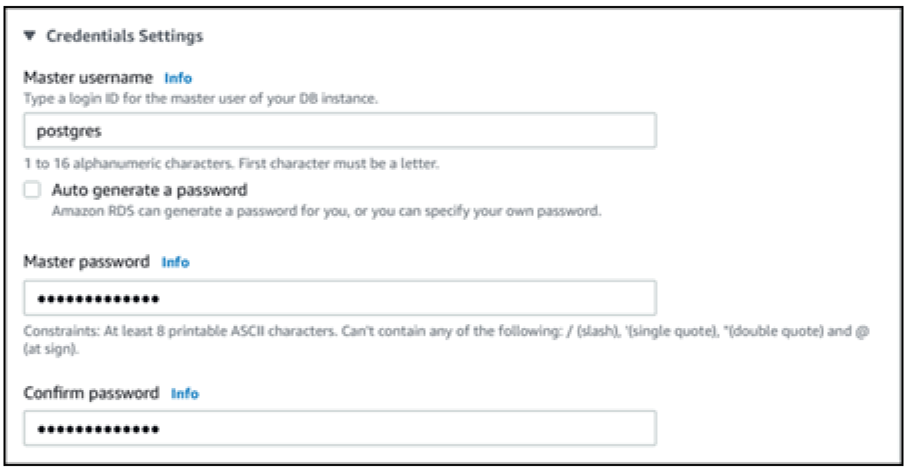
`ROLE`, `ALTER ROLE`, `GRANT`, `REVOKE` 등의 명령은 온프레미스 데이터베이스에서 작동하는 것과 마찬가지로 작동하며, 데이터베이스와 스키마, 테이블을 직접 수정할 때도 동일합니다.

PostgreSQL은 역할을 사용하여 권한을 관리합니다. `rds_superuser` 역할은 Aurora PostgreSQL DB 클러스터에서 가장 많은 권한을 가진 역할입니다. 이 역할은 자동으로 생성되며 DB 클러스터를 생성하는 사용자(마스터 사용자 계정, 기본적으로 `postgres`)에게 부여됩니다. 자세한 내용은 PostgreSQL 역할 및 권한 이해를 참조하세요.

버전 10, 11, 12, 13, 14 이상의 릴리스를 비롯하여 사용 가능한 모든 Aurora PostgreSQL 버전은 메시지 다이제스트(MD5)의 대안으로 암호에 대한 Salted Challenge Response Authentication Mechanism(SCRAM)을 지원합니다. SCRAM은 MD5보다 안전하기 때문에 SCRAM을 사용하는 것이 좋습니다. MD5에서 SCRAM으로 데이터베이스 사용자 암호를 마이그레이션하는 방법을 비롯한 자세한 내용은 SCRAM for PostgreSQL 암호 암호화 사용 섹션을 참조하세요.

PostgreSQL 역할 및 권한 이해

AWS Management Console을 사용하여 Aurora PostgreSQL DB 클러스터 를 만들면 관리자 계정이 동시에 생성됩니다. 기본적으로 해당 이름은 다음 스크린샷에 나와 있는 것처럼 `postgres`입니다.



기본값(`postgres`)을 그대로 사용하지 않고 다른 이름을 선택할 수 있습니다. 이 경우 선택한 이름은 문자로 시작해야 하며 영숫자 1~16자 사이이어야 합니다. 단순하게 하기 위해 이 안내서 전체에서 기본 사용자 계정을 기본값(`postgres`)으로 참조합니다.

AWS Management Console 대신 `create-db-cluster` AWS CLI를 사용하는 경우 명령에서 `master-username` 파라미터로 사용자 이름을 전달하여 사용자 이름을 생성합니다. 자세한 내용은 2단계: Aurora PostgreSQL DB 클러스터 생성 단원을 참조하세요.

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하든, 기본 postgres 이름을 사용하든, 다른 이름을 선택하든 관계없이 첫 번째 데이터베이스 사용자 계정은 rds_superuser 그룹의 멤버이며 rds_superuser 권한을 가집니다.

주제

rds_superuser 역할 이해

PostgreSQL에서는 역할로 데이터베이스의 다양한 객체에 대해 그룹 또는 사용자에게 부여된 사용자, 그룹 또는 특정 권한 집합을 정의할 수 있습니다. CREATE USER 및 CREATE GROUP에 대한 PostgreSQL 명령은 데이터베이스 사용자를 구분하기 위한 특정 속성을 가진 보다 일반적인 CREATE ROLE로 대체되었습니다. 데이터베이스 사용자는 LOGIN 권한을 가진 역할로 간주할 수 있습니다.

> **Note**
>
> CREATE USER 및 CREATE GROUP 명령을 계속 사용할 수 있습니다. 자세한 내용은 PostgreSQL 설명서에서 데이터베이스 역할을 참조하세요.

postgres 사용자는 Aurora PostgreSQL DB 클러스터에서 가장 높은 권한을 지닌 데이터베이스 사용자입니다. 이는 다음 CREATE ROLE 문으로 정의되는 특성을 가지고 있습니다.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

속성 NOSUPERUSER, NOREPLICATION, INHERIT 및 VALID UNTIL 'infinity'는 별도로 지정하지 않는 한 CREATE ROLE의 기본 옵션입니다.

기본적으로 postgres에는 rds_superuser 역할에 부여된 권한과 역할 및 데이터베이스를 생성할 수 있는 권한이 있습니다. rds_superuser 역할이 있으면 postgres 사용자는 다음과 같은 작업을 할 수 있습니다.

- Aurora PostgreSQL을 참조하세요. 자세한 내용은 확장 및 외부 데이터 래퍼 작업 단원을 참조하세요.

- 사용자에 대한 역할을 생성하고 사용자에게 권한을 부여합니다. 자세한 내용은 PostgreSQL 설명서에서 CREATE ROLE 및 GRANT를 참조하세요.
- 데이터베이스를 생성합니다. 자세한 내용은 PostgreSQL 설명서에서 CREATE DATABASE를 참조하세요.
- 이러한 권한이 없는 사용자 역할에 `rds_superuser` 권한을 부여하고 필요에 따라 권한을 회수합니다. 이 역할은 수퍼유저 태스크를 수행하는 사용자에게만 부여하는 것이 좋습니다. 즉, 데이터베이스 관리자(DBA) 또는 시스템 관리자에게 이 역할을 부여할 수 있습니다.
- `rds_superuser` 역할이 없는 데이터베이스 사용자에게 `rds_replication` 역할을 부여(회수)합니다.
- `rds_superuser` 역할이 없는 데이터베이스 사용자에게 `rds_password` 역할을 부여(회수)합니다.
- `pg_stat_activity` 보기를 사용하여 모든 데이터베이스 연결에 대한 상태 정보를 가져옵니다. 필요한 경우 `rds_superuser`는 `pg_terminate_backend` 또는 `pg_cancel_backend`를 사용하여 연결을 중지할 수 있습니다.

`CREATE ROLE postgres...` 문에서 `postgres` 사용자 역할이 특히 PostgreSQL `superuser` 권한을 허용하지 않음을 알 수 있습니다. Aurora PostgreSQL은 관리형 서비스이므로 호스트 OS에 액세스할 수 없으며, PostgreSQL `superuser` 계정을 사용하여 연결할 수 없습니다. 독립 실행형 PostgreSQL에 대한 `superuser` 액세스 권한이 필요한 대부분의 태스크는 Aurora에서 자동으로 관리됩니다.

권한 부여에 대한 자세한 내용은 PostgreSQL 설명서에서 GRANT를 참조하세요.

`rds_superuser` 역할은 Aurora PostgreSQL DB 클러스터에서 미리 정의된 여러 역할 중 하나입니다.

> **Note**
>
> PostgreSQL 13 및 이전 릴리스에서는 미리 정의된 역할을 기본 역할이라고 합니다.

다음 목록에서 새로운 Aurora PostgreSQL DB 클러스터에 대해 자동으로 생성되는 미리 정의된 다른 역할 중 일부를 확인할 수 있습니다. 미리 정의된 역할 및 권한은 변경할 수 없습니다. 미리 정의된 역할에 대한 권한은 삭제하거나 이름을 바꾸거나 수정할 수 없습니다. 이를 시도할 시에는 오류가 발생합니다.

- rds_password - 데이터베이스 사용자를 대상으로 암호를 변경하고 암호 제약 조건을 설정할 수 있는 역할입니다. `rds_superuser` 역할에는 기본적으로 이 역할이 부여되며, 이 역할을 통해 데이터베

이스 사용자에게 역할을 부여할 수 있습니다. 자세한 내용은 PostgreSQL 데이터베이스에 대한 사용자 액세스 제어 섹션을 참조하세요.

- 14 이전의 RDS for PostgreSQL 버전에서 `rds_password` 역할은 암호를 변경하고 데이터베이스 사용자와 `rds_superuser` 역할이 있는 사용자에 대해 암호 제한을 설정할 수 있습니다. 14 이후의 RDS for PostgreSQL 버전에서 `rds_password` 역할은 데이터베이스 사용자에 대해서만 암호를 변경하고 암호 제약 조건을 설정할 수 있습니다. `rds_superuser` 역할이 있는 사용자만 `rds_superuser` 역할을 가진 다른 사용자에게 이러한 작업을 수행할 수 있습니다.

- rdsadmin - `superuser` 권한이 있는 관리자가 독립 실행형 PostgreSQL 데이터베이스에서 수행할 수 있는 많은 관리 태스크를 처리하기 위해 생성된 역할입니다. 이 역할은 Aurora PostgreSQL에서 다양한 관리 태스크에 내부적으로 사용됩니다.

역할 및 해당 권한 보기

PostgreSQL 버전에 따라 서로 다른 명령을 사용하여 RDS for PostgreSQL DB 인스턴스에서 사전 정의된 역할과 해당 권한을 볼 수 있습니다.

PostgreSQL 15 이하 버전의 경우

RDS for PostgreSQL DB 인스턴스에 연결하고 psql에서 \du 명령을 사용합니다.

```
postgres=> \du
                                                              List of roles
    Role name     |                         Attributes                         |
                  Member of
-----------------+-----------------------------------------------------------
+------------------------------------------------------
 postgres        | Create role, Create DB                                     +|
 {rds_superuser}
                 | Password valid until infinity                              |
 rds_ad          | Cannot login                                               | {}
 rds_iam         | Cannot login                                               | {}
 rds_password    | Cannot login                                               | {}
 rds_replication | Cannot login                                               | {}
 rds_superuser   | Cannot login                                               |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                              |
```

PostgreSQL 16 이상 버전의 경우

PostgreSQL 16부터 자세한 역할 멤버십 정보를 보려면 `\drg+` 명령을 사용합니다.

```
postgres=> \drg+
                             List of role grants
   Role name    |          Member of           |       Options        | Grantor
----------------+------------------------------+----------------------+----------
 postgres       | rds_superuser                | INHERIT, SET         | rdsadmin
 rds_superuser  | pg_checkpoint                | ADMIN, INHERIT, SET  | rdsadmin
 rds_superuser  | pg_monitor                   | ADMIN, INHERIT, SET  | rdsadmin
 rds_superuser  | pg_signal_backend            | ADMIN, INHERIT, SET  | rdsadmin
 rds_superuser  | pg_use_reserved_connections  | ADMIN, INHERIT, SET  | rdsadmin
 rds_superuser  | rds_password                 | ADMIN, INHERIT, SET  | rdsadmin
 rds_superuser  | rds_replication              | ADMIN, INHERIT, SET  | rdsadmin
```

PostgreSQL 버전에 관계없이 역할 멤버십을 확인하려면 다음 SQL 쿼리를 사용할 수 있습니다.

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

출력에서 `rds_superuser`는 데이터베이스 사용자 역할은 아니지만(로그인할 수 없음), 다른 많은 역할의 권한을 가집니다. 데이터베이스 사용자 `postgres`가 `rds_superuser` 역할의 멤버임을 알 수도 있습니다. 앞서 언급한 바와 같이 `postgres`는 Amazon RDS 콘솔 데이터베이스 생성(Create database) 페이지의 기본값입니다. 다른 이름을 선택한 경우 해당 이름이 대신 역할 목록에 표시됩니다.

> **Note**
>
> Aurora PostgreSQL 버전 15.2 및 14.7에는 `rds_superuser` 역할의 제한적인 동작이 도입되었습니다. 사용자에게 `rds_superuser` 역할이 부여된 경우에도 연결하려면 Aurora PostgreSQL 사용자에게는 해당 데이터베이스에 대한 `CONNECT` 권한이 부여되어야 합니다. Aurora PostgreSQL 버전 14.7 및 15.2 이전에는 사용자에게 `rds_superuser` 역할이 부여되어 있어야 모든 데이터베이스 및 시스템 테이블에 연결할 수 있었습니다. 이러한 제한적인 동작을 통해 AWS 및 Amazon Aurora는 지속적으로 보안을 강화하고자 최선을 다하고 있습니다. 위의 개선 사항으로 영향을 받는 경우 애플리케이션에서 관련 로직을 업데이트하세요.

PostgreSQL 데이터베이스에 대한 사용자 액세스 제어

PostgreSQL의 새 데이터베이스는 항상 모든 데이터베이스 사용자와 역할이 객체를 만들 수 있도록 허용하는 데이터베이스 `public` 스키마의 기본 권한 집합으로 생성됩니다. 이러한 권한을 통해 데이터베이스 사용자는 데이터베이스에 연결하고, 가령 연결된 동안 임시 테이블을 만들 수 있습니다.

Aurora PostgreSQL DB 클러스터 프라이머리 노드에 생성한 데이터베이스 인스턴스에 액세스할 수 있는 사용자 권한을 보다 효과적으로 제어하려면 이러한 기본 `public` 권한을 회수하는 것이 좋습니다. 이렇게 하고 나서 다음 절차에 표시된 것처럼 데이터베이스 사용자에게 보다 세분화된 구체적인 권한을 부여하면 됩니다.

새 데이터베이스 인스턴스에 대한 역할 및 권한을 설정하는 방법

새로 생성한 Aurora PostgreSQL DB 클러스터에 데이터베이스를 설정하여 여러 연구자가 사용할 수 있도록 하고, 이들 모두 데이터베이스에 대한 읽기-쓰기 액세스 권한이 필요하다고 가정합니다.

1. 다음과 같이 `psql` 또는 pgAdmin을 사용하여 Aurora PostgreSQL DB 클러스터의 프라이머리 DB 인스턴스 에 연결합니다.

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --port=5432 --username=postgres --password
   ```

   메시지가 표시되면 암호를 입력합니다. `psql` 클라이언트는 기본 관리 연결 데이터베이스인 `postgres=>`를 연결하고 프롬프트로 표시합니다.

2. 데이터베이스 사용자가 `public` 스키마에서 객체를 생성하지 못하도록 하려면 다음을 수행합니다.

   ```
   postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
   REVOKE
   ```

3. 다음으로 새 데이터베이스 인스턴스를 생성합니다.

   ```
   postgres=> CREATE DATABASE lab_db;
   CREATE DATABASE
   ```

4. 새 데이터베이스의 `PUBLIC` 스키마에서 모든 권한을 회수합니다.

   ```
   postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
   REVOKE
   ```

5. 데이터베이스 사용자를 위한 역할을 생성합니다.

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. 이 역할이 있는 데이터베이스 사용자에게 데이터베이스 연결 기능을 제공합니다.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. lab_tech 역할이 있는 모든 사용자에게 이 데이터베이스에 대한 모든 권한을 부여합니다.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. 다음과 같이 데이터베이스 사용자를 생성합니다.

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. 다음과 같이 두 사용자에게 lab_tech 역할과 관련된 권한을 부여합니다.

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

이 시점에서 lab_user1과 lab_user2는 lab_db 데이터베이스에 연결할 수 있습니다. 이 예제에서는 여러 데이터베이스 인스턴스와 다양한 스키마 생성, 제한된 권한 부여를 포함할 수 있는 엔터프라이즈 사용에 대한 모범 사례를 따르지 않습니다. 전체 정보 및 추가 시나리오를 알아보려면 PostgreSQL 사용자 및 역할 관리를 참조하세요.

PostgreSQL 데이터베이스의 권한에 대한 자세한 내용은 PostgreSQL 설명서에서 GRANT 명령을 참조하세요.

사용자 암호 관리 위임 및 제어

DBA로서 사용자 암호 관리를 위임할 수 있습니다. 또는 데이터베이스 사용자가 암호를 변경하거나 암호 사용 주기와 같은 암호 제약 조건을 재구성하지 못하도록 할 수 있습니다. 선택한 데이터베이스 사

용자만 암호 설정을 변경할 수 있도록 하려면 제한된 암호 관리 기능을 켜면 됩니다. 이 기능을 활성화하면 rds_password 역할이 부여된 데이터베이스 사용자만 암호를 관리할 수 있습니다.

> **Note**
> 제한된 암호 관리를 사용하려면 Aurora PostgreSQL DB 클러스터가 Amazon Aurora PostgreSQL 10.6 이상을 실행해야 합니다.

기본적으로 이 기능은 다음과 같이 off입니다.

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

이 기능을 켜려면 사용자 정의 파라미터 그룹을 사용하고 rds.restrict_password_commands에 대한 설정을 1로 변경합니다. Aurora PostgreSQL의 프라이머리 DB 인스턴스 를 재부팅해야 설정이 적용됩니다.

이 기능을 활성화하면 다음 SQL 명령에 대한 rds_password 권한이 필요합니다.

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

암호에서 MD5 해시 알고리즘을 사용하는 경우에도 역할 이름 변경(ALTER ROLE myrole RENAME TO newname)이 제한됩니다.

이 기능이 활성화된 상태에서 rds_password 역할 권한 없이 이러한 SQL 명령을 시도하면 다음 오류가 발생합니다.

```
ERROR: must be a member of rds_password to alter passwords
```

암호 관리에만 사용하는 몇 가지 역할에만 `rds_password` 권한을 부여하는 것이 좋습니다. `rds_superuser` 권한이 없는 데이터베이스 사용자에게 `rds_password` 권한을 부여할 경우 `CREATEROLE` 속성도 부여해야 합니다.

만료 및 클라이언트 측에 필요한 복잡성 등의 암호 요구 사항을 확인해야 합니다. 암호 관련 변경 사항에 대해 자체 클라이언트 측 유틸리티를 사용하는 경우 유틸리티가 `rds_password`의 멤버여야 하며 `CREATE ROLE` 권한을 가져야 합니다.

SCRAM for PostgreSQL 암호 암호화 사용

Salted Challenge Response Authentication Mechanism(SCRAM)은 암호를 암호화하는 데 사용하는 PostgreSQL의 기본 메시지 다이제스트(MD5) 알고리즘을 대체합니다. SCRAM 인증 메커니즘은 MD5보다 더 안전한 것으로 간주됩니다. 이러한 2가지 암호 보호 방법에 대한 자세한 내용은 PostgreSQL 설명서의 [암호 인증]을 참조하세요.

MD5가 아닌 SCRAM을 Aurora PostgreSQL DB 클러스터의 암호 암호화 체계로 사용하는 것이 좋습니다. SCRAM은 Aurora PostgreSQL 버전 10 이상의 모든 메이저 및 마이너 버전에서 지원됩니다. 이는 암호 인증 및 암호화에 scram-sha-256 알고리즘을 사용하는 암호화 챌린지-응답 메커니즘입니다.

SCRAM을 지원하려면 클라이언트 애플리케이션의 라이브러리를 업데이트해야 할 수 있습니다. 예를 들어, 42.2.0 이전의 JDBC 버전은 SCRAM을 지원하지 않습니다. 자세한 내용은 PostgreSQL JDBC 드라이버 설명서의 [PostgreSQL JDBC 드라이버]를 참조하세요. 기타 PostgreSQL 드라이버 및 SCRAM 지원 목록은 PostgreSQL 설명서의 [드라이버 목록]을 참조하세요.

Aurora PostgreSQL 버전 14 이상에서는 기본적으로 새 DB 클러스터에 대한 암호 암호화를 위해 scram-sha-256을 지원합니다. 이 버전에서 기본 DB 클러스터 파라미터 그룹(`default.aurora-postgresql14`)의 `password_encryption` 값은 scram-sha-256으로 설정됩니다. SCRAM은 Aurora Serverless v1에서 지원되지 않습니다.

SCRAM이 필요하도록 Aurora PostgreSQL DB 클러스터 설정

Aurora PostgreSQL 14.3 이상 버전의 경우 Aurora PostgreSQL DB 클러스터가 scram-sha-256 알고리즘을 사용하는 암호만 수락하도록 요구할 수 있습니다.

> **Important**
>
> PostgreSQL 데이터베이스를 사용하는 기존 RDS 프록시의 경우, SCRAM만 사용하도록 데이터베이스 인증을 수정하면 최대 60초 동안 프록시를 사용할 수 없게 됩니다. 문제를 방지하려면 다음 중 하나를 수행합니다.

> - 데이터베이스에서 SCRAM 및 MD5 인증이 모두 허용되는지 확인합니다.
> - SCRAM 인증만 사용하려면 새 프록시를 생성하고, 애플리케이션 트래픽을 새 프록시로 마이그레이션한 다음 이전에 데이터베이스와 연결된 프록시를 삭제합니다.

시스템을 변경하기 전에 다음과 같이 전체 프로세스를 이해해야 합니다.

- 모든 데이터베이스 사용자에 대한 전체 역할 및 암호 암호화에 대한 정보를 가져옵니다.
- 암호 암호화를 제어하는 파라미터에 대해 Aurora PostgreSQL DB 클러스터 의 파라미터 설정을 다시 확인합니다.
- Aurora PostgreSQL DB 클러스터 에서 기본 파라미터 그룹을 사용한다면 필요한 경우 파라미터를 수정할 수 있도록 사용자 지정 DB 클러스터 파라미터 그룹 을 생성하고, 이를 Aurora PostgreSQL DB 클러스터 에 적용해야 합니다. Aurora PostgreSQL DB 클러스터 에서 사용자 지정 파라미터 그룹을 사용하는 경우 필요에 따라 과정 후반부에서 필요한 파라미터를 수정할 수 있습니다.
- `password_encryption` 파라미터를 `scram-sha-256`으로 변경합니다.
- 모든 데이터베이스 사용자에게 암호를 업데이트해야 함을 알립니다. `postgres` 계정에도 동일한 작업을 수행합니다. 새 암호는 scram-sha-256 알고리즘을 통해 암호화되어 저장됩니다.
- 모든 암호가 암호화 유형으로 암호화되었는지 확인합니다.
- 모든 암호가 scram-sha-256을 사용하는 경우 `rds.accepted_password_auth_method` 파라미터를 `md5+scram`에서 `scram-sha-256`으로 변경할 수 있습니다.

> **Warning**
>
> `rds.accepted_password_auth_method`를 scram-sha-256으로만 변경하면 md5로 암호화된 암호가 있는 사용자(역할)는 연결할 수 없습니다.

Aurora PostgreSQL DB 클러스터에서 SCRAM을 사용하도록 준비

Aurora PostgreSQL DB 클러스터 를 변경하기에 모든 기존 데이터베이스 사용자 계정을 확인해야 합니다. 또한 암호에 사용되는 암호화 유형도 확인하세요. `rds_tools` 확장을 사용하여 이러한 작업을 수행할 수 있습니다. `rds_tools`를 지원하는 PostgreSQL 버전을 확인하려면 [Amazon RDS for PostgreSQL의 확장 버전](#)을 참조하세요.

데이터베이스 사용자(역할) 및 암호 암호화 방법 목록을 가져오려면

1. 다음과 같이 psql을 사용하여 Aurora PostgreSQL DB 클러스터의 프라이머리 인스턴스 에 연결합니다.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. rds_tools 확장을 설치합니다.

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. 역할 및 암호화 목록을 가져옵니다.

```
postgres=> SELECT * FROM 
      rds_tools.role_password_encryption_type();
```

출력은 다음과 비슷합니다.

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

사용자 지정 DB 클러스터 파라미터 그룹 생성

> Note
>
> Aurora PostgreSQL DB 클러스터 에서 이미 사용자 지정 파라미터 그룹을 사용하는 경우 새로 만들 필요가 없습니다.

Aurora 파라미터 그룹에 대한 개요는 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 섹션을 참조하세요.

암호에 사용되는 암호 암호화 유형은 하나의 파라미터 `password_encryption`으로 설정됩니다. Aurora PostgreSQL DB 클러스터 에서 허용하는 암호화는 다른 파라미터인 `rds.accepted_password_auth_method`로 설정됩니다. 이러한 값 중 하나를 기본값에서 변경하려면 사용자 지정 DB 클러스터 파라미터 그룹 을 생성하여 클러스터 에 적용해야 합니다.

AWS Management Console 또는 RDS API를 사용하여 사용자 지정 DB 클러스터 파라미터 그룹 을 생성할 수도 있습니다. 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 단원을 참조하세요.

DB 파라미터 그룹을 DB 인스턴스에 연결합니다.

사용자 지정 DB 클러스터 파라미터 그룹을 생성하려면

1. `create-db-cluster-parameter-group` CLI 명령을 사용하여 클러스터에 대한 사용자 지정 파라미터 그룹을 생성합니다. 이 예에서는 `aurora-postgresql13`을 사용자 지정 파라미터 그룹의 소스로 사용할 수 있습니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
  --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
```

   Windows의 경우:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
  --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
```

   다음으로 사용자 지정 파라미터 그룹을 클러스터와 연결합니다.

2. `modify-db-cluster` CLI 명령을 사용하여 다음과 같이 사용자 지정 파라미터 그룹을 Aurora PostgreSQL DB 클러스터에 적용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
        --db-cluster-parameter-group-name "docs-lab-scram-passwords
```

Windows의 경우:

```
aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
        --db-cluster-parameter-group-name "docs-lab-scram-passwords
```

사용자 지정 DB 클러스터 파라미터 그룹이 포함된 Aurora PostgreSQL DB 클러스터 를 다시 동기화하려면 클러스터의 프라이머리 및 기타 모든 인스턴스를 재부팅합니다.

SCRAM을 사용하도록 암호 암호화 구성

Aurora PostgreSQL DB 클러스터 에서 사용하는 암호 암호화 메커니즘은 DB 클러스터 파라미터 그룹에 `password_encryption` 파라미터로 설정되어 있습니다. 허용되는 값은 설정하지 않거나 md5 또는 `scram-sha-256`입니다. 기본값은 다음과 같이 Aurora PostgreSQL 버전에 따라 달라집니다.

- Aurora PostgreSQL 14 – 기본값은 `scram-sha-256`입니다.
- Aurora PostgreSQL 13 – 기본값은 md5입니다.

사용자 지정 DB 클러스터 파라미터 그룹 이 Aurora PostgreSQL DB 클러스터 에 연결되어 있으면 암호 암호화 파라미터 값을 변경할 수 있습니다.

Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

암호의 암호화 설정을 scram-sha-256으로 변경하려면

- 다음과 같이 암호의 암호화 값을 scram-sha-256으로 변경합니다. 파라미터가 동적이기 때문에 변경 사항을 즉시 적용할 수 있으므로 변경 사항을 적용하려고 재시작하지 않아도 됩니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
  'docs-lab-scram-passwords' --parameters
 'ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate'
```

Windows의 경우:

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
  "docs-lab-scram-passwords" --parameters
 "ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate"
```

사용자 역할의 암호를 SCRAM으로 마이그레이션

다음에 설명된 대로 사용자 역할에 대한 암호를 SCRAM으로 마이그레이션할 수 있습니다.

MD5에서 SCRAM으로 데이터베이스 사용자(역할) 암호를 마이그레이션하려면

1. 다음과 같이 관리자 사용자로 로그인(기본 사용자 이름 `postgres`)합니다.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 다음 명령을 사용하여 RDS for PostgreSQL DB 인스턴스의 `password_encryption` 파라미터 설정을 확인합니다.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. 이 파라미터의 값을 scram-sha-256으로 변경합니다. 이 파라미터는 동적 파라미터이므로 변경 후에 인스턴스를 재부팅할 필요가 없습니다. 다음과 같이 `scram-sha-256`으로 설정되어 있는지 값을 다시 확인합니다.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
```

```
scram-sha-256
(1 row)
```

4. 모든 데이터베이스 사용자에게 암호 변경을 알립니다. 계정 `postgres`(`rds_superuser` 권한을 지닌 데이터베이스 사용자)의 자체 암호도 변경해야 합니다.

```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

5. Aurora PostgreSQL DB 클러스터에 모든 데이터베이스에 대해 프로세스를 반복합니다.

SCRAM이 필요하도록 파라미터 변경

과정의 마지막 단계입니다. 다음 절차에서 변경한 후에도 암호에 md5 암호화를 계속 사용하는 모든 사용자 계정(역할)은 Aurora PostgreSQL DB 클러스터에 로그인할 수 없습니다.

`rds.accepted_password_auth_method`는 로그인하는 동안 Aurora PostgreSQL DB 클러스터가 사용자 암호에 대해 허용하는 암호화 방법을 지정합니다. 기본값은 `md5+scram`이므로, 두 방법 중 하나를 사용할 수 있습니다. 다음 이미지에서는 이 파라미터에 대한 기본 설정을 찾을 수 있습니다.

Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

이 파라미터에 허용되는 값은 `md5+scram` 또는 `scram`입니다. 이 파라미터 값을 `scram`으로 변경하여 필수로 설정합니다.

암호에 SCRAM 인증을 요구하도록 파라미터 값을 변경하려면

1. Aurora PostgreSQL DB 클러스터 의 모든 데이터베이스에 대한 전체 데이터베이스 사용자 암호가 암호 암호화에 `scram-sha-256`을 사용하는지 확인합니다. 이렇게 하려면 다음과 같이 역할(사용자) 및 암호화 유형에 대해 `rds_tools`를 쿼리합니다.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname          | encryption_type
  -----------------------+-----------------
  pg_monitor             |
```

```
pg_read_all_settings |
pg_read_all_stats    |
pg_stat_scan_tables  |
pg_signal_backend    |
lab_tester           | scram-sha-256
user_465             | scram-sha-256
postgres             | scram-sha-256
( rows)
```

2. Aurora PostgreSQL DB 클러스터의 모든 DB 인스턴스에서 쿼리를 반복합니다.

   모든 암호가 scram-sha-256을 사용하는 경우 그대로 진행하면 됩니다.

3. 다음과 같이 허용된 암호 인증의 값을 scram-sha-256으로 변경합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

   Windows의 경우:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

SSL/TLS를 이용한 Aurora PostgreSQL 데이터 보안

Amazon RDS는 Aurora PostgreSQL DB 클러스터를 위한 보안 소켓 계층(SSL) 및 전송 계층 보안(TLS) 암호화를 지원합니다. SSL/TLS를 사용하여 애플리케이션과 Aurora PostgreSQL DB 클러스터 사이의 연결을 암호화할 수 있습니다. 또한 Aurora PostgreSQL DB 클러스터에 대한 모든 연결에서 SSL/TLS를 사용하도록 지정할 수도 있습니다. Amazon Aurora PostgreSQL은 전송 계층 보안(TLS) 버전 1.1 및 1.2를 지원합니다. TLS 1.2를 암호화 연결에 사용하는 것이 좋습니다. 다음 Aurora PostgreSQL 버전에서 TLSv1.3에 대한 지원을 추가했습니다.

- 15.3 이상의 모든 버전
- 14.8 이상의 14 버전

- 13.11 이상의 13 버전
- 12.15 이상의 12 버전
- 11.20 이상의 11 버전

SSL/TLS 지원 및 PostgreSQL 데이터베이스에 대한 일반적인 정보는 PostgreSQL 설명서의 SSL 지원을 참조하세요. JDBC를 통한 SSL/TLS 연결 사용에 대한 자세한 내용은 PostgreSQL 설명서의 클라이언트 구성을 참조하세요.

주제

- Aurora PostgreSQL DB 클러스터에 대한 SSL/TLS 연결 요구
- SSL/TLS 연결 상태 확인
- Aurora PostgreSQL DB 클러스터 연결을 위한 암호 그룹 구성

Aurora PostgreSQL의 모든 AWS 리전에서 SSL/TLS 지원 기능을 사용할 수 있습니다. Amazon RDS는 DB 클러스터가 생성될 때 Aurora PostgreSQL DB 클러스터의 SSL/TLS 인증서를 생성합니다. SSL/TLS 인증서 확인을 활성화하는 경우에는 SSL/TLS 인증서에 스푸핑 공격으로부터 보호해주는 SSL/TLS 인증서를 위한 일반 이름(CN)으로 DB 클러스터 엔드포인트가 포함됩니다.

SSL/TLS를 통해 Aurora PostgreSQL DB 클러스터에 연결하기

1. 인증서를 다운로드합니다.

   인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하세요.

2. 다음과 같이 사용 중인 운영 체제로 인증서를 가져옵니다.
3. SSL/TLS를 통해 Aurora PostgreSQL DB 클러스터에 연결합니다.

   SSL/TLS를 사용하여 연결하면 클라이언트가 인증서 체인을 확인할지 여부를 선택할 수 있습니다. 연결 파라미터가 `sslmode=verify-ca` 또는 `sslmode=verify-full`을 지정하면, 클라이언트는 RDS CA 인증서를 트러스트 스토어에 있거나 연결 URL에서 참조되게 할 것을 요구합니다. 이 요구 사항은 데이터베이스 인증서에 서명하는 인증서 체인을 확인하는 것입니다.

   psql 또는 JDBC와 같은 클라이언트가 SSL/TLS를 지원하도록 구성되어 있는 경우 클라이언트는 먼저 SSL/TLS를 이용해 데이터베이스에 연결을 시도하도록 기본 설정되어 있습니다. SSL/TLS를 이용해 연결할 수 없는 경우 클라이언트는 SSL/TLS 없이 연결하는 방식으로 전환됩니다. 기본적으로 JDBC 및 libpq 기반 클라이언트의 `sslmode` 옵션은 `prefer`로 설정됩니다.

sslrootcert 파라미터를 사용하여 인증서를 참조합니다(예: sslrootcert=rds-ssl-ca-cert.pem).

다음은 psql을 사용해 Aurora PostgreSQL DB 클러스터에 연결하는 것의 예시입니다.

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

Aurora PostgreSQL DB 클러스터에 대한 SSL/TLS 연결 요구

rds.force_ssl 파라미터를 사용하여 Aurora PostgreSQL DB 클러스터에 대한 연결 시 SSL/TLS를 사용하도록 요구할 수 있습니다.

- SSL/TLS 연결을 요구하려면 rds.force_ssl 파라미터 값을 1(켜짐)로 설정합니다.
- SSL/TLS 연결 요구를 끄려면 rds.force_ssl 파라미터 값을 0(꺼짐)으로 설정합니다.

이 파라미터의 기본값은 Aurora PostgreSQL 버전에 따라 다릅니다.

- Aurora PostgreSQL 버전 17 이상: 기본값은 1(켜짐)입니다.
- Aurora PostgreSQL 버전 16 이상: 기본값은 0(꺼짐)입니다.

> **Note**
>
> Aurora PostgreSQL 버전 16 이하에서 버전 17 이상으로 메이저 버전 업그레이드를 수행하면 파라미터의 기본값이 0(꺼짐)에서 1(켜짐)로 변경됩니다. 이 변경으로 인해 SSL용으로 구성되지 않은 애플리케이션에 대한 연결이 실패할 수 있습니다. 이 파라미터를 0(꺼짐)으로 설정하여 이전 기본 동작으로 되돌릴 수 있습니다.

파라미터 처리에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

rds.force_ssl 파라미터를 업데이트해도 PostgreSQL ssl 파라미터가 1(설정)로 설정되고, DB 클러스터의 pg_hba.conf 파일이 새로운 SSL/TLS 구성을 지원하도록 수정됩니다.

DB 클러스터에 대해 rds.force_ssl 파라미터를 1로 설정하면 연결 시 다음과 같이 SSL/TLS가 요구된다는 출력이 표시됩니다.

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

SSL/TLS 연결 상태 확인

DB 클러스터에 연결할 때 로그온 배너에 연결의 암호화된 상태가 표시됩니다.

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

또한, `sslinfo` 확장을 로드한 다음 `ssl_is_used()` 함수를 호출하여 SSL/TLS가 사용 중인지 확인할 수 있습니다. 이 함수는 연결이 SSL/TLS를 사용할 경우 `t`를 반환하고, 그렇지 않으면 `f`를 반환합니다.

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

`select ssl_cipher()` 명령을 사용하여 SSL/TLS 암호를 확인할 수 있습니다.

```
postgres=> select ssl_cipher();
ssl_cipher
```

```
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

set rds.force_ssl을 활성화하고 DB 클러스터를 다시 시작하면 SSL이 아닌 연결은 다음 메시지와 함께 거부됩니다.

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

sslmode 옵션에 대한 자세한 내용은 PostgreSQL 설명서의 데이터베이스 연결 제어 기능을 참조하세요.

Aurora PostgreSQL DB 클러스터 연결을 위한 암호 그룹 구성

구성 가능한 암호 그룹을 사용하면 데이터베이스 연결의 보안을 더 잘 제어할 수 있습니다. 데이터베이스에 대한 클라이언트 SSL/TLS 연결을 보호하도록 허용할 암호 그룹 목록을 지정할 수 있습니다. 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. 이렇게 하면 안전하지 않거나 사용되지 않는 암호의 사용을 방지하는 데 도움이 됩니다.

구성 가능한 암호 그룹은 Aurora PostgreSQL 버전 11.8 이상에서 지원됩니다.

연결을 암호화하는 데 허용되는 암호 목록을 지정하려면 ssl_ciphers 클러스터 파라미터를 수정합니다. AWS Management Console, AWS CLI, 또는 RDS API를 사용하여 클러스터 파라미터 그룹의 쉼표로 구분된 암호 값 문자열에 ssl_ciphers 파라미터를 설정합니다. 클러스터 파라미터를 설정하려면 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

다음 표에는 유효한 Aurora PostgreSQL 엔진 버전에 지원되는 암호가 나와 있습니다.

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
9.6, 10.20 이하, 11.15 이하, 12.10 이하, 13.6 이하	DHE-RSA-AES128-SHA DHE-RSA-AES128-SHA256	예 아니요 아니요	아니요 예 예	아니요 아니요 아니요

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
	DHE-RSA-AES128-GCM-SHA256 DHE-RSA-AES256-SHA DHE-RSA-AES256-SHA256 DHE-RSA-AES256-GCM-SHA384 ECDHE-ECDSA-AES256-SHA ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA384 ECDHE-RSA-AES128-SHA ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES256-SHA ECDHE-RSA-AES256-GCM-SHA384	아니요 아니요 아니요 예 아니요 아니요 예 아니요 아니요 예 아니요	예 예 예 예 예 예 예 예 예 예 예	아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
10.21, 11.16, 12.11, 13.7, 14.3, 14.4	ECDHE-RSA-AES128-SHATLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH	예 아니요 예 아니요 예 아니요 예 아니요 아니요 예 아니요 예 아니요	예 예 예 예 예 예 예 예 예 예 예 예 예	아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
	_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256			

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
10.22, 11.17, 12.12, 13.8, 14.5 및 15.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	예 아니요 아니요 예 아니요 예 아니요 아니요 예 아니요 아니요 예 아니요 예 예 아니요	예 예 예 예 예 예 예 예 예 예 예 예 예 예 예 예	아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256			

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
11.20, 12.15, 13.11, 14.8, 15.3, 16.1 이상	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	예 아니요 아니요 예 아니요 예 아니요 아니요 예 아니요 아니요 예 아니요 예 예 아니요 아니요 아니요	예 예 예 예 예 예 예 예 예 예 예 예 예 예 예 예 아니요 아니요	아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 아니요 예 예

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256			

Aurora PostgreSQL 엔진 버전	지원되는 암호	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_AES_2 56_GCM_SHA384			

describe-engine-default-cluster-parameters CLI 명령을 사용하여 특정 파라미터 그룹 제품군에 대해 현재 지원되는 암호 그룹을 결정할 수도 있습니다. 다음 예에서는 Aurora PostgreSQL 11에 대한 `ssl_cipher` 클러스터 파라미터에 대해 허용되는 값을 가져오는 방법을 보여줍니다.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

`ssl_ciphers` 파라미터는 허용되는 모든 암호 그룹으로 기본 설정됩니다. 암호에 대한 자세한 내용은 PostgreSQL 문서의 가변 ssl_cipher를 참조하세요.

새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션 업데이트

2023년 1월 13일부터 Amazon RDS는 보안 소켓 계층(SSL) 또는 전송 계층 보안(TLS)을 사용해 Aurora DB 클러스터에 연결하기 위한 용도의 새 인증 기관(CA) 인증서를 게시하였습니다. 아래에서 새 인증서를 사용하기 위해 애플리케이션을 업데이트하는 방법에 관한 정보를 찾으실 수 있습니다.

이 주제는 클라이언트 애플리케이션에서 SSL/TLS를 사용해 DB 클러스터에 연결하는지 여부를 판단하는 데 도움이 됩니다. SSL/TLS를 사용해 연결한다면 이 애플리케이션에서 연결 시 인증서 확인이 필요한지 여부를 추가로 확인할 수 있습니다.

> **Note**
>
> 어떤 애플리케이션은 서버에서 인증서를 성공적으로 확인할 수 있는 경우에만 Aurora PostgreSQL DB 클러스터에 연결하도록 구성되어 있습니다.
> 이러한 애플리케이션의 경우 클라이언트 애플리케이션 트러스트 스토어를 업데이트하여 새 CA 인증서를 포함해야 합니다.

클라이언트 애플리케이션 트러스트 스토어에서 CA 인증서를 업데이트한 후에는 DB 클러스터에서 인증서를 교환할 수 있습니다. 이 절차를 프로덕션 환경에서 구현하기 전에 개발 또는 스테이징 환경에서 테스트해볼 것을 적극 권장합니다.

인증서 교환에 대한 자세한 내용은 SSL/TLS 인증서 교체 단원을 참조하십시오. 인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오. PostgreSQL DB 클러스터에서 SSL/TLS를 사용하는 방법에 관한 자세한 내용은 SSL/TLS를 이용한 Aurora PostgreSQL 데이터 보안 단원을 참조하십시오.

주제

애플리케이션에서 SSL을 사용해 Aurora PostgreSQL DB 클러스터에 연결하는지 여부 확인

`rds.force_ssl` 파라미터의 값에 대한 DB 클러스터 구성을 확인하십시오. 기본적으로 `rds.force_ssl` 파라미터는 `0`(해제)으로 설정됩니다. `rds.force_ssl` 파라미터가 1(켜짐)로 설정된 경우 클라이언트는 연결 시 SSL/TLS를 사용해야 합니다. 파라미터 그룹에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 단원을 참조하십시오.

`rds.force_ssl`이 1(켜짐)로 설정되지 않은 경우 `pg_stat_ssl` 보기를 쿼리하여 SSL을 사용해 연결하는지 확인하십시오. 예를 들어 다음 쿼리에서는 SSL 연결과 SSL을 사용하는 클라이언트에 관한 정보만 반환합니다.

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

SSL/TLS 연결을 사용하는 행만 연결에 관한 정보와 함께 표시됩니다. 다음은 출력 샘플입니다.

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

앞의 쿼리에서는 쿼리 시점의 현재 연결만 표시합니다. 결과가 표시되지 않는다 해도 SSL 연결을 사용하는 애플리케이션이 없는 것은 아닙니다. 다른 SSL 연결이 다른 시점에 설정될 수 있습니다.

클라이언트에서 연결을 위해 인증서 확인이 필요한지 여부 확인

psql 또는 JDBC와 같은 클라이언트가 SSL을 지원하도록 구성되어 있는 경우 클라이언트는 먼저 SSL을 이용해 데이터베이스에 연결을 시도하도록 기본 설정되어 있습니다. SSL을 이용해 연결할 수 없는 경우 클라이언트는 SSL 없이 연결하는 방식으로 전환됩니다. libpq 기반 클라이언트(예: psql)와 JDBC 모두에 사용되는 기본 `sslmode` 모드는 `prefer`로 설정됩니다. 서버의 인증서는 `sslrootcert`에서 `sslmode`가 `verify-ca` 또는 `verify-full`로 설정된 경우에만 확인됩니다. 인증서가 잘못된 경우 오류가 발생합니다.

`PGSSLROOTCERT`를 사용하여 `PGSSLMODE`가 `verify-ca` 또는 `verify-full`로 설정된 `PGSSLMODE` 환경 변수로 인증서를 확인하세요.

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

`sslrootcert` 인수를 사용하여 `sslmode`가 `verify-ca` 또는 `verify-full`로 설정된 연결 문자열 형식의 `sslmode`로 인증서를 확인하세요.

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

예를 들어 앞의 사례에서 잘못된 루트 인증서를 사용하는 경우 클라이언트에서 다음과 비슷한 오류가 발생합니다.

```
psql: SSL error: certificate verify failed
```

애플리케이션 트러스트 스토어 업데이트

PostgreSQL 애플리케이션에 대한 트러스트 스토어 업데이트에 대한 자세한 내용은 PostgreSQL 문서의 SSL을 이용한 TCP/IP 연결의 보안 단원을 참조하십시오.

> **Note**
>
> 트러스트 스토어를 업데이트할 때 새 인증서를 추가할 뿐 아니라 이전 인증서를 유지할 수도 있습니다.

JDBC를 위한 애플리케이션 트러스트 스토어 업데이트

SSL/TLS 연결을 위해 JDBC를 사용하는 애플리케이션에 대해 트러스트 스토어를 업데이트할 수 있습니다.

루트 인증서 다운로드에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오.

인증서를 가져오는 샘플 스크립트는 트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트 섹션을 참조하세요.

다양한 유형의 애플리케이션에 대해 SSL/TLS 연결 사용

아래에서는 다양한 유형의 애플리케이션에 대해 SSL/TLS 연결을 사용하는 방법에 대한 정보를 제공합니다.

- psql

  클라이언트는 명령줄에서 옵션을 연결 문자열 또는 환경 변수로 지정하여 호출합니다. SSL/TLS 연결의 경우 관련 옵션은 `sslmode`(환경 변수 `PGSSLMODE`), `sslrootcert`(환경 변수 `PGSSLROOTCERT`)입니다.

  옵션 전체 목록은 PostgreSQL 문서의 파라미터 키 단어 단원을 참조하십시오. 환경 변수 전체 목록은 PostgreSQL 문서의 환경 변수 단원을 참조하십시오.

- pgAdmin

  이 브라우저 기반 클라이언트는 PostgreSQL 데이터베이스 연결 시 사용할 수 있는 더 사용자 친화적인 인터페이스입니다.

  연결 구성에 대한 자세한 내용은 pgAdmin 설명서를 참조하십시오.

- JDBC

  JDBC를 통해 Java 애플리케이션의 데이터베이스 연결을 활성화할 수 있습니다.

  JDBC를 이용한 PostgreSQL 데이터베이스 연결에 대한 자세한 내용은 PostgreSQL 문서의 데이터베이스에 연결 단원을 참조하십시오. SSL/TLS을 이용한 PostgreSQL 문서의 클라이언트 구성 단원을 참조하십시오.

- Python

  PostgreSQL 데이터베이스에 연결하기 위해 많이 사용되는 인기 있는 Python 라이브러리는 `psycopg2`입니다.

  `psycopg2` 사용에 대한 자세한 내용은 psycopg2 설명서를 참조하십시오. PostgreSQL 데이터베이스에 연결하는 방법에 대한 짧은 자습서는 Psycopg2 자습서를 참조하십시오. psycopg2 모듈 콘텐츠에서 연결 명령이 수락하는 옵션에 대한 정보를 얻을 수 있습니다.

> **Important**
>
> 데이터베이스 연결에서 SSL/TLS를 사용함을 확인하고 애플리케이션 트러스트 스토어를 업데이트한 후에는 데이터베이스에서 rds-ca-rsa2048-g1 인증서를 사용하도록 업데이트할 수 있습니다. 지침은 DB 인스턴스를 수정하여 CA 인증서 업데이트의 3단계를 참조하십시오.

Aurora PostgreSQL과 함께 Kerberos 인증 사용

사용자가 PostgreSQL이 실행되는 DB 클러스터 에 연결할 때 Kerberos를 사용하여 사용자를 인증할 수 있습니다. 이를 위해서는 Kerberos 인증을 위한 AWS Directory Service for Microsoft Active Directory를 사용하도록 클러스터를 구성해야 합니다. AWS Directory Service for Microsoft Active Directory는 AWS Managed Microsoft AD라고도 불립니다. AWS Directory Service에서 사용할 수 있는 기능입니다. 자세한 내용은 AWS Directory Service 관리 가이드의 AWS Directory Service란 무엇입니까?를 참조하세요.

시작하려면 사용자 자격 증명을 저장할 AWS Managed Microsoft AD 디렉터리를 만듭니다. 그런 다음 PostgreSQL DB 클러스터에 Active Directory의 도메인 및 기타 정보를 제공합니다. PostgreSQL DB 클러스터에 대해 사용자가 인증될 때 AWS Managed Microsoft AD 디렉터리에 인증 요청이 전달됩니다.

모든 자격 증명을 동일한 디렉터리에 보관하면 시간과 노력을 절약할 수 있습니다. 여러 DB 클러스터에 대한 자격 증명을 보관하고 관리할 수 있는 중앙 집중식 공간이 있습니다. 디렉터리를 사용하면 전체 보안 프로필을 향상할 수도 있습니다.

자체 온프레미스 Microsoft Active Directory에서 자격 증명에 액세스할 수도 있습니다. 이렇게 하려면 AWS Managed Microsoft AD 디렉터리가 온프레미스 Microsoft Active Directory를 신뢰하도록 신뢰 도메인 관계를 만듭니다. 그러면 사용자가 온프레미스 네트워크에서 워크로드에 액세스할 때와 동일한 Windows SSO(Single Sign-On) 환경을 사용하여 PostgreSQL 클러스터에 액세스할 수 있습니다.

데이터베이스는 Kerberos 또는 AWS Identity and Access Management(IAM) 인증을 사용하거나, Kerberos 인증과 IAM 인증을 모두 사용할 수 있습니다. 하지만 Kerberos 및 IAM 인증은 서로 다른 인증 방법을 제공하므로, 특정 데이터베이스 사용자는 두 가지 인증 방법 중 하나만 사용하여 데이터베이스에 로그인해야 하며 둘 다 사용할 수는 없습니다. IAM 인증에 관한 자세한 내용은 IAM 데이터베이스 인증 단원을 참조하십시오.

주제

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전 리전에 따라 다릅니다. Kerberos 인증을 사용하는 Aurora PostgreSQL의 버전 및 리전 가용성에 대한 자세한 내용은 Aurora PostgreSQL을 사용하는 Kerberos 인증 섹션을 참조하세요.

PostgreSQL DB 클러스터에 대한 Kerberos 인증 개요

PostgreSQL DB 클러스터에 대해 Kerberos 인증을 설정하려면 다음 단계(나중에 자세히 설명함)를 수행하십시오.

1. AWS Managed Microsoft AD를 사용하여 AWS Managed Microsoft AD 디렉터리를 생성합니다. AWS Management Console, AWS CLI 또는 AWS Directory Service API를 사용하여 디렉터리를 생성할 수 있습니다. 디렉터리가 클러스터와 통신할 수 있도록 디렉터리 보안 그룹에서 관련 아웃바운드 포트를 열어야 합니다.
2. AWS Managed Microsoft AD 디렉터리에 호출할 수 있는 Amazon Aurora 액세스를 제공하는 역할을 생성합니다. 이를 위해 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 AWS Identity and Access Management(IAM) 역할을 생성합니다.

   IAM 역할이 액세스를 허용하게 하려면 올바른 AWS Security Token Service 리전에서 AWS STS 계정의 AWS(AWS) 엔드포인트를 활성화해야 합니다. AWS STS 엔드포인트는 기본적으로 모든 AWS 리전에서 활성화되어 있으므로 추가 작업 없이 사용할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS STS 리전에서 AWS 활성화 및 비활성화를 참조하세요.
3. Microsoft Active Directory 도구를 사용하여 AWS Managed Microsoft AD 디렉터리에서 사용자를 만들고 구성합니다. Active Directory에서 사용자를 생성하는 방법에 대한 자세한 내용은 AWS 관리 안내서의 AWS Directory Service 관리형 Microsoft AD에서 사용자 및 그룹 관리를 참조하세요.
4. 디렉터리와 DB 인스턴스를 다른 AWS 계정 또는 Virtual Private Cloud(VPC)에 배치하려면 VPC 피어링을 구성합니다. 자세한 내용은 Amazon VPC Peering Guide의 VPC 피어링이란?을 참조하십시오.
5. 콘솔, CLI 또는 RDS API에서 다음 메서드 중 하나를 사용하여 PostgreSQL DB 클러스터를 생성하거나 수정합니다.
   - Aurora PostgreSQL DB 클러스터 생성 및 연결
   - Amazon Aurora DB 클러스터 수정
   - DB 클러스터 스냅샷에서 복원
   - 지정된 시간으로 DB 클러스터 복원

디렉터리와 동일한 Amazon Virtual Private Cloud(VPC) 또는 다른 AWS 계정이나 VPC에 클러스터를 배치할 수 있습니다. PostgreSQL DB 클러스터를 생성하거나 수정할 때는 다음을 수행합니다.

- 디렉터리를 만들 때 생성된 도메인 식별자(d-* 식별자)를 제공합니다.
- 생성한 IAM 역할의 이름을 제공합니다.
- DB 인스턴스의 보안 그룹이 디렉터리의 보안 그룹에서 인바운드 트래픽을 수신할 수 있는지 확인합니다.

6. RDS 마스터 사용자 자격 증명을 사용하여 PostgreSQL DB 클러스터에 연결합니다. 외부에서 식별할 사용자를 PostgreSQL에서 생성합니다. 외부에서 식별되는 사용자는 Kerberos 인증을 사용하여 PostgreSQL DB 클러스터에 로그인할 수 있습니다.

PostgreSQL DB 클러스터에 대해 Kerberos 인증 설정

AWS Directory Service for Microsoft Active Directory(AWS Managed Microsoft AD)를 사용하여 PostgreSQL DB 클러스터에 대해 Kerberos 인증을 설정합니다. Kerberos 인증을 설정하려면 다음 단계를 수행하십시오.

주제

1단계: AWS Managed Microsoft AD를 사용하여 디렉터리 생성

AWS Directory Service는 AWS 클라우드에서 완전 관리형 Microsoft Active Directory를 생성합니다. AWS Managed Microsoft AD 디렉터리를 생성할 때 AWS Directory Service에서 두 개의 도메인 컨트롤러와 DNS 서버가 자동으로 생성됩니다. 디렉터리 서버는 VPC 내 다른 서브넷에서 생성됩니다. 이러한 중복으로 인해 장애가 발생해도 디렉터리에 액세스할 수 있습니다.

AWS Managed Microsoft AD 디렉터리를 생성할 때 AWS Directory Service에서 다음 작업을 자동으로 수행합니다.

- VPC 내에서 Active Directory를 설정합니다.
- 사용자 이름 `Admin`과 지정된 암호를 사용하여 디렉터리 관리자 계정을 생성합니다. 이 계정을 사용하여 디렉터리를 관리할 수 있습니다.

> **Important**
>
> 반드시 이 암호를 저장해야 합니다. AWS Directory Service에서는 이 암호를 저장하지 않으므로 암호를 검색하거나 다시 설정할 수 없습니다.

- 디렉터리 컨트롤러에 대한 보안 그룹을 만듭니다. 보안 그룹이 PostgreSQL DB 클러스터와의 통신을 허용해야 합니다.

AWS Directory Service for Microsoft Active Directory를 시작하면 AWS가 모든 디렉터리의 객체를 포함하는 OU(조직 단위)를 생성합니다. 디렉터리를 만들 때 입력한 NetBIOS 이름이 있는 이 OU는 도메인 루트에 있습니다. 도메인 루트는 AWS에서 소유하고 관리합니다.

`Admin` 디렉터리를 사용하여 생성한 AWS Managed Microsoft AD 계정은 OU의 가장 일반적인 관리 활동에 대한 권한이 있습니다.

- 사용자 생성, 업데이트 또는 삭제
- 도메인(예: 파일 또는 인쇄 서버)에 리소스를 추가한 다음 OU 내의 사용자에 해당 리소스에 대한 권한 할당
- 추가 OU 및 컨테이너 생성
- 권한 위임
- Active Directory 휴지통에서 삭제된 객체 복원
- Active Directory 웹 서비스에서 Active Directory 및 Windows PowerShell에 대한 DNS(Domain Name Service) 모듈 실행

또한 `Admin` 계정은 다음 도메인 차원 활동을 수행할 권한이 있습니다.

- DNS 구성 관리(레코드, 영역 및 전달자 추가, 제거 또는 업데이트)
- DNS 이벤트 로그 보기
- 보안 이벤트 로그 보기

AWS Managed Microsoft AD으로 디렉터리를 생성하려면

1. [AWS Directory Service 콘솔](https://console.aws.amazon.com/directoryservicev2/) 탐색 창에서 디렉터리를 선택한 후 디렉터리 설정을 선택합니다.
2. AWS Managed Microsoft AD를 선택합니다. AWS Managed Microsoft AD는 현재 Amazon Aurora에서 사용하도록 지원되는 유일한 옵션입니다.
3. 다음을 선택합니다.
4. 디렉터리 정보 입력 페이지에서 다음 정보를 제공합니다.

   Edition

   요구 사항에 맞는 에디션을 선택합니다.

   디렉터리 DNS 이름

   디렉터리를 위한 정규화된 이름(예: **`corp.example.com`**)입니다.

   디렉터리 NetBIOS 이름

   디렉터리의 선택적 짧은 이름(예: `CORP`)입니다.

   디렉터리 설명

   디렉터리에 대한 선택적 설명을 입력합니다.

   관리자 암호

   디렉터리 관리자의 암호입니다. 디렉터리 생성 프로세스에서는 사용자 이름 `Admin`와 이 암호를 사용하여 관리자 계정을 생성합니다.

   디렉터리 관리자 암호는 "admin"이라는 단어를 포함할 수 없습니다. 암호는 대소문자를 구분하며 길이가 8~64자 사이여야 합니다. 또한 다음 네 범주 중 세 개에 해당하는 문자를 1자 이상 포함해야 합니다.

   - 소문자(a–z)
   - 대문자(A–Z)
   - 숫자(0–9)
   - 영숫자 외의 특수 문자(~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

   [Confirm Password]

   관리자 암호를 다시 입력합니다.

> **Important**
>
> 반드시 이 암호를 저장해야 합니다. AWS Directory Service에서는 이 암호를 저장하지 않으며 암호를 검색하거나 재설정할 수 없습니다.

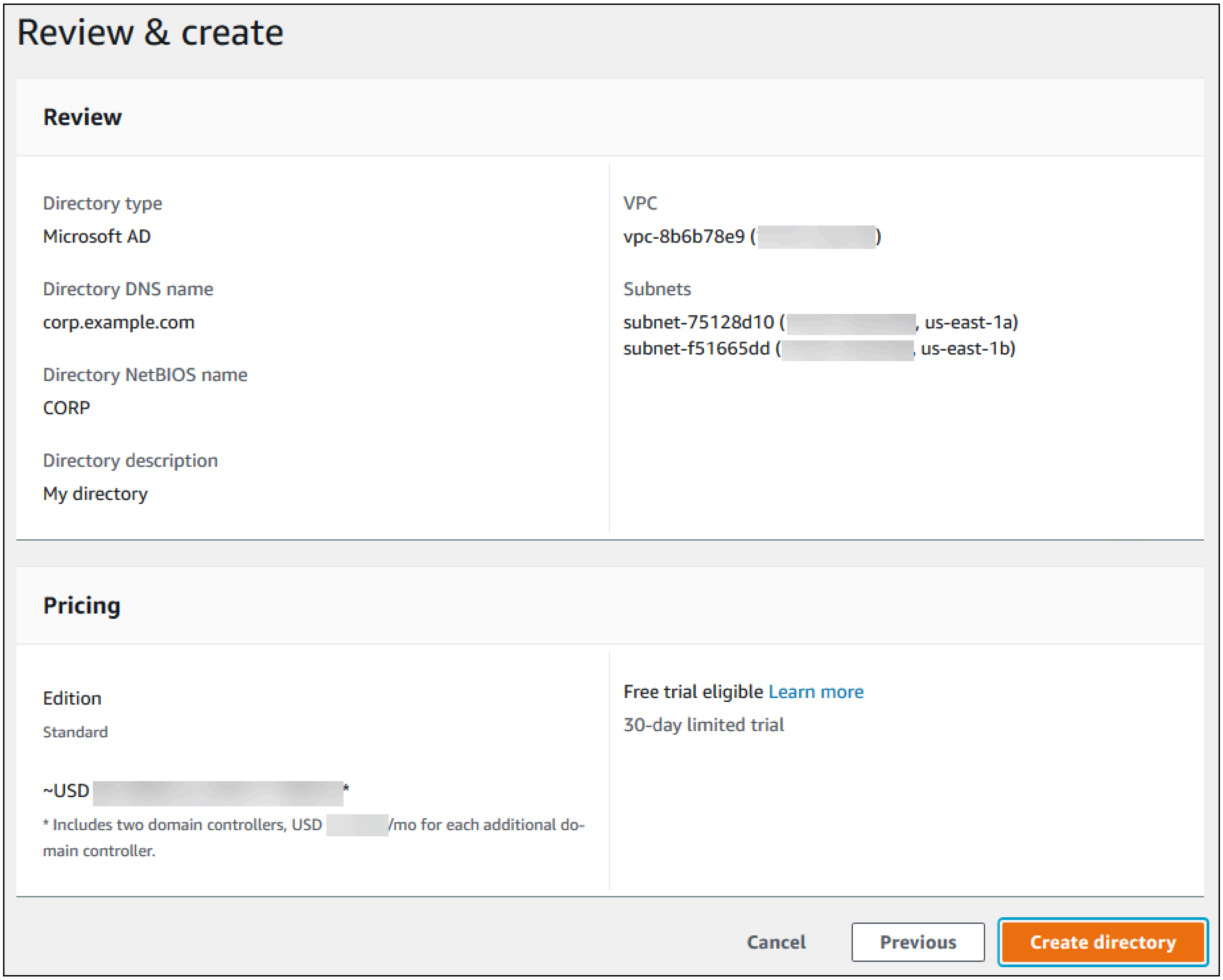
5. Next(다음)를 선택합니다.
6. Choose VPC and subnets(VPC 및 서브넷 선택) 페이지에 다음 정보를 입력합니다.

   VPC

   디렉터리에 대한 VPC를 선택합니다. PostgreSQL DB 클러스터는 이와 동일한 VPC 또는 다른 VPC에서 생성할 수 있습니다.

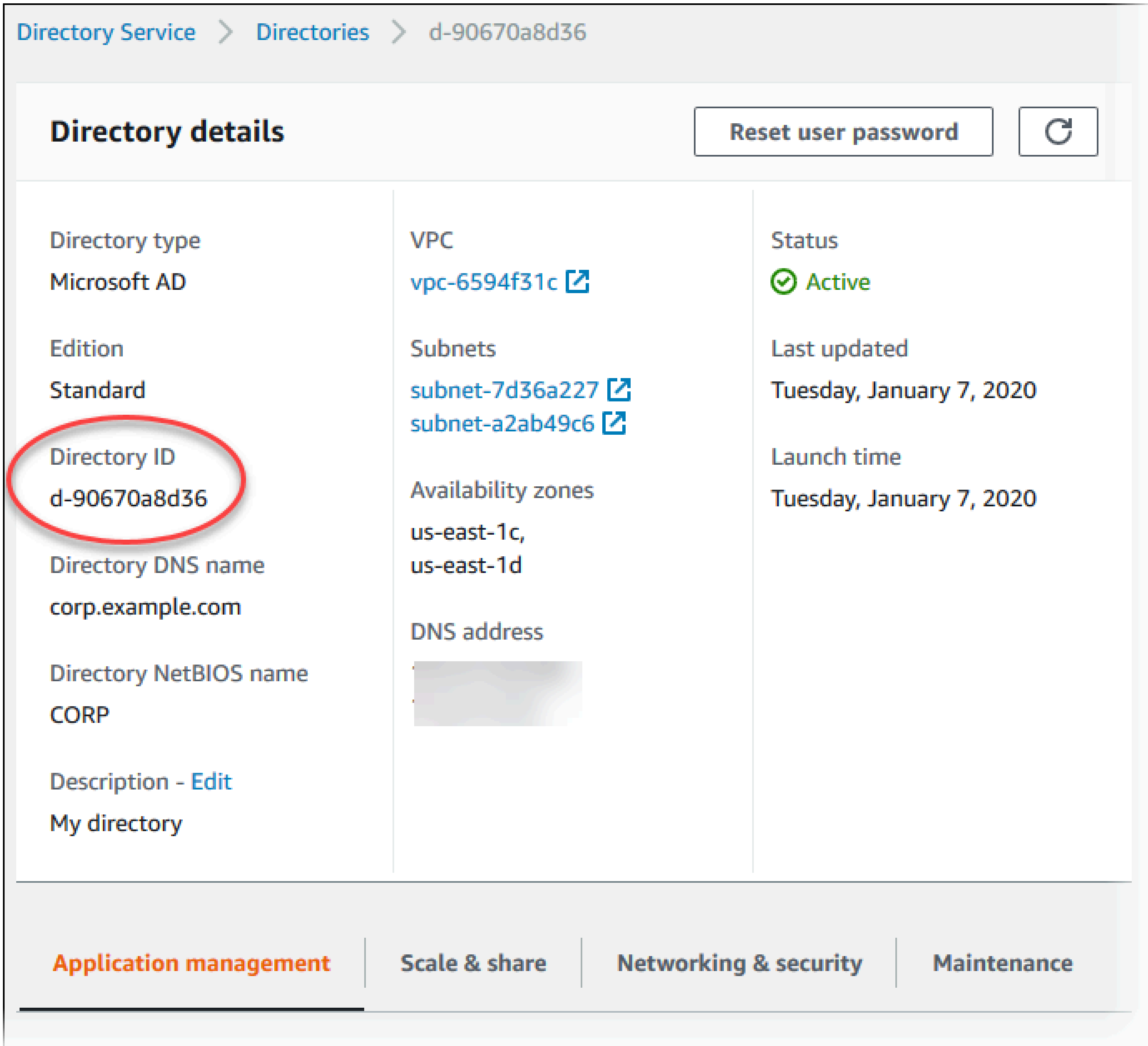
   서브넷

   디렉터리 서버에 대한 서브넷을 선택합니다. 두 서브넷이 서로 다른 가용 영역에 있어야 합니다.
7. Next(다음)를 선택합니다.
8. 디렉터리 정보를 검토합니다. 변경이 필요하면 이전을 선택하여 변경합니다. 정보가 올바르면 Create directory(디렉터리 생성)을 선택합니다.



디렉터리를 생성하는 데 몇 분 정도 걸립니다. 디렉터리가 성공적으로 생성되면 상태 값이 활성으로 변경됩니다.

디렉터리에 대한 정보를 보려면 디렉터리 목록에서 해당 디렉터리 ID를 선택합니다. 디렉터리 ID 값을 적어 두십시오. PostgreSQL DB 인스턴스를 생성하거나 수정할 때 이 값이 필요합니다.



2단계: (선택 사항) 온프레미스 Active Directory와 AWS Directory Service 간에 신뢰 관계 생성

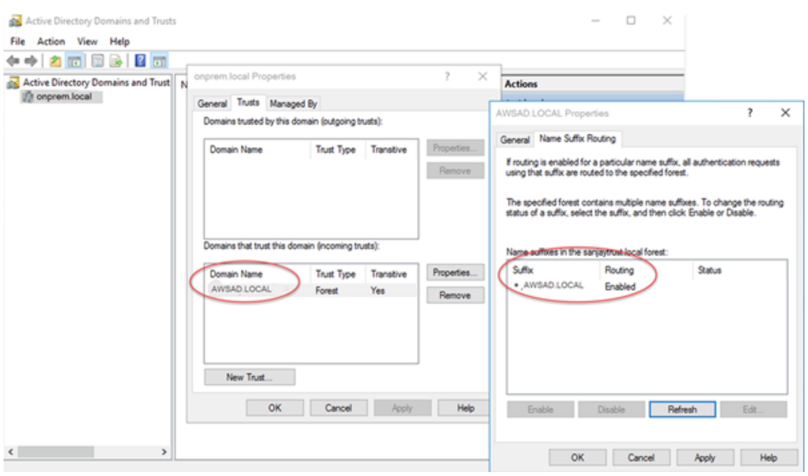
자체 온프레미스 Microsoft Active Directory를 사용하지 않으려는 경우 3단계: Amazon Aurora가 AWS Directory Service에 액세스할 수 있는 IAM 역할 생성로 건너뜁니다.

온프레미스 Active Directory를 사용하여 Kerberos 인증을 받으려면 온프레미스 Microsoft Active Directory와 AWS Managed Microsoft AD에서 만든 1단계: AWS Managed Microsoft AD를 사용하여 디렉터리 생성 디렉터리 간에 forest trust를 사용하여 신뢰 도메인 관계를 생성해야 합니다. 신뢰는 AWS Managed Microsoft AD 디렉터리가 온프레미스 Microsoft Active Directory를 신뢰하는 단방향일 수도

있고, 두 Active Directory가 서로를 신뢰하는 양방향일 수도 있습니다. AWS Directory Service를 사용하여 신뢰를 설정하는 방법에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 신뢰 관계를 생성해야 하는 경우를 참조하세요.

> **Note**
>
> 온프레미스 Microsoft Active Directory를 사용하는 경우:
>
> - Windows 클라이언트는 rds.amazonaws.com이 아닌 엔드포인트에 있는 AWS Directory Service의 도메인 이름을 사용하여 연결해야 합니다. 자세한 내용은 Kerberos 인증을 사용하여 PostgreSQL 연결 섹션을 참조하세요.
> - Windows 클라이언트가 Aurora 사용자 지정 엔드포인트를 사용하여 연결할 수 없습니다. 자세한 내용은 Amazon Aurora 엔드포인트 연결를 참조하세요.
> - 글로벌 데이터베이스의 경우:
>   - Windows 클라이언트가 글로벌 데이터베이스의 기본 AWS 리전에 있는 인스턴스 엔드포인트 또는 클러스터 엔드포인트만 사용하여 연결할 수 있습니다.
>   - Windows 클라이언트가 보조 AWS 리전의 클러스터 엔드포인트를 사용하여 연결할 수 없습니다.

온프레미스 Microsoft Active Directory 도메인 이름에 새로 만든 신뢰 관계에 해당하는 DNS 접미사 라우팅이 포함되어 있는지 확인합니다. 다음 스크린샷은 예를 보여줍니다.



3단계: Amazon Aurora가 AWS Directory Service에 액세스할 수 있는 IAM 역할 생성

AWS Directory Service를 호출하는 Amazon Aurora의 경우 AWS 계정에 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 IAM 역할이 필요합니다. 이 역할을 사용하여

Amazon Aurora에서 AWS Directory Service를 호출할 수 있습니다. (AWS Directory Service에 액세스하기 위한 이 IAM 역할은 IAM 데이터베이스 인증에 사용되는 IAM 역할과 다름)

AWS Management Console을 사용하여 DB 인스턴스를 생성할 때 콘솔 사용자에게 `iam:CreateRole` 권한이 있으면 콘솔에서 필요한 IAM 역할을 자동으로 생성합니다. 이 경우 역할 이름은 `rds-directoryservice-kerberos-access-role`입니다. 그렇지 않으면 IAM 역할을 수동으로 생성해야 합니다. 이 IAM 역할을 생성할 때 `Directory Service`를 선택하고 여기에 AWS 관리형 정책인 `AmazonRDSDirectoryServiceAccess`를 연결합니다.

서비스에 대한 IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.

> **Note**
>
> RDS for Microsoft SQL Server에 대한 Windows 인증에 사용되는 IAM 역할은 Amazon Aurora에 사용할 수 없습니다.

`AmazonRDSDirectoryServiceAccess` 관리형 정책인을 사용하는 대신 필요한 권한이 포함된 정책을 생성할 수 있습니다. 이 경우 IAM 역할에 다음과 같은 IAM 신뢰 정책이 있어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

또한 역할에는 다음과 같은 IAM 역할 정책도 있어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

옵트인 AWS 리전의 경우 IAM 역할 신뢰 정책에서 리전별 서비스 위탁자를 사용합니다. 이러한 리전의 서비스에 대한 신뢰 정책을 만들 때 서비스 위탁자에 리전 코드를 지정합니다.

다음 예는 리전별 서비스 위탁자가 포함된 신뢰 정책을 보여 줍니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
```

```
    ]
}
```

REGION-CODE를 사용자의 특정 리전 코드로 바꾸세요. 예를 들어 아시아 태평양(멜버른) 리전에는 다음 서비스 위탁자를 사용합니다.

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

4단계: 사용자 생성 및 구성

Active Directory Users and Computers 도구를 사용하여 사용자를 생성할 수 있습니다. 이 도구는 Active Directory 도메인 서비스 및 Active Directory Lightweight Directory Services 도구 중 하나입니다. 자세한 내용은 Microsoft 설명서의 Active Directory 도메인에 사용자 및 컴퓨터 추가를 참조하세요. 이 경우 사용자는 도메인에 속하며 디렉터리에서 ID가 유지되는 개인 또는 기타 엔터티(예: 사용자의 컴퓨터)입니다.

AWS Directory Service 디렉터리에서 사용자를 생성하려면 AWS Directory Service 디렉터리의 멤버인 Windows 기반 Amazon EC2 인스턴스에 연결되어 있어야 합니다. 이와 동시에 사용자를 생성할 권한이 있는 사용자로 로그인한 상태이어야 합니다. 자세한 내용은 AWS Directory Service관리 안내서의 사용자 생성을 참조하십시오.

5단계: 디렉터리와 DB 인스턴스 사이에 VPC 간 트래픽 활성화

디렉터리와 DB 클러스터를 동일한 VPC에 배치하려면 이 단계를 건너뛰고 6단계: PostgreSQL DB 클러스터 생성 또는 수정 단원으로 이동하십시오.

디렉터리와 DB 인스턴스를 서로 다른 VPC에 배치하려면 VPC 피어링 또는 AWS Transit Gateway를 사용하여 VPC 간 트래픽을 구성하세요.

다음 절차는 VPC 피어링을 사용하여 VPC 간 트래픽을 활성화합니다. Amazon Virtual Private Cloud 피어링 안내서의 VPC 피어링이란? 지침을 따르십시오.

VPC 피어링을 사용하여 VPC 간 트래픽을 활성화하려면

1. 네트워크 트래픽이 양방향으로 흐를 수 있도록 적절한 VPC 라우팅 규칙을 설정합니다.

2. DB 인스턴스의 보안 그룹이 디렉터리의 보안 그룹에서 인바운드 트래픽을 수신할 수 있는지 확인합니다.
3. 트래픽을 차단하는 네트워크 ACL(액세스 제어 목록) 규칙이 없어야 합니다.

다른 AWS 계정이 디렉터리를 소유하는 경우 디렉터리를 공유해야 합니다.

AWS 계정 간에 디렉터리를 공유하려면

1. AWS 관리 안내서의 자습서: 원활한 EC2 도메인 조인을 위해 AWS Managed Microsoft AD 디렉터리 공유에 있는 지침에 따라 DB 인스턴스가 생성될 AWS Directory Service 계정과 디렉터리를 공유하는 작업을 시작합니다.
2. DB 인스턴스용 계정을 사용하여 AWS Directory Service 콘솔에 로그인하고 계속하기 전에 도메인이 `SHARED` 상태가 되었는지 확인합니다.
3. DB 인스턴스용 계정을 사용하여 AWS Directory Service 콘솔에 로그인하는 동안 디렉터리 ID 값을 기록해 둡니다. 이 디렉터리 ID를 사용하여 DB 인스턴스를 도메인에 조인합니다.

6단계: PostgreSQL DB 클러스터 생성 또는 수정

디렉터리에서 사용할 PostgreSQL DB 클러스터를 생성하거나 수정합니다. 콘솔, CLI 또는 RDS API를 사용하여 DB 클러스터를 디렉터리에 연결할 수 있습니다. 이 작업을 다음 중 한 가지 방법으로 수행할 수 있습니다.

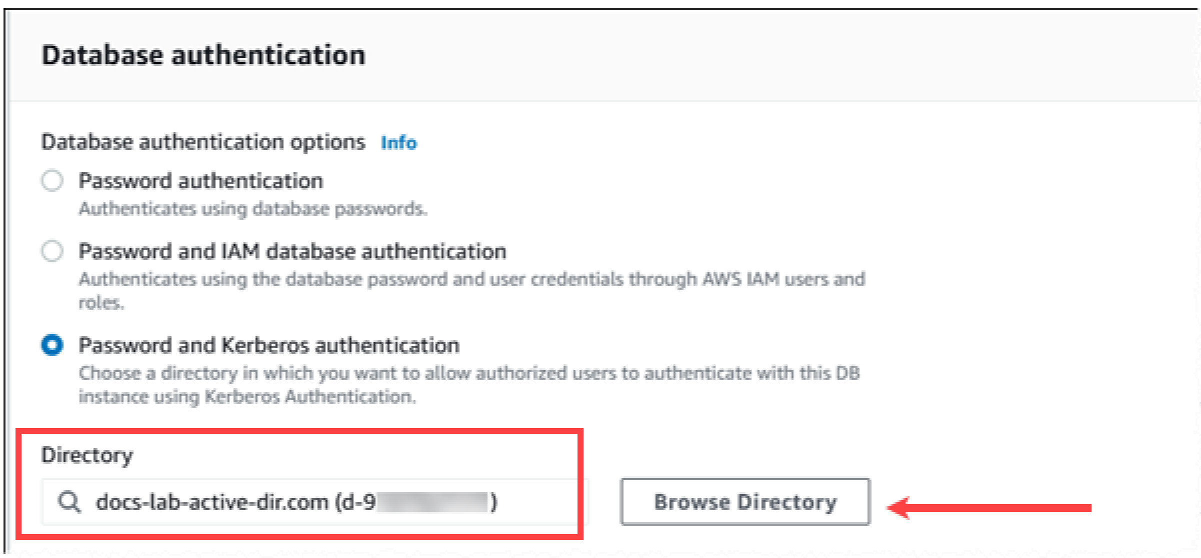
- 콘솔, create-db-cluster CLI 명령 또는 CreateDBCluster RDS API 작업을 사용하여 새 PostgreSQL DB 클러스터를 생성합니다. 지침은 Aurora PostgreSQL DB 클러스터 생성 및 연결 섹션을 참조하세요.
- 콘솔, modify-db-cluster CLI 명령 또는 ModifyDBCluster RDS API 작업을 사용하여 기존 PostgreSQL DB 클러스터를 수정합니다. 지침은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.
- 콘솔, restore-db-cluster-from-db-snapshot CLI 명령 또는 RestoreDBClusterFromDBSnapshot RDS API 작업을 사용하여 DB 스냅샷에서 PostgreSQL DB 클러스터를 복원합니다. 지침은 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.
- 콘솔, restore-db-instance-to-point-in-time CLI 명령 또는 RestoreDBClusterToPointInTime RDS API 작업을 사용하여 PostgreSQL DB 클러스터를 특정 시점으로 복원합니다. 지침은 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요.

Kerberos 인증은 VPC의 PostgreSQL DB 클러스터에 대해서만 지원됩니다. DB 클러스터는 디렉터리와 동일한 VPC 또는 다른 VPC에 있을 수 있습니다. DB 클러스터가 디렉터리와 통신할 수 있도록 DB 클러스터는 디렉터리의 VPC 내 수신 및 송신을 허용하는 보안 그룹을 사용해야 합니다.

> **Note**
>
> RDS for PostgreSQL에서 마이그레이션하는 동안에는 Aurora PostgreSQL DB 클러스터에서 Kerberos 인증을 활성화하는 기능이 현재 지원되지 않습니다. Kerberos 인증은 독립 실행형 Aurora PostgreSQL DB 클러스터에서만 활성화할 수 있습니다.

콘솔

콘솔을 사용하여 DB 클러스터를 생성, 수정 또는 복원하는 경우 [데이터베이스 인증(Database authentication)] 섹션에서 [Kerberos 인증(Kerberos authentication)]을 선택합니다. 그런 다음 [디렉터리 찾아보기(Browse Directory)]를 선택합니다. 디렉터리를 선택하거나 [새 디렉토리 생성(Create a new directory)]을 클릭하여 Directory Service를 사용합니다.



AWS CLI

AWS CLI를 사용하는 경우 생성한 디렉터리를 DB 클러스터에서 사용하려면 다음과 같은 파라미터가 필요합니다.

- `--domain` 파라미터의 경우 디렉터리를 만들 때 생성된 도메인 식별자("d-*" 식별자)를 사용하십시오.
- `--domain-iam-role-name` 파라미터의 경우 귀하가 생성한, 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 역할을 사용하십시오.

예를 들어, 다음 CLI 명령은 디렉터리를 사용하도록 DB 클러스터를 수정합니다.

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

Important

DB 클러스터 를 수정하여 Kerberos 인증을 사용 설정하는 경우 변경 후 DB 클러스터 를 재부팅합니다.

7단계: Kerberos 보안 주체를 위한 PostgreSQL 사용자 생성

이때 Aurora PostgreSQL DB 클러스터 가 AWS Managed Microsoft AD 도메인에 조인됩니다. 4단계: 사용자 생성 및 구성 에서 디렉터리에서 생성한 사용자는 PostgreSQL 데이터베이스 사용자로 설정하고 데이터베이스에 로그인할 수 있는 권한을 부여해야 합니다. `rds_superuser` 권한이 있는 데이터베이스 사용자로 로그인하면 됩니다. 예를 들어 Aurora PostgreSQL DB 클러스터, 를 생성할 때 기본값을 수락한 경우 다음 단계에 표시된 대로 `postgres`를 사용합니다.

Kerberos 보안 주체를 위한 PostgreSQL 데이터베이스 사용자를 생성하는 방법

1. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 DB 인스턴스 엔드포인트에 연결합니다. 다음 예에서는 `rds_superuser` 역할에 기본 `postgres` 계정을 사용합니다.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 데이터베이스에 액세스하도록 할 각 Kerberos 보안 주체(Active Directory 사용자 이름)에 대한 데이터베이스 사용자 이름을 생성합니다. Active Directory 인스턴스에 정의된 표준 사용자 이름(ID), 즉 해당 사용자 이름에 대한 Active Directory 도메인의 소문자 `alias`(Active Directory에서는 사용자 이름)와 대문자 이름을 사용합니다. Active Directory 사용자 이름은 외부에서 인증된 사용자이므로 다음과 같이 이름을 따옴표로 묶습니다.

```
postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
CREATE ROLE
```

3. 데이터베이스 사용자에게 `rds_ad` 역할을 부여합니다.

```
postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
```

```
GRANT ROLE
```

Active Directory 사용자 ID에 대한 모든 PostgreSQL 사용자 생성을 완료하면 사용자는 Kerberos 보안 인증 정보를 사용하여 Aurora PostgreSQL DB 클러스터에 액세스할 수 있습니다.

Kerberos를 사용하여 인증하는 데이터베이스 사용자는 Active Directory 도메인의 멤버인 클라이언트 컴퓨터에서 인증을 수행하도록 요구됩니다.

`rds_ad` 역할이 부여된 데이터베이스 사용자는 `rds_iam` 역할까지 가질 수는 없습니다. 이는 중첩된 멤버십에도 적용됩니다. 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요.

대소문자를 구분하지 않는 사용자 이름을 위해 Aurora PostgreSQL DB 클러스터 구성

Aurora PostgreSQL 버전 14.5, 13.8, 12.12, 11.17은 `krb_caseins_users` PostgreSQL 파라미터를 지원합니다. 이 파라미터는 대소문자를 구분하지 않는 Active Directory 사용자 이름을 지원합니다. 기본적으로 이 파라미터는 false로 설정되므로 Aurora PostgreSQL은 사용자 이름을 대소문자를 구분하여 해석합니다. 이는 모든 이전 Aurora PostgreSQL 버전의 기본 동작입니다. 하지만 사용자 지정 DB 클러스터 파라미터 그룹에서 이 파라미터를 `true`로 설정하고 Aurora PostgreSQL DB 클러스터가 사용자 이름을 대소문자를 구분하지 않고 해석하도록 할 수 있습니다. 데이터베이스 사용자가 Active Directory를 사용하여 인증할 때 사용자 이름의 대소문자를 잘못 입력할 수 있으므로 사용자의 편의를 위해 이 방법을 고려해 보세요.

`krb_caseins_users` 파라미터를 변경하려면 Aurora PostgreSQL DB 클러스터가 사용자 지정 DB 클러스터 파라미터 그룹을 사용 중이어야 합니다. 사용자 지정 DB 클러스터 파라미터 그룹 작업에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

AWS CLI 또는 AWS Management Console를 사용하여 설정을 변경할 수 있습니다. 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

8단계: PostgreSQL 클라이언트 구성

PostgreSQL 클라이언트를 구성하려면 다음 단계를 수행하십시오.

- 도메인을 가리키도록 krb5.conf 파일(또는 동등한 파일)을 생성합니다.
- 클라이언트 호스트와 AWS Directory Service 간에 트래픽이 흐를 수 있는지 확인합니다. Netcat과 같은 네트워크 유틸리티를 사용하여 다음을 수행하십시오.
  - 포트 53의 DNS를 통한 트래픽을 확인합니다.
  - 포트 53 및 AWS Directory Service용 포트 88 및 464를 포함하는 Kerberos의 TCP/UDP를 통한 트래픽을 확인합니다.

- 데이터베이스 포트를 통해 클라이언트 호스트와 DB 인스턴스 간에 트래픽이 흐를 수 있는지 확인합니다. 예를 들어, psql을 사용하여 데이터베이스에 연결하고 액세스합니다.

다음은 AWS Managed Microsoft AD의 샘플 krb5.conf 콘텐츠입니다.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

다음은 온프레미스 Microsoft Active Directory의 샘플 krb5.conf 콘텐츠입니다.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Active Directory 도메인에서 Aurora PostgreSQL DB 클러스터 관리

콘솔, CLI 또는 RDS API를 사용하여 DB 클러스터 및 Microsoft Active Directory와의 관계를 관리할 수 있습니다. 예를 들어, Active Directory를 연결하여 Kerberos 인증을 활성화할 수 있습니다. 또한 Active Directory 연결을 제거하여 Kerberos 인증을 비활성화할 수 있습니다. 또한 DB 클러스터를 이동하여 한 Microsoft Active Directory에서 다른 Microsoft Active Directory로 외부적으로 인증해 줄 수 있습니다.

예를 들어 CLI를 사용하여 다음 작업을 수행할 수 있습니다.

- 실패한 멤버십에 대한 Kerberos 인증 활성화를 다시 시도하려면 modify-db-cluster CLI 명령을 사용합니다. `--domain` 옵션에 대해 현재 멤버십의 디렉터리 ID를 지정합니다.
- DB 인스턴스에서 Kerberos 인증을 비활성화하려면 modify-db-cluster CLI 명령을 사용합니다. none 옵션의 경우 `--domain`을 지정합니다.
- 한 도메인에서 다른 도메인으로 DB 인스턴스를 이동하려면 modify-db-cluster CLI 명령을 사용합니다. `--domain` 옵션에 대해 새 도메인의 도메인 식별자를 지정합니다.

도메인 멤버십 이해

DB 클러스터를 생성하거나 수정하면 DB 인스턴스가 도메인의 멤버가 됩니다. 콘솔에서 또는 describe-db-instances CLI 명령을 실행하여 도메인 멤버십의 상태를 확인할 수 있습니다. DB 인스턴스의 상태는 다음 중 한 가지가 될 수 있습니다.

- `kerberos-enabled` - DB 인스턴스에 Kerberos 인증이 활성화되어 있습니다.
- `enabling-kerberos` - AWS에서 이 DB 인스턴스에 대한 Kerberos 인증 활성화를 진행 중입니다.
- `pending-enable-kerberos` - 이 DB 인스턴스에 대한 Kerberos 인증 활성화가 보류 중입니다.
- `pending-maintenance-enable-kerberos` - AWS에서 예약된 다음 유지 관리 기간에 DB 인스턴스에 대한 Kerberos 인증을 활성화하려 합니다.
- `pending-disable-kerberos` - 이 DB 인스턴스에 대한 Kerberos 인증 비활성화가 보류 중입니다.
- `pending-maintenance-disable-kerberos` - AWS에서 예약된 다음 유지 관리 기간에 DB 인스턴스에 대한 Kerberos 인증을 비활성화하려 합니다.
- `enable-kerberos-failed` - 구성 문제로 인해 AWS가 DB 인스턴스에 대해 Kerberos 인증을 활성화하지 못했습니다. DB 인스턴스 수정 명령을 다시 실행하기 전에 구성 문제를 해결하십시오.
- `disabling-kerberos` - AWS에서 이 DB 인스턴스에 대한 Kerberos 인증 비활성화를 진행 중입니다.

네트워크 연결 문제 또는 잘못된 IAM 역할로 인해 Kerberos 인증 활성화 요청이 실패할 수 있습니다. 경우에 따라 DB 클러스터를 생성하거나 수정할 때 Kerberos 인증을 사용하려고 하면 실패할 수 있습니다. 이런 경우 올바른 IAM 역할을 사용하고 있는지 확인한 다음 도메인에 조인하도록 DB 클러스터를 수정합니다.

Kerberos 인증을 사용하여 PostgreSQL 연결

pgAdmin 인터페이스 또는 psql과 같은 명령줄 인터페이스를 사용하여 Kerberos 인증으로 PostgreSQL에 연결할 수 있습니다. 연결에 대한 자세한 내용은 Amazon Aurora PostgreSQL DB 클러스터에 연결 섹션을 참조하세요. 연결에 필요한 엔드포인트, 포트 번호 및 기타 세부 정보를 얻는 방법에 대한 자세한 내용은 Aurora 클러스터의 엔드포인트 보기 단원을 참조하십시오.

> **Note**
>
> PostgreSQL의 GSSAPI 인증 및 암호화는 Kerberos 라이브러리 `libkrb5.so`에 의해 구현됩니다. `postgres_fdw` 및 `dblink`와 같은 기능은 Kerberos 인증 또는 암호화를 사용한 아웃바운드 연결에도 동일한 라이브러리를 사용합니다.

pgAdmin

pgAdmin을 사용하여 Kerberos 인증으로 PostgreSQL에 연결하려면 다음 단계를 수행하십시오.

1. 클라이언트 컴퓨터에서 pgAdmin 애플리케이션을 실행합니다.
2. [Dashboard] 탭에서 [Add New Server]를 선택합니다.
3. 생성 - 서버 대화 상자에서 pgAdmin의 서버를 식별하기 위해 일반 탭에 이름을 입력합니다.
4. 연결 탭에서 Aurora PostgreSQL 데이터베이스에 있는 다음 정보를 입력합니다.
   - 호스트의 경우 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 엔드포인트를 입력합니다. 엔드포인트는 다음과 유사하게 표시됩니다.

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     Windows 클라이언트에서 온-프레미스 Microsoft Active Directory에 연결하려면 호스트 엔드포인트의 `rds.amazonaws.com` 대신 AWS Managed Active Directory의 도메인 이름을 사용합니다. 예를 들어 AWS Managed Active Directory의 도메인 이름이 `corp.example.com`일 경우 그런 다음호스트에서는 엔드포인트가 다음과 같이 지정됩니다.

```
AUR-cluster-instance.111122223333.aws-region.corp.example.com
```

- 포트에 할당된 포트를 입력합니다.
- Maintenance database(유지 관리 데이터베이스)에 클라이언트가 연결될 초기 데이터베이스의 이름을 입력합니다.
- Username(사용자 이름)에 [7단계: Kerberos 보안 주체를 위한 PostgreSQL 사용자 생성](#) 의 Kerberos 인증을 위해 입력했던 사용자 이름을 입력합니다.

5. 저장을 선택합니다.

psql

psql을 사용하여 Kerberos 인증으로 PostgreSQL에 연결하려면 다음 단계를 수행하십시오.

1. 명령 프롬프트에서 다음 명령을 실행합니다.

   ```
   kinit username
   ```

   *username*을 사용자 이름으로 대체합니다. 프롬프트에서 Microsoft Active Directory에 저장된 사용자 암호를 입력합니다.

2. PostgreSQL DB 클러스터 가 공개적으로 액세스 가능한 VPC를 사용하는 경우 DB 클러스터 엔드포인트의 IP 주소를 EC2 클라이언트의 `/etc/hosts` 파일에 넣습니다. 예를 들어 다음 명령은 IP 주소를 얻은 다음 `/etc/hosts` 파일에 넣습니다.

   ```
   % dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
   ;; Truncated, retrying in TCP mode.
   ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
   34.210.197.118

   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
   hosts
   ```

   Windows 클라이언트에서 온프레미스 Microsoft Active Directory를 사용하는 경우 특수 엔드포인트를 사용하여 연결해야 합니다. 호스트 엔드포인트에서 Amazon 도메인 `rds.amazonaws.com`을 사용하는 대신 AWS Managed Active Directory의 도메인 이름을 사용합니다.

예를 들어 AWS Managed Active Directory의 도메인 이름이 `corp.example.com`일 경우 엔드포인트에 *`PostgreSQL-endpoint`*`.`*`AWS-Region`*`.corp.example.com` 형식을 사용하고 `/etc/hosts` 파일에 넣습니다.

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. 다음 psql 명령을 사용하여 Active Directory와 통합된 PostgreSQL DB 클러스터에 로그인합니다. 클러스터 또는 인스턴스 엔드포인트를 사용합니다.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

온프레미스 Active Directory를 사용하여 Windows 클러스터에서 PostgreSQL DB 클러스터에 로그인하려면 이전 단계의 도메인 이름(`corp.example.com`)과 함께 다음 psql 명령을 사용합니다.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

Aurora PostgreSQL 액세스 제어를 위한 AD 보안 그룹 사용

Aurora PostgreSQL 14.10 및 15.5 버전부터 Microsoft Active Directory(AD) 보안 그룹용 AWS Directory Service를 사용하여 Aurora PostgreSQL 액세스 제어를 관리할 수 있습니다. 이전 버전의 Aurora PostgreSQL은 개별 사용자에 대해서만 AD를 통한 Kerberos 기반 인증을 지원합니다. 액세스 권한을 얻으려면 각 AD 사용자를 DB 클러스터에 명시적으로 프로비저닝해야 했습니다.

비즈니스 요구 사항에 따라 각 AD 사용자를 DB 클러스터에 명시적으로 프로비저닝하는 대신, 아래 설명과 같이 AD 보안 그룹을 활용할 수 있습니다.

- AD 사용자는 Active Directory에 있는 다양한 AD 보안 그룹의 구성원입니다. 이는 DB 클러스터 관리자가 지시하는 것이 아니라 비즈니스 요구 사항을 기반으로 하며, AD 관리자가 처리합니다.
- DB 클러스터 관리자는 비즈니스 요구 사항에 따라 DB 인스턴스에 DB 역할을 생성합니다. 이러한 DB 역할은 서로 권한이 다를 수 있습니다.
- DB 클러스터 관리자는 DB 클러스터별로 AD 보안 그룹에서 DB 역할로의 매핑을 구성합니다.
- DB 사용자는 AD 보안 인증 정보를 사용하여 DB 클러스터에 액세스할 수 있습니다. 액세스는 AD 보안 그룹 구성원 자격을 기반으로 합니다. AD 사용자는 AD 그룹 구성원 자격에 따라 자동으로 액세스 권한을 얻거나 잃습니다.

사전 조건

AD 보안 그룹의 확장을 설정하기 전에 다음 사항이 준비되었는지 확인하세요.

- PostgreSQL DB 클러스터에 대한 Kerberos 인증 설정. 자세한 내용은 [PostgreSQL DB 클러스터에 대한 Kerberos 인증 설정]을 참조하세요.

> **Note**
>
> AD 보안 그룹의 경우 이 설정 절차에서 '7단계: Kerberos 보안 주체를 위한 PostgreSQL 사용자 생성' 단계를 건너뛰세요.

- 도메인에서 DB 클러스터 관리. 자세한 내용은 [도메인에서 DB 클러스터 관리]를 참조하세요.

pg_ad_mapping 확장 설정

Aurora PostgreSQL은 이제 Aurora PostgreSQL 클러스터에서 AD 보안 그룹과 DB 역할 간의 매핑을 관리할 수 있는 `pg_ad_mapping` 확장을 제공합니다. `pg_ad_mapping`에서 제공하는 함수에 대한 자세한 내용은 [`pg_ad_mapping` 확장의 함수 사용] 섹션을 참조하세요.

Aurora PostgreSQL DB 클러스터에 pg_ad_mapping 확장을 설정하려면 먼저 Aurora PostgreSQL DB 클러스터용 사용자 지정 DB 클러스터 파라미터 그룹의 공유 라이브러리에 pg_ad_mapping을 추가해야 합니다. 사용자 지정 DB 클러스터 파라미터 그룹 생성에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요. 다음으로 pg_ad_mapping 확장을 설치합니다. 이 섹션에 절차가 설명되어 있습니다. AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

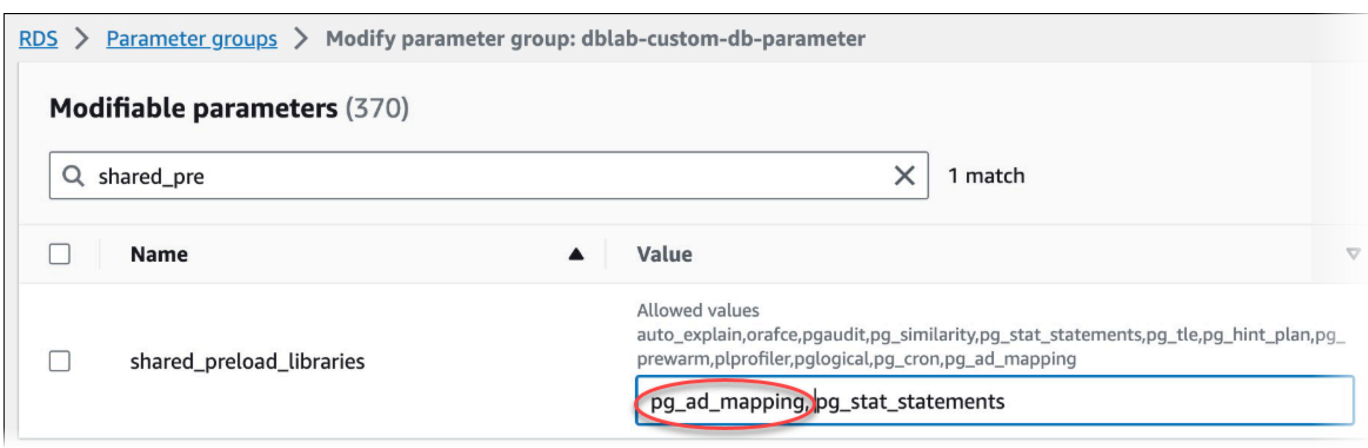
이 모든 작업을 수행하려면 rds_superuser 역할의 권한이 있어야 합니다.

다음 단계에서는 사용자의 Aurora PostgreSQL DB 클러스터가 사용자 지정 DB 클러스터 파라미터 그룹에 연결되어 있다고 가정합니다.

콘솔

pg_ad_mapping 확장 설정 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택합니다.
3. Aurora PostgreSQL DB 클러스터 라이터 인스턴스의 구성 탭을 엽니다. 인스턴스 세부 정보 중에서 파라미터 그룹 링크를 찾습니다.
4. 링크를 선택하여 Aurora PostgreSQL DB 클러스터와 연결된 사용자 지정 파라미터를 엽니다.
5. 파라미터 검색 필드에 shared_pre를 입력하여 shared_preload_libraries 파라미터를 찾습니다.
6. 파라미터 편집을 선택하여 속성 값에 액세스합니다.
7. 값 필드의 목록에 pg_ad_mapping를 추가합니다. 쉼표를 사용하여 값 목록에서 항목을 구분합니다.



8. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 `shared_preload_libraries` 파라미터 변경 사항이 적용되도록 합니다.
9. 인스턴스를 사용할 수 있게 되면 pg_ad_mapping가 초기화되었는지 확인합니다. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. pg_ad_mapping이 초기화되었으므로 이제 확장을 생성할 수 있습니다. 라이브러리를 초기화한 후에 확장을 생성해야 이 확장에서 제공되는 함수를 사용할 수 있습니다.

```
CREATE EXTENSION pg_ad_mapping;
```

11. `psql` 세션을 닫습니다.

```
labdb=> \q
```

AWS CLI

pg_ad_mapping 설정 방법

AWS CLI를 사용하여 pg_ad_mapping을 설정하려면 다음 절차와 같이 [modify-db-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-parameter-group.html) 작업을 호출하여 사용자 지정 파라미터 그룹에 이 파라미터를 추가합니다.

1. 다음 AWS CLI 명령을 사용하여 `shared_preload_libraries` 파라미터에 `pg_ad_mapping`를 추가합니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters 
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-reboot" \
   --region aws-region
```

2. 다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 pg_ad_mapping이 초기화되도록 합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 인스턴스를 사용할 수 있게 되면 pg_ad_mapping가 초기화되었는지 확인할 수 있습니다. psql을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

pg_ad_mapping이 초기화되었으므로 이제 확장을 생성할 수 있습니다.

```
CREATE EXTENSION pg_ad_mapping;
```

4. AWS CLI를 사용할 수 있도록 psql 세션을 닫습니다.

```
labdb=> \q
```

PowerShell에서 Active Directory 그룹 SID 검색

보안 식별자(SID)는 보안 주체 또는 보안 그룹을 고유하게 식별하는 데 사용됩니다. Active Directory에서 보안 그룹이나 계정을 만들 때마다 SID가 할당됩니다. Active Directory에서 AD 보안 그룹 SID를 가져오려면 해당 Active Directory 도메인과 연결된 Windows 클라이언트 컴퓨터에서 Get-ADGroup cmdlet을 사용할 수 있습니다. Identity 파라미터는 해당 SID를 가져올 Active Directory 그룹 이름을 지정합니다.

다음 예제에서는 AD 그룹 *adgroup1*의 SID를 반환합니다.

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

DB 역할을 AD 보안 그룹에 매핑

데이터베이스의 AD 보안 그룹을 PostgreSQL DB 역할로 명시적으로 프로비저닝해야 합니다. 하나 이상의 프로비저닝된 AD 보안 그룹에 속하는 AD 사용자는 데이터베이스에 액세스할 수 있습니다. AD 그룹 보안 기반 DB 역할에 `rds_ad role` 역할을 부여해서는 안 됩니다. 보안 그룹에 대한 Kerberos 인증은 도메인 이름 접미사(예: *user1@example.com*)를 사용하여 트리거됩니다. 이 DB 역할은 데이터베이스에 액세스하기 위해 암호 또는 IAM 인증을 사용할 수 없습니다.

> **Note**
>
> 데이터베이스에 해당 DB 역할과 `rds_ad` 역할이 부여된 AD 사용자는 AD 보안 그룹의 일원으로 로그인할 수 없습니다. 이들은 개별 사용자로서 DB 역할을 통해 액세스 권한을 얻게 됩니다.

예를 들어, accounts-group은 Aurora PostgreSQL에서 이 보안 그룹을 accounts-role로 프로비저닝하려는 AD의 보안 그룹입니다.

AD 보안 그룹	PostgreSQL DB 역할
accounts-group	accounts-role

DB 역할을 AD 보안 그룹에 매핑할 때는 DB 역할에 LOGIN 속성이 설정되어 있고 필수 로그인 데이터베이스에 대한 CONNECT 권한이 있는지 확인해야 합니다.

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

이제 관리자는 AD 보안 그룹과 PostgreSQL DB 역할 간의 매핑을 생성할 수 있습니다.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

AD 보안 그룹의 SID 검색에 대한 자세한 내용은 [PowerShell에서 Active Directory 그룹 SID 검색](#) 섹션을 참조하세요.

AD 사용자가 여러 그룹에 속해 있는 경우가 있을 수 있으며, 이 경우 AD 사용자는 가장 높은 가중치로 프로비저닝된 DB 역할의 권한을 상속받게 됩니다. 두 역할의 가중치가 같으면 AD 사용자는 최근에 추가된 매핑에 해당하는 DB 역할의 권한을 상속받습니다. 개별 DB 역할의 상대적 권한을 반영하는 가중치를 지정하는 것이 좋습니다. DB 역할의 권한이 높을수록 매핑 항목과 관련된 가중치도 높아야 합니다. 이렇게 하면 가중치가 같은 두 매핑의 모호성을 피할 수 있습니다.

다음 표에는 AD 보안 그룹에서 Aurora PostgreSQL DB 역할로의 샘플 매핑이 나와 있습니다.

AD 보안 그룹	PostgreSQL DB 역할	가중치
accounts-group	accounts-role	7
sales-group	sales-role	10
dev-group	dev-role	7

다음 예제에서 `user1`은 가중치가 더 높기 때문에 sales-role의 권한을 상속하고, `user2`는 이 역할에 대한 매핑이 `accounts-role` 이후에 생성되었기 때문에 `accounts-role`과 동일한 가중치를 공유하는 `dev-role`의 권한을 상속합니다.

사용자 이름	보안 그룹 구성원 자격
user1	accounts-group sales-group
user2	accounts-group dev-group

매핑을 설정하고, 나열하고, 지우는 psql 명령은 아래와 같습니다. 현재는 단일 매핑 항목을 수정할 수 없습니다. 기존 항목을 삭제하고 매핑을 다시 만들어야 합니다.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid        | pg_role         | weight | ad_grp
-------------+----------------+--------+---------------
```

```
S-1-5-67-890 | accounts-role | 7       | accounts-group
S-1-2-34-560 | sales-role    | 10      | sales-group
S-1-8-43-612 | dev-role      | 7       | dev-group
(3 rows)
```

AD 사용자 ID 로깅/감사

다음 명령을 사용하여 기존 또는 세션 사용자가 상속하는 데이터베이스 역할을 확인합니다.

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

AD 보안 주체 ID를 확인하려면 아래 명령을 사용하세요.

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

현재로서는 AD 사용자 ID가 감사 로그에 표시되지 않습니다. `log_connections` 파라미터를 활성화하여 DB 세션 설정을 기록할 수 있습니다. 자세한 내용은 log_connections를 참조하세요. 이에 대한 출력에는 아래와 같이 AD 사용자 ID가 포함됩니다. 이 출력과 관련된 백엔드 PID는 실제 AD 사용자에게 작업을 귀속시키는 데 도움이 됩니다.

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

제한 사항

- Azure Active Directory로 알려진 Microsoft Entra ID는 지원되지 않습니다.

`pg_ad_mapping` 확장의 함수 사용

`pg_ad_mapping` 확장은 다음 함수에 대한 지원을 제공합니다.

pgadmap_set_mapping

이 함수는 관련 가중치를 사용하여 AD 보안 그룹과 데이터베이스 역할 간의 매핑을 설정합니다.

구문

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

인수

파라미터	설명
ad_group	AD 그룹의 이름입니다. 값이 null 또는 빈 문자열이 될 수 없습니다.
db_role	지정된 AD 그룹에 매핑할 데이터베이스 역할입니다. 값이 null 또는 빈 문자열이 될 수 없습니다.
ad_group_sid	AD 그룹을 고유하게 식별하는 데 사용되는 보안 식별자입니다. 값은 'S-1-'로 시작하며 null 또는 빈 문자열이 될 수 없습니다. 자세한 내용은 PowerShell에서 Active Directory 그룹 SID 검색 단원을 참조하십시오.
weight	데이터베이스 역할과 관련된 가중치입니다. 사용자가 여러 그룹의 구성원으로 속해 있는 경우 가중치가 가장 높은 역할이 우선합니다. 가중치의 기본값은 1입니다.

반환 타입

None

사용 노트

이 함수는 AD 보안 그룹의 새 매핑을 데이터베이스 역할에 추가합니다. rds_superuser 권한을 가진 사용자만 DB 클러스터의 기본 DB 인스턴스에서 이 함수를 실행할 수 있습니다.

예시

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

이 함수는 `pgadmap_set_mapping` 함수를 사용하여 설정한 AD 보안 그룹과 DB 역할 간의 매핑을 나열합니다.

구문

```
pgadmap_read_mapping()
```

인수

None

반환 타입

파라미터	설명
ad_group_sid	AD 그룹을 고유하게 식별하는 데 사용되는 보안 식별자입니다. 값은 'S-1-'로 시작하며 null 또는 빈 문자열이 될 수 없습니다. 자세히 알아보려면 PowerShell에서 Active Directory 그룹 SID 검색 섹션을 참조하세요.

파라미터	설명
db_role	지정된 AD 그룹에 매핑할 데이터베이스 역할입니다. 값이 null 또는 빈 문자열이 될 수 없습니다.
weight	데이터베이스 역할과 관련된 가중치입니다. 사용자가 여러 그룹의 구성원으로 속해 있는 경우 가중치가 가장 높은 역할이 우선합니다. 가중치의 기본값은 1입니다.
ad_group	AD 그룹의 이름입니다. 값이 null 또는 빈 문자열이 될 수 없습니다.

사용 노트

이 함수를 호출하여 AD 보안 그룹과 DB 역할 간에 사용 가능한 모든 매핑을 나열합니다.

예시

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                                 | pg_role       | weight | ad_grp
---------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678         | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

이 함수는 `pgadmap_set_mapping` 함수를 사용하여 설정된 하나의 매핑 또는 모든 매핑을 재설정합니다.

구문

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

인수

파라미터	설명
ad_group_sid	AD 그룹을 고유하게 식별하는 데 사용되는 보안 식별자입니다.
db_role	지정된 AD 그룹에 매핑할 데이터베이스 역할입니다.
weight	데이터베이스 역할과 관련된 가중치입니다.

인수가 제공되지 않으면 모든 AD 그룹과 DB 역할 매핑이 재설정됩니다. 모든 인수를 제공하거나 어떤 인수도 제공하지 않아야 합니다.

반환 타입

None

사용 노트

특정 AD 그룹과 DB 역할 매핑을 삭제하거나 모든 매핑을 재설정하려면 이 함수를 호출합니다. `rds_superuser` 권한을 가진 사용자만 DB 클러스터의 기본 DB 인스턴스에서 이 함수를 실행할 수 있습니다.

예시

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                          | pg_role      | weight       | ad_grp
-------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10           | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10           | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();
```

```
   ad_sid                          | pg_role      | weight      | ad_grp
-------------------------------+-------------+------------+---------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                          | pg_role      | weight      | ad_grp
-------------------------------+-------------+------------+--------------
 (0 rows)
```

데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션

기존 데이터베이스의 데이터를 Amazon Aurora PostgreSQL 호환 버전 DB 클러스터로 마이그레이션하기 위한 몇 가지 옵션이 있습니다. 마이그레이션할 데이터베이스와 데이터 크기에 따라서도 마이그레이션 옵션이 달라집니다. 마이그레이션 옵션은 다음과 같습니다.

[스냅샷을 사용하여 RDS for PostgreSQL DB 인스턴스 마이그레이션]

RDS for PostgreSQL DB 스냅샷의 데이터를 Aurora PostgreSQL DB 클러스터로 직접 마이그레이션할 수 있습니다.

[Aurora 읽기 전용 복제본을 사용하여 RDS for PostgreSQL DB 인스턴스 마이그레이션]

또한 PostgreSQL DB 인스턴스의 Aurora PostgreSQL 읽기 전용 복제본을 생성하여 RDS for PostgreSQL DB 인스턴스에서 마이그레이션할 수도 있습니다. RDS for PostgreSQL DB 인스턴스와 Aurora PostgreSQL 읽기 전용 복제본 사이의 복제 지연 시간이 0이 되면 복제를 멈출 수 있습니다. 이때부터 읽기 및 쓰기 작업에서 Aurora 읽기 복제본을 독립 실행형 Aurora PostgreSQL DB 클러스터로 사용할 수 있습니다.

[Aurora PostgreSQL로 Amazon S3 데이터 가져오기]

데이터를 Amazon S3에서 Aurora PostgreSQL DB 클러스터에 속한 테이블로 가져와 마이그레이션할 수 있습니다.

PostgreSQL과 호환되지 않는 데이터베이스에서 마이그레이션

AWS Database Migration Service(AWS DMS)를 사용하여 PostgreSQL과 호환되지 않는 데이터베이스의 데이터를 마이그레이션할 수 있습니다. AWS DMS에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서에서 AWS Database Migration Service란 무엇입니까?를 참조하세요.

> **Note**
>
> RDS for PostgreSQL에서 마이그레이션하는 동안에는 Aurora PostgreSQL DB 클러스터에서 Kerberos 인증을 활성화하는 기능이 현재 지원되지 않습니다. Kerberos 인증은 독립 실행형 Aurora PostgreSQL DB 클러스터에서만 활성화할 수 있습니다.

Aurora를 사용할 수 있는 AWS 리전 목록은 AWS 일반 참조에서 Amazon Aurora 섹션을 참조하세요.

> **Important**
>
> 가까운 장래에 RDS for PostgreSQL DB 인스턴스를 Aurora PostgreSQL DB 클러스터로 마이그레이션하려는 경우 마이그레이션 계획 단계 초기에 DB 인스턴스의 자동 마이너 버전 업그레이드를 비활성화하는 것이 좋습니다. Aurora PostgreSQL에서 해당 RDS for PostgreSQL 버전이 아직 지원되지 않는 경우 Aurora PostgreSQL로의 마이그레이션이 지연될 수 있습니다. Aurora PostgreSQL 버전에 대한 자세한 내용은 Amazon Aurora PostgreSQL의 엔진 버전을 참조하세요.

RDS for PostgreSQL DB 인스턴스의 스냅샷을 Aurora PostgreSQL DB 클러스터로 마이그레이션합니다.

Aurora PostgreSQL DB 클러스터를 생성하려면 RDS for PostgreSQL DB 인스턴스의 DB 스냅샷을 마이그레이션할 수 있습니다. 새롭게 생성된 Aurora PostgreSQL DB 클러스터는 원본 RDS for PostgreSQL DB 인스턴스의 데이터로 채워집니다. DB 스냅샷 생성에 대한 자세한 내용은 DB 스냅샷 생성을 참조하세요.

경우에 따라 DB 스냅샷이 데이터를 저장할 AWS 리전에 속하지 않을 수도 있습니다. 이때는 Amazon RDS 콘솔을 사용하여 DB 스냅샷을 해당 AWS 리전으로 복사합니다. DB 스냅샷 복사에 대한 자세한 내용은 DB 스냅샷 복사를 참조하세요.

지정된 AWS 리전에서 사용 가능한 Aurora PostgreSQL 버전과 호환되는 RDS for PostgreSQL 스냅샷을 마이그레이션할 수 있습니다. 예를 들어, RDS PostgreSQL 11.1 DB 인스턴스의 스냅샷은 미국 서부(캘리포니아 북부)에서 Aurora PostgreSQL 버전 11.4, 11.7, 11.8 또는 11.9로 마이그레이션할 수 있습니다. RDS PostreSQL 10.11 스냅샷은 Aurora PostgreSQL 10.11, 10.12, 10.13, 10.14로 마이그레이션할 수 있습니다. 즉, RDS PostgreSQL 스냅샷은 Aurora PostgreSQL과 같거나 낮은 마이너 버전을 사용해야 합니다.

또한 AWS KMS key를 사용하여 새 Aurora PostgreSQL DB 클러스터가 유휴 상태에서 암호화되도록 선택할 수도 있습니다. 이 옵션은 암호화되지 않은 DB 스냅샷에만 가능합니다.

RDS for PostgreSQL DB 스냅샷을 Aurora PostgreSQL DB 클러스터로 마이그레이션하려면 AWS Management Console, AWS CLI 또는 RDS API를 사용합니다. AWS Management Console을 사용할 경우 콘솔은 DB 클러스터와 프라이머리 인스턴스를 모두 생성하는 데 필요한 작업을 수행합니다.

콘솔

RDS 콘솔을 사용해 PostgreSQL DB 스냅샷을 마이그레이션하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. [Snapshots]를 선택합니다.
3. [스냅샷(Snapshots)] 페이지에서 Aurora PostgreSQL DB 클러스터로 마이그레이션하려는 RDS for PostgreSQL 스냅샷을 선택합니다.
4. 작업 을 선택한 다음 스냅샷 마이그레이션을 선택합니다.
5. [Migrate Database] 페이지에서 다음과 같이 값을 설정합니다.

   - DB 엔진 버전: 마이그레이션된 새 인스턴스에 사용할 DB 엔진 버전을 선택합니다.
   - [DB 인스턴스 식별자(DB instance identifier)]: 선택한 AWS 리전의 계정에 대해 고유한 DB 클러스터의 이름을 입력합니다. 이 식별자는 DB 클러스터에 속한 인스턴스의 엔드포인트 주소로 사용됩니다. 선택한 AWS 리전 및 DB 엔진 포함(예: **`aurora-cluster1`**) 등의 몇 가지 지능적 요소를 이름에 추가할 수 있습니다.

     DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다.

     - 1–63자의 영숫자 또는 하이픈으로 구성되어야 합니다.
     - 첫 번째 문자는 글자이어야 합니다.
     - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
     - 각 AWS 리전별로 AWS 계정마다 모든 DB 인스턴스가 고유해야 합니다.

- DB 인스턴스 클래스: 데이터베이스에 필요한 스토리지와 용량을 지닌 DB 인스턴스 클래스를 선택합니다(예: `db.r6g.large`). 데이터베이스의 데이터 용량이 늘어날수록 Aurora 클러스터 볼륨도 자동 확장됩니다. 따라서 현재 스토리지 요구 사항에 맞는 DB 인스턴스 클래스를 선택해야 합니다. 자세한 내용은 Amazon Aurora 스토리지 개요 단원을 참조하십시오.
- Virtual Private Cloud(VPC): 기존 VPC가 있는 경우, VPC 식별자(예: `vpc-a464d1c1`)를 선택하여 해당 VPC를 Aurora PostgreSQL DB 클러스터에 사용할 수 있습니다. VPC 생성에 대한 내용은 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 섹션을 참조하세요.

  기존 VPC가 없다면 [Create a new VPC]를 선택하여 Amazon RDS에서 VPC를 새로 생성하도록 할 수 있습니다.
- DB 서브넷 그룹: 기존 서브넷 그룹이 있으면 해당 서브넷 그룹 식별자(예: `gs-subnet-group1`)를 선택하여 Aurora PostgreSQL DB 클러스터에 기존 서브넷 그룹을 사용할 수 있습니다.
- 퍼블릭 액세스 가능: VPC에 있는 리소스만 DB 클러스터의 인스턴스에 액세스할 수 있도록 하려면 아니요를 선택합니다. 퍼블릭 네트워크에 있는 리소스가 DB 클러스터의 인스턴스에 액세스할 수 있도록 하려면 예를 선택합니다.

  > **Note**
  >
  > 퍼블릭 서브넷에서는 프로덕션 DB 클러스터가 필요 없을 수도 있습니다. 애플리케이션 서버만 DB 클러스터에 액세스하기 때문입니다. DB 클러스터가 퍼블릭 서브넷에 필요 없는 경우에는 [Publicly Accessible]을 아니오로 설정합니다.
- VPC 보안 그룹: VPC 보안 그룹을 선택하여 데이터베이스에 대한 액세스를 허용합니다.
- 가용 영역: Aurora PostgreSQL DB 클러스터의 기본 인스턴스를 호스팅할 가용 영역을 선택합니다. Amazon RDS가 가용 영역을 선택하도록 하려면 기본 설정 없음을 선택합니다.
- 데이터베이스 포트: Aurora PostgreSQL DB 클러스터의 인스턴스에 연결할 때 사용할 기본 포트를 입력합니다. 기본값은 `5432`입니다.

  > **Note**
  >
  > 기업 방화벽 뒤에 있어서 PostgreSQL 기본 포트인 5432 같은 기본 포트에 액세스하지 못할 수도 있습니다. 이런 경우에는 기업 방화벽이 허용하는 포트 값을 입력합니다. 나중에 Aurora PostgreSQL DB 클러스터에 연결할 때도 필요하므로 이 포트 값을 기억해야 합니다.

- 암호화 활성화: 새 Aurora PostgreSQL DB 클러스터를 저장 상태에서 암호화하려면 암호화 활성화를 선택합니다. 또한 KMS 키를 AWS KMS key 값으로 선택합니다.
- 마이너 버전 자동 업그레이드: PostgreSQL DB 엔진의 마이너 버전 업그레이드가 있을 때 Aurora PostgreSQL DB 클러스터가 업그레이드를 자동으로 수신하도록 하려면 마이너 버전 자동 업그레이드 사용을 선택합니다.

  마이너 버전 자동 업그레이드 옵션은 Aurora PostgreSQL DB 클러스터에 대해 PostgreSQL 엔진 마이너 버전으로의 업그레이드에만 적용됩니다. 시스템 안정성 유지를 위한 정기 패치에는 적용되지 않습니다.

6. [Migrate]를 선택하여 DB 스냅샷을 마이그레이션합니다.
7. 새 DB 클러스터를 보려면 [데이터베이스] 를 선택합니다. 새 DB 클러스터를 선택하여 마이그레이션 진행률을 모니터링합니다. 마이그레이션이 완료되면 클러스터의 상태가 사용 가능으로 표시됩니다. 연결 및 보안 탭에서 DB 클러스터의 기본 작성기 인스턴스에 연결하는 데 사용할 클러스터 엔드포인트를 찾을 수 있습니다. Aurora PostgreSQL DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

AWS CLI

AWS CLI를 사용하여 RDS for PostgreSQL DB 스냅샷을 Aurora PostgreSQL로 마이그레이션하려면 2개의 개별 AWS CLI 명령이 필요합니다. 먼저 `restore-db-cluster-from-snapshot` AWS CLI 명령을 사용하여 새 Aurora PostgreSQL DB 클러스터를 생성합니다. 그런 다음 `create-db-instance` 명령을 사용하여 새 클러스터에 프라이머리 DB 인스턴스를 생성하여 마이그레이션을 완료합니다. 다음 절차에서는 스냅샷 생성에 사용된 DB 인스턴스와 구성이 동일한 프라이머리 DB 인스턴스로 Aurora PostgreSQL DB 클러스터를 생성합니다.

Aurora PostgreSQL DB 클러스터로 RDS for PostgreSQL DB 스냅샷 마이그레이션

1. describe-db-snapshots 명령을 사용하여 마이그레이션하려는 DB 스냅샷에 대한 정보를 가져옵니다. 명령에 `--db-instance-identifier` 파라미터 또는 `--db-snapshot-identifier`를 지정할 수 있습니다. 이러한 파라미터 중 하나를 지정하지 않으면 모든 스냅샷을 가져옵니다.

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. 이 명령은 지정된 DB 인스턴스에서 생성된 모든 스냅샷에 대한 모든 구성 세부 정보를 반환합니다. 출력에서 마이그레이션할 스냅샷을 찾고 Amazon 리소스 이름(ARN)을 찾습니다. Amazon RDS ARN에 대해 자세히 알아보려면 Amazon Relational Database Service(Amazon RDS)를 참조하세요. ARN은 다음 출력과 유사합니다.

```
“DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
```

또한 출력에서 엔진 버전, 할당된 스토리지, DB 인스턴스 암호화 여부와 같은 RDS for PostgreSQL DB 인스턴스에 대한 구성 세부 정보를 찾을 수 있습니다.

3. restore-db-cluster-from-snapshot 명령을 사용하여 마이그레이션을 시작합니다. 다음 파라미터를 지정합니다.

   - `--db-cluster-identifier` - Aurora PostgreSQL DB 클러스터에 지정하려는 이름입니다. 이 Aurora DB 클러스터는 DB 스냅샷 마이그레이션의 대상입니다.
   - `--snapshot-identifier` - 마이그레이션할 DB 스냅샷의 Amazon 리소스 이름(ARN)입니다.
   - `--engine` - Aurora DB 클러스터 엔진에 대해 `aurora-postgresql`을 지정합니다.
   - `--kms-key-id` - Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 이 선택적 파라미터를 사용하면 암호화되지 않은 DB 스냅샷에서 암호화된 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 또한 DB 스냅샷에 사용된 키와 DB 클러스터에 대해 다른 암호화 키를 선택할 수 있습니다.

   > **Note**
   >
   > 암호화된 DB 스냅샷에서 암호화되지 않은 Aurora PostgreSQL DB 클러스터를 생성할 수 없습니다.

   다음과 같이 `--kms-key-id` 파라미터가 지정되지 않은 경우 restore-db-cluster-from-snapshot AWS CLI 명령은 DB 스냅샷과 동일한 키를 사용하여 암호화되거나 원본 DB 스냅샷이 암호화되지 않은 경우 암호화되지 않은 빈 Aurora PostgreSQL DB 클러스터를 생성합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name \
    --engine aurora-postgresql
```

   Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
```

```
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name ^
    --engine aurora-postgresql
```

4. 이 명령은 마이그레이션을 위해 생성 중인 Aurora PostgreSQL DB 클러스터에 대한 세부 정보를 반환합니다. describe-db-clusters AWS CLI 명령을 사용하여 Aurora PostgreSQL DB 클러스터의 상태를 확인할 수 있습니다.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. DB 클러스터가 "사용 가능" 상태가 되면 create-db-instance 명령을 사용하여 Amazon RDS DB 스냅샷을 기반으로 하는 DB 인스턴스로 Aurora PostgreSQL DB 클러스터를 채웁니다. 다음 파라미터를 지정합니다.

   - `--db-cluster-identifier` - 이전 단계에서 생성한 새 Aurora PostgreSQL DB 클러스터의 이름입니다.
   - `--db-instance-identifier` - DB 인스턴스에 부여할 이름입니다. 이 인스턴스는 Aurora PostgreSQL DB 클러스터의 프라이머리 노드가 됩니다.
   - `----db-instance-class` - 사용할 DB 인스턴스 클래스를 지정합니다. 마이그레이션할 Aurora PostgreSQL 버전에서 지원하는 DB 인스턴스 클래스 중에서 선택합니다. 자세한 내용은 DB 인스턴스 클래스 유형 및 DB 인스턴스 클래스에 지원되는 DB 엔진 단원을 참조하세요.
   - `--engine` - DB 인스턴스에 대해 `aurora-postgresql`을 지정합니다.

   `create-db-instance` AWS CLI 명령에 적절한 옵션을 전달하여 원본 DB 스냅샷과 다른 구성으로 DB 인스턴스를 생성할 수도 있습니다. 자세한 내용은 create-db-instance 명령을 참조하세요.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

   Windows의 경우:

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
```

```
    --engine aurora-postgresql
```

마이그레이션 프로세스가 완료되면 Aurora PostgreSQL 클러스터에 채워진 프라이머리 DB 인스턴스가 있습니다.

Aurora 읽기 전용 복제본을 사용하여 RDS for PostgreSQL DB 인스턴스의 데이터를 Aurora PostgreSQL DB 클러스터로 마이그레이션

마이그레이션 프로세스용 Aurora 읽기 전용 복제본을 사용하여 새로운 Aurora PostgreSQL DB 클러스터의 기반으로 RDS for PostgreSQL DB 인스턴스를 사용할 수 있습니다. Aurora 읽기 전용 복제본 옵션은 동일한 AWS 리전 및 계정 내에서 마이그레이션하는 데 만 사용할 수 있으며, 리전에서 RDS for PostgreSQL DB 인스턴스에 대해 호환되는 버전의 Aurora PostgreSQL DB 인스턴스를 제공하는 경우에만 사용할 수 있습니다. 여기에서 호환은 Aurora PostgreSQL 버전이 RDS for PostgreSQL 버전과 동일하거나 동일한 메이저 버전 제품군에서 더 높은 마이너 버전임을 의미합니다.

예를 들어 이 기술을 사용하여 RDS for PostgreSQL 11.14 DB 인스턴스를 마이그레이션하려면 리전에서 PostgreSQL 버전 11 제품군의 Aurora PostgreSQL 버전 11.14 이상의 마이너 버전을 제공해야 합니다.

주제

- Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 개요
- Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 준비
- Aurora 읽기 전용 복제본 생성
- Aurora 읽기 전용 복제본 승격

Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 개요

RDS for PostgreSQL DB 인스턴스에서 Aurora PostgreSQL DB 클러스터로의 마이그레이션은 절차가 여러 단계 있습니다. 먼저 소스 RDS for PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성합니다. 그러면 RDS for PostgreSQL DB 인스턴스에서 복제본 클러스터로 알려진 특수 용도 DB 클러스터로 복제 프로세스를 시작합니다. 복제본 클러스터는 Aurora 읽기 전용 복제본(리더 인스턴스)로만 구성됩니다.

> **Note**
> 마이그레이션이 완료되려면 데이터의 테라바이트당 몇 시간이 걸릴 수 있습니다.

Aurora PostgreSQL 복제본 승격

Aurora PostgreSQL DB 클러스터를 만든 후 다음 단계에 따라 Aurora 복제본을 승격합니다.

1. 소스 RDS for PostgreSQL DB 인스턴스에서 모든 데이터베이스 쓰기 워크로드를 중지합니다.
2. 소스 RDS for PostgreSQL DB 인스턴스에서 현재 `WAL LSN`을 가져옵니다.

```
SELECT pg_current_wal_lsn();
pg_current_wal_lsn
--------------------
0/F0000318
(1 row)
```

3. Aurora PostgreSQL 복제본 클러스터에서 재생된 LSN이 2단계의 LSN보다 큰지 확인합니다.

```
SELECT pg_last_wal_replay_lsn();
pg_last_wal_replay_lsn
------------------------
0/F0000400
(1 row)
```

   또는 소스 RDS for PostgreSQL DB 인스턴스에서 다음 쿼리를 사용할 수 있습니다.

```
SELECT restart_lsn FROM pg_replication_slots;
```

4. Aurora PostgreSQL 복제본 클러스터를 승격합니다.

   복제가 중지되면 복제본 클러스터가 독립 실행형 Aurora PostgreSQL DB 클러스터로 승격되고, 리더가 클러스터의 라이터 인스턴스로 승격됩니다. 이때 Aurora PostgreSQL DB 클러스터에 인스턴스를 추가하여 사용 사례에 따라 크기를 조정할 수 있습니다. 원래 RDS for PostgreSQL DB 인스턴스가 더 이상 필요하지 않은 경우 삭제할 수 있습니다.

RDS for PostgreSQL DB 인스턴스에 Aurora 읽기 전용 복제본이 이미 있거나 교차 리전 읽기 전용 복제본이 있는 경우 Aurora 읽기 전용 복제본을 생성할 수 없습니다.

Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 준비

Aurora 읽기 전용 복제본을 사용하는 마이그레이션 프로세스 중에 소스 RDS for PostgreSQL DB 인스턴스에 적용된 업데이트는 복제본 클러스터의 Aurora 읽기 전용 복제본에 비동기식으로 복제됩니다.

이 프로세스는 소스 인스턴스에 미리 쓰기 로그(WAL) 세그먼트를 저장하는 PostgreSQL의 기본 스트리밍 복제 기능을 사용합니다. 이 마이그레이션 프로세스를 시작하기 전에 테이블에 나열된 지표 값을 확인하여 인스턴스의 스토리지 용량이 충분한지 확인해야 합니다.

지표	설명
FreeStorageSpace	사용 가능한 스토리지 공간. 단위: 바이트
OldestReplicationSlotLag	가장 지연된 복제본에 있는 WAL 데이터의 지연 크기. 단위: 메가바이트
RDSToAuroraPostgreSQLReplicaLag	Aurora PostgreSQL DB 클러스터가 원본 RDS DB 인스턴스보다 늦어지는 시간(초).
TransactionLogsDiskUsage	트랜잭션 로그에 사용된 디스크 공간. 단위: 메가바이트

RDS 인스턴스 모니터링에 대한 자세한 내용은 Amazon RDS 사용 설명서에서 모니터링 단원을 참조하십시오.

Aurora 읽기 전용 복제본 생성

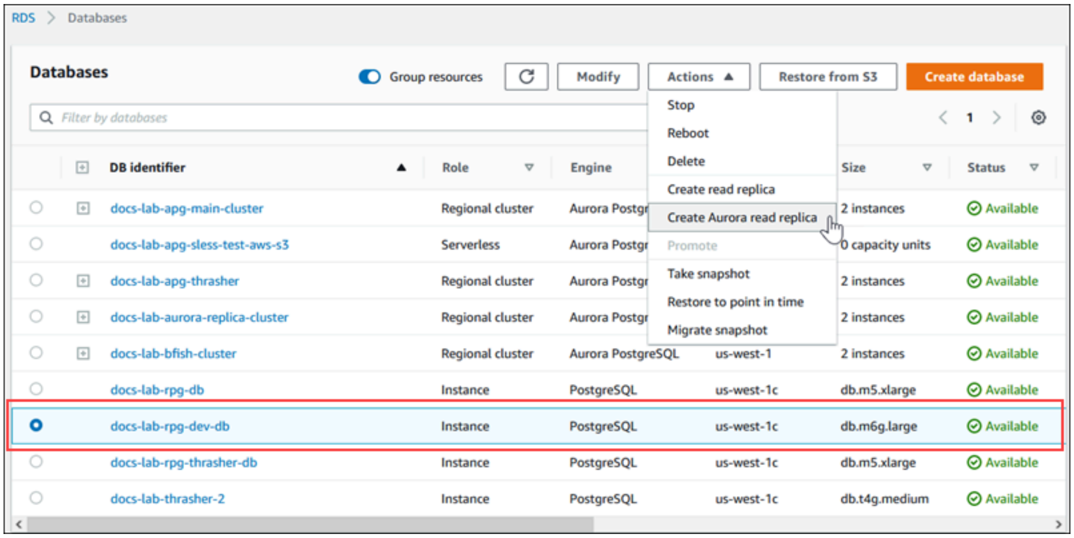
RDS for PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본은 AWS Management Console 또는 AWS CLI를 사용해 생성할 수 있습니다. AWS Management Console을 사용하여 Aurora 읽기 전용 복제본을 생성하는 옵션은 AWS 리전에서 호환되는 Aurora PostgreSQL 버전을 제공하는 경우에만 이용 가능합니다. 이 경우, Aurora PostgreSQL 버전이 RDS for PostgreSQL 버전과 동일하거나 동일한 주 버전 제품군에서 더 높은 마이너 버전일 경우에만 가능합니다.

콘솔

원본 PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하는 방법은 다음과 같습니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

3. Aurora 읽기 전용 복제본의 소스로 사용할 RDS for PostgreSQL DB 인스턴스를 선택합니다. 작업에서 Aurora 읽기 전용 복제본 생성을 선택합니다. 이 선택 항목이 표시되지 않으면 호환되는 Aurora PostgreSQL 버전이 리전에서 사용할 수 없음을 의미합니다.



4. Aurora 읽기 전용 복제본 설정 생성 페이지에서 다음 표와 같이 Aurora PostgreSQL DB 클러스터의 속성을 구성합니다. 복제본 DB 클러스터는 소스 DB 인스턴스의 스냅샷으로 소스와 동일한 '마스터' 사용자 이름과 암호를 사용하여 만드는 것이기 때문에 현재는 변경할 수 없습니다.

옵션	설명
DB 인스턴스 클래스	DB 클러스터의 기본 인스턴스에 대한 처리 및 메모리 요건을 충족하는 DB 인스턴스 클래스를 선택합니다. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.
다중 AZ 배포	마이그레이션 중에는 사용할 수 없음
DB 인스턴스 식별자	DB 인스턴스에 부여할 이름을 입력합니다. 이 식별자는 새 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다. DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다. • 1–63자의 영숫자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다.

옵션	설명
	• 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • 개별 AWS 리전별로 각 AWS 계정의 모든 DB 인스턴스에 대해 고유해야 합니다.
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS에서 VPC를 생성하도록 하려면 새 VPC 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.
DB 서브넷 그룹	DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다. Amazon RDS에서 자동으로 DB 서브넷 그룹을 생성하도록 하려면 새 DB 서브넷 그룹 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 단원을 참조하세요.
퍼블릭 액세스 가능성	DB 클러스터에 퍼블릭 IP 주소를 할당하려면 [예(Yes)]를 선택하고 그렇지 않으면 [아니요(No)]를 선택합니다. DB 클러스터의 인스턴스는 퍼블릭 DB 인스턴스와 프라이빗 DB 인스턴스를 모두 조합하여 사용할 수 있습니다. 모든 사용자의 액세스에서 인스턴스를 숨기는 방법에 대한 자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 단원을 참조하십시오.
가용 영역	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 단원을 참조하십시오.
VPC 보안 그룹	VPC 보안 그룹을 한 개 이상 선택하여 DB 클러스터에 대한 네트워크 액세스를 보안합니다. Amazon RDS에서 VPC 보안 그룹을 생성하게 하려면 새 VPC 보안 그룹 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.

옵션	설명
데이터베이스 포트	애플리케이션과 유틸리티가 데이터베이스에 액세스할 때 사용할 포트를 지정합니다. Aurora PostgreSQL DB 클러스터는 기본적으로 PostgreSQL 포트, 5432로 지정됩니다. 일부 기업에서는 방화벽으로 이 포트에 대한 연결을 차단합니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 새로운 DB 클러스터에 다른 포트를 선택해야 합니다.
DB 파라미터 그룹	Aurora PostgreSQL DB 클러스터에 사용할 DB 파라미터 그룹을 선택합니다. Aurora에 사용할 수 있는 기본 파라미터 그룹이 포함되어 있거나, 직접 파라미터 그룹을 생성할 수 있습니다. DB 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.
DB 클러스터 파라미터 그룹	Aurora PostgreSQL DB 클러스터에 사용할 DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본값으로 사용할 수 있는 DB 클러스터 파라미터 그룹을 제공하며, 자체 DB 클러스터 파라미터 그룹을 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하십시오.
암호화	새 Aurora DB 클러스터를 유휴 상태에서 암호화하려면 암호화 활성화를 선택합니다. 암호화 사용을 선택한 경우 KMS 키를 AWS KMS key 값으로 선택합니다.
우선 순위	DB 클러스터의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스의 결함으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.

옵션	설명
백업 보관 기간	Aurora가 데이터베이스 백업 사본을 보관하는 기간을 1~35일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹션을 참조하세요.
역할 모니터링	확장 모니터링 활성화를 선택하는 경우에만 사용할 수 있습니다. Enhanced Monitoring에 사용할 AWS Identity and Access Management(IAM) 역할입니다. 자세한 내용은 Enhanced Monitoring 설정 및 활성화 섹션을 참조하세요.
세부 수준	확장 모니터링 활성화를 선택하는 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
마이너 버전 자동 업그레이드	PostgreSQL DB 엔진의 마이너 버전 업그레이드가 있을 때 Aurora PostgreSQL DB 클러스터가 자동으로 업그레이드를 받도록 하려면 예를 선택합니다. 마이너 버전 자동 업그레이드 옵션은 Aurora PostgreSQL DB 클러스터에 대해 PostgreSQL 엔진 마이너 버전으로의 업그레이드에만 적용됩니다. 시스템 안정성 유지를 위한 정기 패치에는 적용되지 않습니다.
유지보수 윈도우	시스템 유지 관리를 실행하는 기간을 주 단위로 선택합니다.

5. Create read replica(읽기 전용 복제본 생성)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 소스 RDS for PostgreSQL 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 먼저 create-db-cluster CLI 명령을 사용하여 빈 Aurora DB 클러스터를 생성합니다. DB 클러스터가

존재하면 create-db-instance 명령을 사용하여 DB 클러스터의 기본 인스턴스를 생성합니다. 기본 인스턴스는 클러스터에 생성된 최초의 Aurora DB 인스턴스입니다. 이 경우 처음에는 RDS for PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본으로 생성됩니다. 프로세스가 끝나면 RDS for PostgreSQL DB 인스턴스는 Aurora PostgreSQL DB 클러스터로 효과적으로 마이그레이션된 상태입니다.

기본 사용자 계정(일반적으로 `postgres`), 암호 또는 데이터베이스 이름을 지정할 필요가 없습니다. Aurora 읽기 전용 복제본이 사용자가 AWS CLI 명령을 호출할 때 식별되는 소스 RDS for PostgreSQL DB 인스턴스에서 자동으로 해당 정보를 가져옵니다.

Aurora PostgreSQL DB 클러스터와 DB 인스턴스에 사용할 엔진 버전은 사용자가 지정해야 합니다. 지정한 버전은 소스 RDS for PostgreSQL DB 인스턴스와 일치해야 합니다. 소스 RDS for PostgreSQL DB 인스턴스가 암호화되면 Aurora PostgreSQL DB 클러스터 기본 인스턴스에 대한 암호화도 지정해야 합니다. 암호화된 인스턴스를 암호화되지 않은 Aurora DB 클러스터로 마이그레이션하는 것은 지원되지 않습니다.

다음 예제는 암호화되지 않은 RDS DB 원본 인스턴스를 사용하는 `my-new-aurora-cluster`라는 Aurora PostgreSQL DB 클러스터를 생성합니다. 먼저 `create-db-cluster` CLI 명령을 호출하여 Aurora PostgreSQL DB 클러스터를 생성합니다. 이 예에서는 선택적으로 `--storage-encrypted` 파라미터를 사용하여 DB 클러스터를 암호화해야 한다는 것을 지정하는 방법을 보여줍니다. 소스 DB가 암호화되지 않았기 때문에 사용할 키를 지정하는 데 `--kms-key-id`가 사용됩니다. 필수 및 선택적 파라미터에 대한 자세한 내용은 예시 아래의 목록을 참조하세요.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster  \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Windows의 경우:

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
```

```
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

다음 목록에서 예시에 표시된 일부 옵션에 대한 자세한 정보를 찾을 수 있습니다. 별도로 지정하지 않는 한 이러한 파라미터가 필요합니다.

- `--db-cluster-identifier` - 새 Aurora PostgreSQL DB 클러스터에 이름을 지정해야 합니다.
- `--db-subnet-group-name` - 소스 DB 인스턴스와 동일한 DB 서브넷에 Aurora PostgreSQL DB 클러스터를 생성합니다.
- `--vpc-security-group-ids` - Aurora PostgreSQL DB 클러스터의 보안 그룹을 지정합니다.
- `--engine-version` - Aurora PostgreSQL DB 클러스터에 사용할 버전을 지정합니다. 소스 RDS for PostgreSQL DB 인스턴스에서 사용하는 버전과 같거나 그보다 더 높은 마이너 버전이어야 합니다.
- `--replication-source-identifier` - Amazon 리소스 이름(ARN)을 사용하여 RDS for PostgreSQL DB 인스턴스를 식별합니다. Amazon RDS ARN에 대한 자세한 내용은 DB 클러스터의 AWS 일반 참조에서 [Amazon Relational Database Service(RDS)](#) 섹션을 참조하세요.
- `--storage-encrypted` – 선택 사항입니다. 암호화를 지정하는 데 필요한 경우에만 다음과 같이 사용합니다.
  - 소스 DB 인스턴스에 암호화된 스토리지가 있는 경우 이 파라미터를 사용합니다. 암호화된 스토리지가 있는 소스 DB 인스턴스에서 이 파라미터를 사용하지 않으면 `create-db-cluster`에 대한 호출이 실패합니다. Aurora PostgreSQL DB 클러스터에 소스 DB 인스턴스에서 사용하는 키와 다른 키를 사용하려면 `--kms-key-id`도 지정해야 합니다.
  - 소스 DB 인스턴스의 스토리지가 암호화되지 않았지만 Aurora PostgreSQL DB 클러스터에서 암호화를 사용하도록 하려는 경우에 사용합니다. 그렇다면 `--kms-key-id` 파라미터와 함께 사용할 암호화 키도 식별해야 합니다.
- `--kms-key-id` – 선택 사항입니다. 사용할 경우 ARN, ID, 별칭 ARN 또는 해당 별칭 이름을 사용하여 스토리지 암호화(`--storage-encrypted`)에 사용할 키를 지정할 수 있습니다. 이 파라미터는 다음 상황에서만 필요합니다.
  - Aurora PostgreSQL DB 클러스터에 대해 소스 DB 인스턴스에서 사용하는 키와 다른 키를 선택합니다.

- 암호화되지 않은 소스에서 암호화된 클러스터를 만드는 방법 이 경우 Aurora PostgreSQL이 암호화에 사용해야 하는 키를 지정해야 합니다.

Aurora PostgreSQL DB 클러스터를 생성한 후 다음과 같이 `create-db-instance` CLI 명령을 사용하여 기본 인스턴스를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

Windows의 경우:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

다음 목록에서 예시에 표시된 일부 옵션에 대한 자세한 정보를 찾을 수 있습니다.

- `--db-cluster-identifier` - 이전 단계에서 `create-db-instance` 명령으로 생성한 Aurora PostgreSQL DB 클러스터의 이름을 지정합니다.
- `--db-instance-class` - 기본 인스턴스에 사용할 DB 인스턴스 클래스의 이름입니다(예: `db.r4.xlarge`, `db.t4g.medium`, `db.x2g.16xlarge` 등). 사용 가능한 인스턴스 클래스 목록은 DB 인스턴스 클래스 유형 섹션을 참조하세요.
- `--db-instance-identifier` - 기본 인스턴스의 이름을 지정합니다.
- `--engine aurora-postgresql` - 엔진에 대한 `aurora-postgresql`을 지정합니다.

RDS API

소스 RDS for PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 먼저 `CreateDBCluster` RDS API 작업을 사용하여 소스 RDS for PostgreSQL DB 인스턴스에서 생성되는 Aurora 읽기 전용 복제본의 새 Aurora DB 클러스터를 생성합니다. Aurora PostgreSQL DB 클러스터를

사용할 수 있게 되면 CreateDBInstance를 사용하여 Aurora DB 클러스터의 기본 인스턴스를 생성합니다.

기본 사용자 계정(일반적으로 postgres), 암호 또는 데이터베이스 이름을 지정할 필요가 없습니다. Aurora 읽기 전용 복제본이 사용자가 ReplicationSourceIdentifier로 지정된 소스 RDS for PostgreSQL DB 인스턴스에서 자동으로 해당 정보를 가져옵니다.

Aurora PostgreSQL DB 클러스터와 DB 인스턴스에 사용할 엔진 버전은 사용자가 지정해야 합니다. 지정한 버전은 소스 RDS for PostgreSQL DB 인스턴스와 일치해야 합니다. 소스 RDS for PostgreSQL DB 인스턴스가 암호화되면 Aurora PostgreSQL DB 클러스터 기본 인스턴스에 대한 암호화도 지정해야 합니다. 암호화된 인스턴스를 암호화되지 않은 Aurora DB 클러스터로 마이그레이션하는 것은 지원되지 않습니다.

Aurora 읽기 전용 복제본의 Aurora DB 클러스터를 생성하려면 다음 파라미터와 함께 CreateDBCluster RDS API 작업을 사용합니다.

- DBClusterIdentifier - 생성할 DB 클러스터의 이름입니다.
- DBSubnetGroupName - 이 DB 클러스터와 연결할 DB 서브넷 그룹의 이름입니다.
- Engine=aurora-postgresql - 사용할 엔진 이름입니다.
- ReplicationSourceIdentifier - 원본 PostgreSQL DB 인스턴스의 Amazon 리소스 이름(ARN)입니다. Amazon RDS ARN에 대한 자세한 내용은 Amazon Web Services 일반 참조에서 Amazon Relational Database Service(RDS) 섹션을 참조하세요. ReplicationSourceIdentifier가 암호화된 소스를 식별하는 경우, 사용자가 KmsKeyId 옵션을 사용하여 다른 키를 지정하지 않는 한 Amazon RDS가 기본 KMS 키를 사용합니다.
- VpcSecurityGroupIds - DB 클러스터와 연결할 Amazon EC2 VPC 보안 그룹 목록입니다.
- StorageEncrypted - DB 클러스터의 암호화 여부를 나타냅니다. ReplicationSourceIdentifier를 지정하지 않고 이 파라미터를 사용할 경우, Amazon RDS는 기본 KMS 키를 사용합니다.
- KmsKeyId - 암호화된 클러스터의 키입니다. 사용할 경우 ARN, ID, 별칭 ARN 또는 해당 별칭 이름을 사용하여 스토리지 암호화에 사용할 키를 지정할 수 있습니다.

자세한 내용은 Amazon RDS API 참조에서 CreateDBCluster 섹션을 참조하세요.

Aurora DB 클러스터를 사용할 수 있게 되면 CreateDBInstance RDS API 작업을 다음 파라미터와 함께 사용하여 기본 인스턴스를 생성할 수 있습니다.

- DBClusterIdentifier - DB 클러스터의 이름입니다.

- `DBInstanceClass` - 기본 인스턴스에 사용할 DB 인스턴스 클래스의 이름입니다.
- `DBInstanceIdentifier` - 기본 인스턴스의 이름입니다.
- `Engine=aurora-postgresql` - 사용할 엔진 이름입니다.

자세한 내용은 Amazon RDS API 참조에서 `CreateDBInstance` 섹션을 참조하세요.

Aurora 읽기 전용 복제본 승격

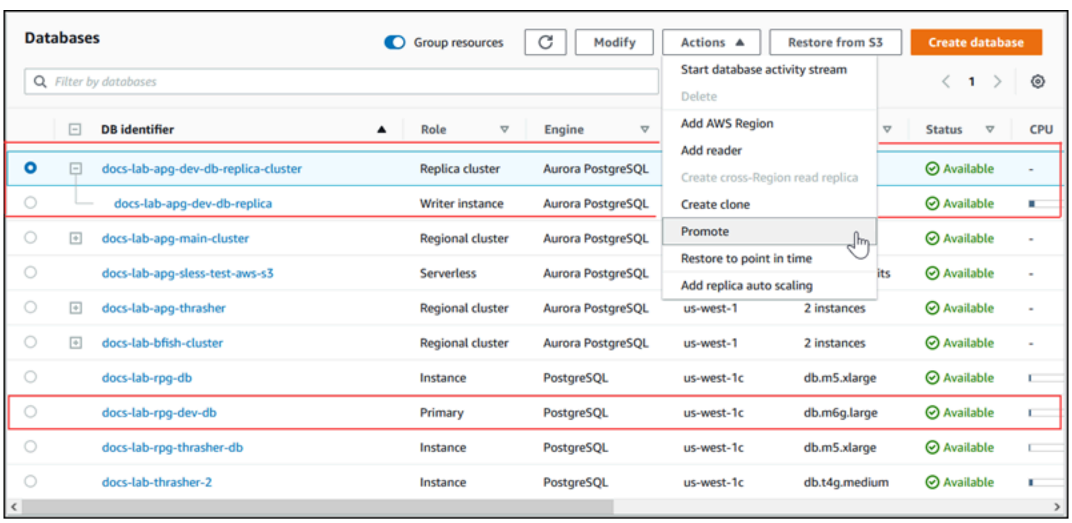
Aurora PostgreSQL로의 마이그레이션은 복제본 클러스터를 승격하기 전까지는 완료되지 않으므로 RDS for PostgreSQL 소스 DB 인스턴스를 아직 삭제하지 마세요.

복제본 클러스터를 승격하기 전에 RDS for PostgreSQL DB 인스턴스에 진행 중인 트랜잭션이나 데이터베이스 쓰기 작업이 없는지 확인하세요. Aurora 읽기 전용 복제본의 복제본 지연이 0일 때 복제본 클러스터를 승격할 수 있습니다. 복제본 지연 모니터링에 대한 자세한 내용은 Aurora PostgreSQL 복제 모니터링 및 Amazon Aurora에 대한 인스턴스 수준 지표 섹션을 참조하세요.

콘솔

Aurora 읽기 전용 복제본을 Aurora DB 클러스터로 승격시키려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 복제본 클러스터를 선택합니다.



4. Actions(작업)에서 Promote(승격)를 선택합니다. 몇 분 정도 소요될 수 있으며 가동 중지가 발생할 수 있습니다.

프로세스가 완료되면 Aurora 복제본 클러스터는 Regional Aurora PostgreSQL DB 클러스터이며, 라이터 인스턴스에는 RDS for PostgreSQL DB 인스턴스의 데이터가 들어 있습니다.

AWS CLI

Aurora 읽기 전용 복제본을 독립형 DB 클러스터로 승격하려면 [promote-read-replica-db-cluster] AWS CLI 명령을 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Windows의 경우:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

RDS API

Aurora 읽기 복제본을 독립형 DB 클러스터로 승격하려면 RDS API 작업 PromoteReadReplicaDBCluster를 사용합니다.

복제본 클러스터를 승격한 후 다음과 같이 이벤트 로그를 확인하여 승격이 완료되었는지 확인할 수 있습니다.

Aurora 복제본이 승격되었는지 확인하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Events(이벤트) 를 선택합니다.
3. 이벤트 페이지에서 소스(Source) 목록의 해당 클러스터 이름을 찾습니다. 각 이벤트에는 소스, 유형, 시간 및 메시지가 있습니다. 계정의 AWS 리전에서 발생한 모든 이벤트를 볼 수 있습니다. 승격에 성공하면 다음 메시지가 생성됩니다.

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

승격이 완료된 후에는 소스 RDS for PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터 링크가 해제됩니다. 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본의 엔드포인트로 보낼 수 있습니다. Aurora 엔드포인트에 대한 자세한 내용은 Amazon Aurora 엔드포인트 연결 단원을 참조하세요. 이때 DB 인스턴스를 안전하게 삭제할 수 있습니다.

Aurora PostgreSQL에서 쿼리 성능 최적화

쿼리 성능을 최적화하면 더 적은 리소스를 사용하면서 데이터베이스가 더 빠르고 효율적으로 실행되어 사용자 경험이 향상되고 운영 비용이 절감되므로 이는 매우 중요합니다. Amazon Aurora PostgreSQL은 PostgreSQL 워크로드의 쿼리 성능을 최적화하는 데 도움이 되는 몇 가지 기능을 제공합니다.

주제

Aurora 최적화된 읽기로 Aurora PostgreSQL 쿼리 성능 개선

Aurora 최적화된 읽기로 Aurora PostgreSQL의 쿼리 처리 속도를 높일 수 있습니다. Aurora 최적화된 읽기를 사용하는 Aurora PostgreSQL DB 인스턴스는 DB 인스턴스의 메모리 용량을 초과하는 대규모 데이터 세트를 보유한 애플리케이션에서 쿼리 지연 시간이 최대 8배 개선되고 비용이 최대 30% 절감됩니다.

주제

PostgreSQL의 Aurora 최적화된 읽기 개요

Aurora 최적화된 읽기는 비휘발성 메모리 익스프레스(NVMe) 스토리지와 함께 Graviton 기반 R6gd 및 Intel 기반 R6id 인스턴스를 사용하는 DB 클러스터를 생성할 때 기본적으로 사용할 수 있습니다. 다음 PostgreSQL 버전에서 사용할 수 있습니다.

- 16.1 이상의 모든 버전
- 15.4 이상 버전
- 14.9 이상 버전

Aurora 최적화된 읽기는 계층형 캐시와 임시 객체라는 두 가지 기능을 지원합니다.

최적화된 읽기가 활성화된 계층형 캐시 - 계층형 캐시를 사용하면 DB 인스턴스 캐싱 용량을 인스턴스 메모리의 최대 5배까지 확장할 수 있습니다. 이렇게 하면 트랜잭션 측면에서 가장 최근의 일관성 있는 데이터를 포함하도록 캐시가 자동으로 유지되므로 애플리케이션은 외부 결과 집합 기반 캐싱 솔루션의 데이터 흐름을 관리하는 오버헤드에서 벗어날 수 있습니다. 이전에 Aurora 스토리지에서 데이터를 가져오던 쿼리에 대해 최대 8배 더 개선된 지연 시간을 제공합니다.

Aurora에서 기본 파라미터 그룹의 `shared_buffers` 값은 일반적으로 사용 가능한 메모리의 약 75%로 설정됩니다. 하지만 r6gd 및 r6id 인스턴스 유형의 경우 Aurora는 최적화된 읽기 캐시에 대한 메타데이터를 호스팅하기 위해 `shared_buffers` 공간을 4.5% 줄입니다.

최적화된 읽기가 활성화된 임시 객체 - 임시 객체를 사용하면 PostgreSQL에서 생성된 임시 파일을 로컬 NVMe 스토리지에 배치하여 보다 빠르게 쿼리를 처리할 수 있습니다. 이렇게 하면 네트워크를 통해 Elastic Block Storage(EBS)로 전송되는 트래픽이 줄어듭니다. DB 인스턴스에서 사용할 수 있는 메모리 용량을 초과하는 대량의 데이터를 정렬, 조인 또는 병합하는 고급 쿼리에 대해 최대 2배 더 개선된 지연 시간과 처리량을 제공합니다.

Aurora I/O 최적화 클러스터에서 최적화된 읽기는 계층형 캐시와 NVMe 스토리지의 임시 객체를 모두 사용합니다. 최적화된 읽기가 활성화된 계층형 캐시 기능을 통해 Aurora는 인스턴스 메모리의 2배를 임시 객체에 할당하고, 스토리지의 약 10%는 내부 작업으로, 나머지 스토리지는 계층형 캐시로 할당합니다. Aurora Standard 클러스터에서 최적화된 읽기는 임시 객체만 사용합니다.

Aurora I/O 최적화 클러스터를 사용하면 인스턴스 수준에서 `aurora_temp_space_size` 동적 파라미터를 사용하여 최적화된 읽기 지원 임시 객체에 할당된 공간의 크기를 조정할 수 있습니다. 이 크기 조정 기능은 다음 PostgreSQL 버전에서 사용할 수 있습니다.

- 16.8 이상의 모든 버전

- 15.12 이상의 15 버전
- 14.17 이상의 14 버전

이 파라미터를 사용하면 데이터베이스 엔진을 다시 시작할 필요 없이 인스턴스 메모리의 2배에서 최대 6배까지 용량을 조정할 수 있습니다. 임시 객체 공간을 확장하면 동시 워크로드에 관계없이 변경 사항이 즉시 적용됩니다. 그러나 공간을 줄이면 임시 객체에 새 크기 요청을 수용할 수 있는 충분한 미사용 공간이 있는 경우에만 조정이 완료됩니다. 최적화된 읽기 지원 임시 객체의 크기를 조정하면 계층형 캐시가 사용 가능한 공간을 사용하도록 자동으로 조정됩니다.

Engine	클러스터 스토리지 구성	최적화된 읽기가 활성화된 임시 객체	최적화된 읽기가 활성화된 계층형 캐시	지원되는 버전
Aurora PostgreSQL 호환 버전	표준	예	아니요	Aurora PostgreSQL 버전 16.1 이상의 모든 버전, 15.4 이상, 버전 14.9 이상
	I/O 최적화	예	예	

> **Note**
>
> NVMe 기반 DB 인스턴스 클래스에서 IO 최적화 클러스터와 Standard 클러스터 사이를 전환하면 데이터베이스 엔진이 즉시 재시작됩니다.

Aurora PostgreSQL에서는 임시 객체가 저장되는 테이블 공간을 구성하는 데 `temp_tablespaces` 파라미터를 사용합니다.

임시 객체가 구성되었는지 확인하려면 다음 명령을 사용합니다.

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
aurora_temp_tablespace
(1 row)
```

`aurora_temp_tablespace`는 NVMe 로컬 스토리지를 가리키는 Aurora에서 구성한 테이블스페이스입니다. 이 파라미터는 수정하거나 Amazon EBS 스토리지로 다시 전환할 수 없습니다.

최적화된 읽기 캐시가 켜져 있는지 확인하려면 다음 명령을 사용합니다.

```
postgres=> show shared_preload_libraries;
                 shared_preload_libraries
--------------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

Aurora 최적화된 읽기 사용

NVMe 기반 DB 인스턴스 중 하나를 사용하여 Aurora PostgreSQL DB 인스턴스를 프로비저닝하면 DB 인스턴스는 자동으로 Aurora 최적화된 읽기를 사용합니다.

Aurora Optimized Read를 켜려면 다음 중 하나를 수행합니다.

- NVMe 기반 DB 인스턴스 클래스 중 하나를 사용하여 Aurora PostgreSQL DB 클러스터를 생성합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.
- NVMe 기반 DB 인스턴스 클래스 중 하나를 사용하여 기존 Aurora PostgreSQL DB 클러스터를 수정합니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

Aurora 최적화된 읽기는 로컬 NVMe SSD 스토리지가 있는 DB 인스턴스 클래스 중 하나 이상이 지원되는 모든 AWS 리전에서 사용 가능합니다. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

최적화되지 않은 읽기 Aurora 인스턴스로 다시 전환하려면 Aurora 인스턴스의 DB 인스턴스 클래스를 데이터베이스 워크로드에 대해 NVMe 임시 스토리지가 없는 유사한 인스턴스 클래스로 수정하세요. 예를 들어, 현재 DB 인스턴스 클래스가 db.r6gd.4xlarge인 경우 db.r6g.4xlarge를 선택하여 다시 전환합니다. 자세한 내용은 Aurora DB 인스턴스 수정을 참조하세요.

Aurora 최적화된 읽기 사용 사례

최적화된 읽기가 활성화된 계층형 캐시

계층형 캐시가 있는 최적화된 읽기가 도움이 될 수 있는 몇 가지 사용 사례입니다.

- 엄격한 성능 SLA가 있는 결제 처리, 청구, 전자 상거래와 같은 인터넷 규모의 애플리케이션
- 지표/데이터 수집을 위해 수백 개의 포인트 쿼리를 실행하는 실시간 보고 대시보드

- 수백 개의 벡터 임베딩에서 정확한 이웃 또는 가장 가까운 이웃을 검색하기 위해 pgvector 확장을 갖춘 생성형 AI 애플리케이션

최적화된 읽기가 활성화된 임시 객체

다음은 임시 객체가 있는 최적화된 읽기가 도움이 될 수 있는 몇 가지 사용 사례입니다.

- CTE(Common Table Expression), 파생된 테이블, 그룹화 작업을 포함하는 분석 쿼리.
- 애플리케이션의 최적화되지 않은 쿼리를 처리하는 읽기 전용 복제본.
- 항상 적절한 인덱스를 사용할 수 없는 GROUP BY 및 ORDER BY 같은 복잡한 작업이 포함된 온디맨드 또는 동적 보고 쿼리.
- 정렬을 위한 `CREATE INDEX` 또는 `REINDEX` 작업입니다.
- 내부 임시 테이블을 사용하는 기타 워크로드.

Aurora 최적화된 읽기를 사용하는 DB 인스턴스 모니터링

최적화된 읽기가 활성화된 계층형 캐시를 사용하는 쿼리는 다음 예와 같이 EXPLAIN 명령을 사용하여 모니터링할 수 있습니다.

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
--------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> 계획 설명의 `Buffers` 섹션에 있는 `aurora_orcache_hit` 및 `aurora_storage_read` 필드는 최적화된 읽기가 켜져 있고 값이 0보다 큰 경우에만 표시됩니다. 읽기 필드는 `aurora_orcache_hit` 필드와 `aurora_storage_read` 필드의 합계입니다.

다음 CloudWatch 지표를 사용하여 Aurora 최적화된 읽기를 사용하는 DB 인스턴스를 모니터링할 수 있습니다.

- `AuroraOptimizedReadsCacheHitRatio`
- `FreeEphemeralStorage`
- `ReadIOPSEphemeralStorage`
- `ReadLatencyEphemeralStorage`
- `ReadThroughputEphemeralStorage`
- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

이러한 지표는 사용 가능한 인스턴스 스토어 스토리지, IOPS 및 처리량(throughput)에 대한 데이터를 제공합니다. 이러한 지표에 대한 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표](#) 섹션을 참조하세요.

`pg_proctab` 확장을 사용하여 NVMe 스토리지를 모니터링할 수도 있습니다.

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |     819350 |           0 |
 3847580 |      831020
```

```
       |       | rdsephemeralstorage |          23271 |          0 |        193098 |
    2620 |          114961 |           0 |        13845120 |     130770 |           0 |
   215010 |       133410
 (2 rows)
```

Aurora 최적화된 읽기 모범 사례

Aurora 최적화된 읽기에 대한 다음 모범 사례를 따릅니다.

- CloudWatch 지표 `FreeEphemeralStorage`를 사용하여 인스턴스 스토어에서 사용 가능한 스토리지 공간을 모니터링합니다. DB 인스턴스의 워크로드로 인해 인스턴스 스토어가 한도에 도달하면 임시 객체를 많이 사용하는 동시성 및 쿼리를 조정하거나 더 큰 DB 인스턴스 클래스를 사용하도록 수정합니다.
- 최적화된 읽기 캐시 적중률에 대한 CloudWatch 지표를 모니터링합니다. VACUUM과 같은 작업은 많은 수의 블록을 매우 빠르게 수정합니다. 이로 인해 적중률이 일시적으로 떨어질 수 있습니다. `pg_prewarm` 확장을 사용하여 데이터를 버퍼 캐시에 로드하면 Aurora가 이러한 블록 중 일부를 최적화된 읽기 캐시에 선제적으로 기록할 수 있습니다.
- 클러스터 캐시 관리(CCM)를 사용하여 장애 조치 대상으로 사용될 Tier-0 리더의 버퍼 캐시 및 계층형 캐시를 워밍업할 수 있습니다. CCM이 활성화되면 버퍼 캐시를 정기적으로 스캔하여 제거 가능한 페이지를 계층형 캐시에 기록합니다. CCM에 대한 자세한 내용은 장애 조치 후 Aurora PostgreSQL용 클러스터 캐시 관리를 통한 신속한 복구 섹션을 참조하세요.

Aurora PostgreSQL에서 상관관계가 있는 하위 쿼리 최적화

상관관계가 있는 하위 쿼리는 외부 쿼리의 표 열을 참조합니다. 외부 쿼리에서 반환된 모든 행에 대해 한 번씩 평가됩니다. 다음 예에서는 하위 쿼리가 표 ot의 열을 참조합니다. 이 표는 하위 쿼리의 FROM 절에는 포함되지 않지만, 외부 쿼리의 FROM 절에서는 참조됩니다. 표 ot의 행이 100만 개이면 하위 쿼리를 100만 번 평가해야 합니다.

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

> **Note**
>
> - 하위 쿼리 변환 및 하위 쿼리 캐시는 버전 16.8부터 Aurora PostgreSQL에서 사용할 수 있으며 Babelfish for Aurora PostgreSQL은 4.2.0부터 이러한 기능을 지원합니다.
> - Babelfish for Aurora PostgreSQL 버전 4.6.0 및 5.2.0부터 다음 파라미터가 이러한 기능을 제어합니다.

- babelfishpg_tsql.apg_enable_correlated_scalar_transform
- babelfishpg_tsql.apg_enable_subquery_cache

기본적으로 두 파라미터 모두 켜져 있습니다.

하위 쿼리 변환을 사용하여 Aurora PostgreSQL 쿼리 성능 개선

Aurora PostgreSQL은 상관관계가 있는 하위 쿼리를 상응하는 외부 조인으로 변환하여 해당 하위 쿼리를 가속화할 수 있습니다. 이 최적화는 다음과 같은 두 유형의 상관관계가 있는 하위 쿼리에 적용됩니다.

- 단일 집계 값을 반환하고 SELECT 목록에 나타나는 하위 쿼리입니다.

```
SELECT ot.a, ot.b, (SELECT AVG(it.b) FROM it WHERE it.a = ot.a) FROM ot;
```

- 단일 집계 값을 반환하고 WHERE 절에 나타나는 하위 쿼리입니다.

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

하위 쿼리에서 변환 활성화

상관관계가 있는 하위 쿼리를 동등한 외부 조인으로 변환할 수 있도록 하려면 `apg_enable_correlated_scalar_transform` 파라미터를 ON으로 설정합니다. 이 파라미터의 기본값은 `OFF`입니다.

클러스터 또는 인스턴스 파라미터 그룹을 수정하여 파라미터를 설정할 수 있습니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹를 참조하세요.

또는 다음 명령을 사용하여 현재 세션에 대한 설정만 구성할 수도 있습니다.

```
SET apg_enable_correlated_scalar_transform TO ON;
```

변환 확인

EXPLAIN 명령을 사용하여 상관관계가 있는 하위 쿼리가 쿼리 계획에서 외부 조인으로 변환되었는지 확인합니다.

변환이 활성화되면 상관관계가 있는 관련 하위 쿼리 부분이 외부 조인으로 변환됩니다. 예시:

```
postgres=> CREATE TABLE ot (a INT, b INT);
CREATE TABLE
postgres=> CREATE TABLE it (a INT, b INT);
CREATE TABLE

postgres=> SET apg_enable_correlated_scalar_transform TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);

                         QUERY PLAN
--------------------------------------------------------------
 Hash Join
   Hash Cond: (ot.a = apg_scalar_subquery.scalar_output)
   Join Filter: ((ot.b)::numeric < apg_scalar_subquery.avg)
   ->  Seq Scan on ot
   ->  Hash
         ->  Subquery Scan on apg_scalar_subquery
               ->  HashAggregate
                     Group Key: it.a
                     ->  Seq Scan on it
```

GUC 파라미터를 OFF로 설정해도 동일한 쿼리는 변환되지 않습니다. 이 계획에는 외부 조인이 없고, 대신 하위 계획이 있습니다.

```
postgres=> SET apg_enable_correlated_scalar_transform TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
                QUERY PLAN
----------------------------------------
 Seq Scan on ot
   Filter: ((b)::numeric < (SubPlan 1))
   SubPlan 1
     ->  Aggregate
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

제한 사항

- 하위 쿼리는 SELECT 목록 또는 where 절의 조건 중 하나에 있어야 합니다. 그렇지 않으면 변환되지 않습니다.

- 하위 쿼리는 집계 함수를 반환해야 합니다. 사용자 정의 집계 함수는 변환에 지원되지 않습니다.
- 반환 표현식이 단순 집계 함수가 아닌 하위 쿼리는 변환되지 않습니다.
- 하위 쿼리 WHERE 절의 상관관계가 있는 조건은 단순한 열 참조여야 합니다. 그렇지 않으면 변환되지 않습니다.
- 절이 일반 등식 조건자여야 하는 하위 쿼리의 상관관계가 있는 조건입니다.
- 하위 쿼리에는 HAVING 또는 GROUP BY 절을 포함할 수 없습니다.
- 하위 쿼리의 where 절에는 AND와 결합된 하나 이상의 조건자가 포함될 수 있습니다.

> **Note**
> 변환의 성능 영향은 스키마, 데이터 및 워크로드에 따라 달라집니다. 변환과 관련된 하위 쿼리 실행은 외부 쿼리에서 생성된 행 수가 증가함에 따라 성능을 크게 개선할 수 있습니다. 프로덕션 환경에서 활성화하기 전에 실제 스키마, 데이터 및 워크로드를 사용하여 비프로덕션 환경에서 이 기능을 테스트하는 것이 좋습니다.

하위 쿼리 캐시를 사용하여 Aurora PostgreSQL 쿼리 성능 향상

Aurora PostgreSQL은 하위 쿼리 캐시를 지원하여 상관관계가 있는 하위 쿼리의 결과를 저장합니다. 이 기능은 하위 쿼리 결과가 이미 캐시에 있는 경우 상관관계가 있는 반복 하위 쿼리 실행을 건너뜁니다.

하위 쿼리 캐시 이해

PostgreSQL의 Memoize 노드는 하위 쿼리 캐시의 핵심 부분입니다. Memoize 노드는 로컬 캐시에 해시 표를 유지하여 입력 파라미터 값을 쿼리 결과 행으로 매핑합니다. 해시 표의 메모리 제한은 work_mem과 hash_mem_multiplier의 곱입니다. 자세한 내용은 리소스 소비를 참조하세요.

쿼리 실행 중에 하위 쿼리 캐시는 캐시 적중률(CHR)을 사용하여 캐시가 쿼리 성능을 개선하는지를 추정하고 쿼리 런타임 시 캐시를 계속 사용할지를 결정합니다. CHR은 총 요청 수에 대한 캐시 적중 횟수의 비율입니다. 예를 들어, 상관관계가 있는 하위 쿼리를 100번 실행해야 하고 해당 실행 결과 중 70개를 캐시에서 검색할 수 있는 경우 CHR은 0.7입니다.

apg_subquery_cache_check_interval 캐시 누락 횟수가 발생할 때마다 CHR이 apg_subquery_cache_hit_rate_threshold보다 큰지 확인하여 하위 쿼리 캐시의 이점을 평가합니다. 이보다 작으면 캐시가 메모리에서 삭제되고 쿼리 실행은 캐시되지 않은 원래 하위 쿼리 재실행 상태로 돌아갑니다.

하위 쿼리 캐시 동작을 제어하는 파라미터

다음 표에는 하위 쿼리 캐시 동작을 제어하는 파라미터가 나열되어 있습니다.

파라미터	설명	Default	허용
apg_enable_subquery_cache	상관관계가 있는 스칼라 하위 쿼리에 캐시를 사용할 수 있도록 지원합니다.	OFF	ON, OFF
apg_subquery_cache_check_interval	하위 쿼리 캐시 적중률을 평가할 빈도(캐시 누락 횟수)를 설정합니다.	500	0–2147483647
apg_subquery_cache_hit_rate_threshold	하위 쿼리 캐시 적중률의 임계값을 설정합니다.	0.3	0.0~1.0

> **Note**
>
> - `apg_subquery_cache_check_interval` 값이 클수록 CHR 기반 캐시 이점 추정의 정확도는 향상될 수 있지만, 캐시 표에 `apg_subquery_cache_check_interval` 행이 있을 때까지 CHR이 평가되지 않으므로 캐시 오버헤드가 증가합니다.
> - `apg_subquery_cache_hit_rate_threshold` 값이 클수록 하위 쿼리 캐시를 포기하고 캐시되지 않은 원래 하위 쿼리 재실행으로 돌아가는 경향이 있습니다.

클러스터 또는 인스턴스 파라미터 그룹을 수정하여 파라미터를 설정할 수 있습니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹를 참조하세요.

또는 다음 명령을 사용하여 현재 세션에 대한 설정만 구성할 수도 있습니다.

```
SET apg_enable_subquery_cache TO ON;
```

Aurora PostgreSQL에서 하위 쿼리 캐시 켜기

하위 쿼리 캐시가 활성화되면 Aurora PostgreSQL은 캐시를 적용하여 하위 쿼리 결과를 저장합니다. 그러면 쿼리 계획의 SubPlan 아래에 Memoize 노드가 생깁니다.

예를 들어, 다음 명령 시퀀스는 하위 쿼리 캐시 없이 상관관계가 있는 단순 하위 쿼리의 예상 쿼리 실행 계획을 보여줍니다.

```
postgres=> SET apg_enable_subquery_cache TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
------------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Seq Scan on it
           Filter: (a = ot.a)
```

apg_enable_subquery_cache를 활성화한 후 쿼리 계획의 SubPlan 노드 아래에 Memoize 노드가 포함되며, 이는 하위 쿼리가 캐시를 사용할 계획임을 나타냅니다.

```
postgres=> SET apg_enable_subquery_cache TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
------------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Memoize
           Cache Key: ot.a
           Cache Mode: binary
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

실제 쿼리 실행 계획에는 캐시 적중 및 캐시 누락을 포함하여 하위 쿼리 캐시에 대한 자세한 정보가 포함되어 있습니다. 다음 출력은 표에 일부 값을 삽입한 후의 위 예제 쿼리의 실제 쿼리 실행 계획을 보여줍니다.

```
postgres=> EXPLAIN (COSTS FALSE, TIMING FALSE, ANALYZE TRUE) SELECT ot.a, ot.b FROM ot
 WHERE ot.b < (SELECT it.b FROM it WHERE it.a = ot.a);
            QUERY PLAN
-----------------------------------------------------------------------------
 Seq Scan on ot (actual rows=2 loops=1)
   Filter: (b < (SubPlan 1))
   Rows Removed by Filter: 8
   SubPlan 1
     ->  Memoize (actual rows=0 loops=10)
           Cache Key: ot.a
           Cache Mode: binary
           Hits: 4  Misses: 6  Evictions: 0  Overflows: 0  Memory Usage: 1kB
           ->  Seq Scan on it (actual rows=0 loops=6)
                 Filter: (a = ot.a)
                 Rows Removed by Filter: 4
```

총 캐시 적중 수는 4이고 총 캐시 누락 수는 6입니다. 총 적중 및 누락 수가 Memoize 노드의 루프 수보다 적으면 CHR 평가를 통과하지 못했고 캐시가 어느 시점에서 정리되고 폐기되었음을 의미합니다. 이에 따라 하위 쿼리 실행이 캐시되지 않은 원래 재실행으로 되돌아갔습니다.

제한 사항

하위 쿼리 캐시는 특정 패턴의 상관관계가 있는 하위 쿼리를 지원하지 않습니다. 이러한 유형의 쿼리는 하위 쿼리 캐시가 설정되어 있더라도 캐시 없이 실행됩니다.

- IN/EXISTS/ANY/ALL 상관관계가 있는 하위 쿼리
- 비결정적 함수를 포함하는 상관관계가 있는 하위 쿼리.
- 해싱 또는 등식 연산을 지원하지 않는 데이터 유형이 있는 외부 테이블 열을 참조하는 상관관계가 있는 하위 쿼리입니다.

적응형 조인을 사용하여 쿼리 성능 개선

개요

적응형 조인은 쿼리 성능을 개선하는 데 도움이 되는 Aurora PostgreSQL 17.4의 미리 보기 기능입니다. 이 기능은 기본적으로 비활성화되어 있지만 글로벌 사용자 구성(GUC) 파라미터를 사용하여 활성

화할 수 있습니다. 이 기능은 미리 보기 기능이므로 기본 파라미터 값이 변경될 수 있습니다. 활성화된 경우 적응형 조인은 런타임에 중첩 루프 조인에서 해시 조인으로 동적으로 전환하여 쿼리 성능을 최적화하는 데 도움이 됩니다. 이 전환은 부정확한 카디널리티 추정치로 인해 PostgreSQL 최적화 프로그램이 중첩 루프 조인을 잘못 선택한 경우 발생합니다.

적응형 조인 구성

다음 세 가지 GUC 파라미터를 사용하여 적응형 조인을 제어할 수 있습니다.

적응형 조인 구성 파라미터

GUC 파라미터	설명	기본 및 구성 옵션
apg_adaptive_join_crossover_multiplier	이 승수는 행 교차점과 함께 작동하여 중첩 루프에서 해시 조인으로 전환할 시기를 결정합니다. 행 교차점은 SQL 최적화 프로그램이 중첩 루프 및 해시 조인 작업의 비용이 동일하다고 추정하는 지점입니다. 승수 값이 높을수록 적응형 조인이 해시 조인으로 전환될 가능성이 줄어듭니다.	적응형 조인이 활성화되었는지 여부를 제어합니다. • 기본값: -1(비활성화됨) • 유효한 범위: -1~DBL_MAX • 활성화하려면 1보다 크거나 같은 값으로 설정합니다.
apg_adaptive_join_cost_threshold	이 파라미터는 최소 쿼리 비용 임계값을 설정합니다. 적응형 조인은 이 임계값 미만의 쿼리에 대해 자동으로 비활성화됩니다. 이렇게 하면 적응형 조인을 계획하는 데 드는 비용이 중첩 루프에서 해시 조인으로 전환하는 이익을 초과할 수 있는 단순 쿼리의 성능 오버헤드를 방지할 수 있습니다.	쿼리에 대한 최소 비용 임계값을 설정합니다. • 기본 값: 100 • 유효한 범위: 0~DBL_MAX
apg_enable_parameterized_adaptive_join	이 파라미터는 활성화된 경우 적응형 조인 기능을 파라미터화된 중첩 루프 조인으로 확장합니다. 기본적으로 적응형 조인은 파라미터화되지 않은 중첩 루프 조인에서만 작동합니다. 해시 조인으로 전환하면 이점을 얻을 가능성이 높기 때문입니다. 파라미터화된	중첩 루프 조인의 적응형 조인 동작을 제어합니다. • 기본값: false • 유효값: true/false

GUC 파라미터	설명	기본 및 구성 옵션
	중첩 루프 조인은 일반적으로 성능이 더 좋으므로 해시 조인으로 전환할 필요성이 덜합니다.	• false인 경우: 파라미터화되지 않은 중첩 루프 조인에서만 작동합니다. • true인 경우: 파라미터화된 중첩 루프 조인과 파라미터화되지 않은 중첩 루프 조인 모두에서 작동합니다. **Note** 먼저 `apg_adaptive_join_crossover_multiplier` 를 활성화해야 합니다.

Aurora PostgreSQL에서 로깅되지 않은 테이블 작업

Amazon Aurora PostgreSQL은 라이터 인스턴스가 실패하거나 또는 장애 조치된 후에도 충돌이 방지되고 데이터 무결성을 유지하는 로깅되지 않은 테이블을 지원합니다. 표준 PostgreSQL에서는 로깅되지 않은 테이블이 쓰기 작업 중에 Write-Ahead Log(WAL)를 우회하여 쓰기 속도가 빨라집니다. 그러나 로깅되지 않은 테이블은 충돌이 방지되지 않으며 시스템 장애 또는 비정상 종료 후 데이터가 손실될 수 있으므로 내구성이 낮습니다. 이러한 로깅되지 않은 테이블은 충돌 또는 비정상 종료 후 자동으로 잘립니다. 콘텐츠와 인덱스도 대기 서버에 복제되지 않습니다.

반면 Aurora PostgreSQL은 분산 스토리지 아키텍처로 인해 로깅되지 않은 테이블을 다르게 처리합니다. 이는 Aurora 스토리지 시스템의 내구성이 기존 PostgreSQL WAL에 의존하지 않기 때문입니다. 그러나 표준 PostgreSQL에서 로깅되지 않은 테이블과 일반적으로 연관되는 성능 이점은 Aurora에서 그다지 중요하지 않을 수 있습니다. 이는 Aurora의 분산 스토리지 아키텍처로 인해 표준 PostgreSQL에 사용되는 로컬 스토리지에 비해 추가 오버헤드가 발생할 수 있기 때문입니다.

Aurora PostgreSQL에서 로깅되지 않은 테이블을 사용하는 경우 다음을 고려하세요.

- 로깅되지 않은 테이블은 Aurora DB 클러스터의 라이터 노드에서만 액세스할 수 있습니다.
- 리더 노드는 라이터 상태로 승격된 경우에만 로깅되지 않은 테이블에 액세스할 수 있습니다.
- 리더 노드에서 로깅되지 않은 테이블에 액세스하려고 하면 다음 오류가 발생합니다.

  ```
  cannot access temporary or unlogged relations during recovery.
  ```

로깅되지 않은 테이블 만들기

Aurora PostgreSQL에서 로깅되지 않은 테이블을 만들려면 CREATE TABLE 문에 UNLOGGED 키워드를 추가합니다.

```
CREATE UNLOGGED TABLE staging_sales_data (
    transaction_id bigint,
    customer_id bigint,
    product_id bigint,
    transaction_date date,
    amount NUMERIC
);
```

로깅되지 않은 테이블을 로깅된 테이블로 변환

로깅되지 않은 테이블을 다시 로깅된 테이블로 변환해야 하는 경우 다음 명령을 사용할 수 있습니다.

```
ALTER TABLE table_name SET LOGGED;
```

이 작업은 전체 테이블을 다시 작성하고 작업이 완료될 때까지 테이블을 배타적으로 잠급니다. 대형 테이블의 경우 이로 인해 상당한 가동 중지 시간이 발생할 수 있습니다.

로깅되지 않은 테이블 및 논리적 복제

논리적 복제는 WAL을 사용하여 변경 사항을 캡처하고 전송하기 때문에 로깅되지 않은 테이블은 일반적으로 논리적 복제에 포함되지 않습니다. 기본적으로 로깅되지 않은 테이블

에 대한 변경 사항은 WAL 로깅되지 않고 복제 스트림에서 제외되므로 논리적 복제가 필요한 사용 사례에 적합하지 않습니다. 그러나 Aurora PostgreSQL은 이 동작을 제어할 수 있는 `rds.logically_replicate_unlogged_tables`라는 파라미터를 제공합니다.

- `rds.logically_replicate_unlogged_tables`가 0(꺼짐)으로 설정되면 로깅되지 않은 테이블이 논리적 복제에서 제외됩니다.
- `rds.logically_replicate_unlogged_tables`가 1(켜짐)로 설정되면 로깅되지 않은 테이블이 논리적 복제에 포함됩니다.

> **Note**
>
> Aurora PostgreSQL에서 `rds.logically_replicate_unlogged_tables` 파라미터는 기본적으로 버전 14 이하에서는 1(켜짐), 버전 15 이상에서는 0(꺼짐)으로 설정됩니다.

Amazon Aurora PostgreSQL에서 PostgreSQL 자동 정리 사용

자동 정리 기능을 사용하여 PostgreSQL DB 인스턴스의 상태를 유지 관리하는 것이 좋습니다. Autovacuum은 VACUUM 및 ANALYZE 명령의 시작을 자동화합니다. Autovacuum은 삽입되고 업데이트되거나 삭제된 튜플 수가 많은 테이블이 있는지 확인합니다. 확인이 끝나면 Autovacuum은 PostgreSQL 데이터베이스에서 폐기된 데이터 또는 튜플을 제거하여 스토리지를 회수합니다.

기본적으로 자동 정리는 기본 PostgreSQL DB 파라미터 그룹을 사용하여 만든 Aurora PostgreSQL DB 인스턴스에서 켜져 있습니다. 자동 정리 기능과 관련된 다른 구성 파라미터도 기본적으로 설정됩니다. 이러한 기본값은 다소 일반적이기 때문에 특정 워크로드에 대해 자동 정리 기능과 관련된 일부 파라미터를 조정하면 도움이 될 수 있습니다.

다음에서 자동 정리에 대한 자세한 정보와 Aurora PostgreSQL DB 인스턴스에 대한 파라미터를 조정하는 방법을 확인할 수 있습니다.

주제

- 수동 vacuum freeze 수행
- Autovacuum이 실행 중인 경우 테이블 인덱스 다시 지정
- 대용량 인덱스를 사용하여 autovacuum 관리
- Autovacuum에 영향을 주는 기타 파라미터
- 테이블 수준 Autovacuum 파라미터 설정
- autovacuum 및 vacuum 활동 로그
- 잘못된 데이터베이스를 사용한 autovacuum 동작 이해
- Aurora PostgreSQL에서 공격적인 vacuum 블로커 식별 및 해결

Autovacuum에 메모리 할당

autovacuum 성능에 영향을 미치는 가장 중요한 파라미터 중 하나는 `autovacuum_work_mem` 파라미터입니다. Aurora PostgreSQL 버전 14 이하에서는 `autovacuum_work_mem` 파라미터가 -1로 설정되어 `maintenance_work_mem`의 설정이 대신 사용됨을 나타냅니다. 다른 모든 버전의 경우 `autovacuum_work_mem`는 GREATEST({DBInstanceClassMemory/32768}, 65536)에 의해 결정됩니다.

수동 vacuum 작업은 항상 기본 `maintenance_work_mem` 설정인 GREATEST({DBInstanceClassMemory/63963136*1024}, 65536)을 사용하며, 보다 대상화된 수동 `VACUUM` 작업을 위해 `SET` 명령을 사용하여 세션 수준에서 조정할 수도 있습니다.

`autovacuum_work_mem`는 인덱스 vacuum용 데드 튜플(`pg_stat_all_tables.n_dead_tup`)의 ID를 저장하기 위한 autovacuum 메모리를 결정합니다.

`autovacuum_work_mem` 파라미터 값을 결정하기 위해 계산할 때는 다음 사항에 유의하세요.

- 파라미터를 너무 낮게 설정하면 vacuum 프로세스가 테이블을 여러 번 스캔해야 작업이 완료될 수 있습니다. 이러한 다중 스캔은 성능에 부정적인 영향을 줄 수 있습니다. 더 큰 인스턴스의 경우 `maintenance_work_mem` 또는 `autovacuum_work_mem`을 최소 1GB로 설정하면 데드 튜플 수가 많은 테이블을 vacuum하는 성능이 향상될 수 있습니다. 그러나 PostgreSQL 버전 16 이하에서는 vacuum 메모리 사용량이 1GB로 제한되므로 한 번의 전달로 약 1억 7,900만 개의 데드 튜플을 처리하기에 충분합니다. 테이블에 이보다 더 많은 데드 튜플이 있는 경우 vacuum은 테이블의 인덱스를 여러 번 전달해야 하므로 필요한 시간이 크게 늘어납니다. PostgreSQL 버전 17부터는 1GB의 제한이 없으며, autovacuum은 radix 트리를 사용하여 1억 7,900만 개 이상의 튜플을 처리할 수 있습니다.

  튜플 식별자의 크기는 6바이트입니다. 테이블의 인덱스를 vacuum하는 데 필요한 메모리를 추정하려면 `pg_stat_all_tables.n_dead_tup`를 쿼리하여 데드 튜플 수를 찾은 다음 이 수에 6을 곱

하여 인덱스를 단일 전달로 vacuum하는 데 필요한 메모리를 결정합니다. 다음 쿼리를 사용할 수 있습니다.

```
SELECT
    relname AS table_name,
    n_dead_tup,
    pg_size_pretty(n_dead_tup * 6) AS estimated_memory
FROM
    pg_stat_all_tables
WHERE
    relname = 'name_of_the_table';
```

- `autovacuum_work_mem` 파라미터는 `autovacuum_max_workers` 파라미터와 함께 작동합니다. `autovacuum_max_workers` 중 각각의 작업자는 할당된 메모리를 사용할 수 있습니다. 작은 테이블이 많이 있는 경우에는 `autovacuum_max_workers`를 더 많이 할당하고 `autovacuum_work_mem`을 더 적게 할당합니다. 큰 테이블이 많이 있는 경우(100GB 이상)에는 메모리를 더 많이 할당하고 작업자 프로세스를 더 적게 할당합니다. 가장 큰 테이블에서 성공적으로 작업을 수행하려면 충분한 메모리를 할당해 두어야 합니다. 따라서 작업자 프로세스와 메모리를 합한 양이 할당하려는 전체 메모리 양과 같아야 합니다.

트랜잭션 ID 랩어라운드의 가능성 감소

경우에 따라 Autovacuum과 관련된 파라미터 그룹 설정이 트랜잭션 ID 랩어라운드를 방지하기에 충분히 공격적이지 않을 수 있습니다. 이를 해결하기 위해 Aurora PostgreSQL은 자동 정리 파라미터 값을 자동으로 조정하는 메커니즘을 제공합니다. 적응형 자동 정리는 Aurora PostgreSQL의 기능입니다. TransactionID wraparound에 대한 자세한 설명은 PostgreSQL 설명서에 나와 있습니다.

동적 파라미터 `rds.adaptive_autovacuum`이 켜짐으로 설정된 Aurora PostgreSQL 인스턴스의 경우 적응형 자동 정리가 기본적으로 켜져 있습니다. 이 설정을 항상 활성화해 놓는 것이 좋습니다. 그러나 적응형 Autovacuum 파라미터 튜닝을 끄려면 `rds.adaptive_autovacuum` 파라미터를 0 또는 OFF로 설정합니다.

Aurora Amazon RDS가 자동 정리 파라미터를 조정하더라도 트랜잭션 ID 랩어라운드는 계속 가능합니다. 트랜잭션 ID 랩어라운드에 대한 Amazon CloudWatch 경보를 구현하는 것이 좋습니다. 자세한 내용은 AWS 데이터베이스 블로그의 RDS for PostgreSQL에서 트랜잭션 ID 랩어라운드에 대한 조기 경고 시스템 구축 게시물을 참조하세요.

적응형 자동 정리 파라미터 튜닝을 사용 설정한 경우 Amazon RDS는 CloudWatch 지표 `MaximumUsedTransactionIDs`가 `autovacuum_freeze_max_age` 파라미터 값 또는 500,000,000 중 큰 값에 도달하면 자동 정리 파라미터를 조정하기 시작합니다.

테이블이 계속 트랜잭션 ID 랩어라운드 방향으로 향하면 Amazon RDS는 Autovacuum의 파라미터를 계속 조정합니다. 이러한 각각의 조정은 랩어라운드를 피하기 위해 Autovacuum에 더 많은 리소스를 할애합니다. Amazon RDS는 다음 자동 정리 관련 파라미터를 업데이트합니다.

- [autovacuum_vacuum_cost_delay]
- [autovacuum_vacuum_cost_limit]
- [`autovacuum_work_mem`]
- [autovacuum_naptime]

RDS는 새 값이 Autovacuum을 보다 공격적으로 만드는 경우에만 이러한 파라미터를 수정합니다. 파라미터는 DB 인스턴스의 메모리에서 수정됩니다. 파라미터 그룹의 값은 변경되지 않습니다. 현재 인 메모리 설정을 보려면 PostgreSQL [SHOW] SQL 명령을 사용하십시오.

Amazon RDS가 이러한 Autovacuum 파라미터를 수정하면 영향받은 DB 인스턴스에 대한 이벤트를 생성합니다. 이 이벤트는 AWS Management Console 및 Amazon RDS API에서 볼 수 있습니다. `MaximumUsedTransactionIDs` CloudWatch 지표가 임계값 미만의 값을 반환하면 Amazon RDS는 메모리의 자동 정리 관련 파라미터를 파라미터 그룹에 지정된 값으로 다시 설정합니다. 그런 다음 이 변경에 해당하는 다른 이벤트를 생성합니다.

데이터베이스의 테이블을 vacuum해야 하는지 여부를 결정

다음 쿼리를 사용하여 데이터베이스의 고정되지 않은 트랜잭션 수를 표시할 수 있습니다. 데이터베이스 `datfrozenxid` 행의 `pg_database` 열은 해당 데이터베이스에 나타나는 정상 트랜잭션 ID의 하한 값입니다. 이 열은 데이터베이스 내 테이블 단위 `relfrozenxid` 값 중 최소값입니다.

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

예를 들어 앞의 쿼리를 실행하면 다음과 같은 결과가 나올 수 있습니다.

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
```

```
template1   | 1721757888
rdsadmin    | 1694008527
postgres    | 1693881061
(5 rows)
```

데이터베이스의 수명이 20억 트랜잭션 ID에 도달하면 트랜잭션 ID(XID) 랩어라운드가 발생하고 데이터베이스는 읽기 전용이 됩니다. 이 쿼리를 사용하면 지표를 생성하여 하루에 몇 번 실행되도록 할 수 있습니다. 기본적으로 autovacuum은 트랜잭션 수명을 200,000,000(`autovacuum_freeze_max_age`) 미만으로 유지하도록 설정됩니다.

샘플 모니터링 전략은 다음과 같습니다.

- `autovacuum_freeze_max_age` 값을 2억 개 트랜잭션으로 설정하십시오.
- 테이블이 5억 개의 고정되지 않은 트랜잭션에 도달하면 낮은 심각도 경보가 트리거됩니다. 이 값은 타당한 값이지만 autovacuum이 계속 수행되고 있지 않음을 나타낼 수 있습니다.
- 테이블 수명이 10억이 되면 조치를 취해야 할 경보로 처리되어야 합니다. 성능상의 이유로 수명을 `autovacuum_freeze_max_age`에 더 가깝게 유지하려는 경우가 대부분입니다. 다음 권장 사항을 사용하여 조사하는 것이 좋습니다.
- 테이블이 15억 개의 vacuum되지 않은 트랜잭션에 도달하면 높은 심각도 경보가 트리거됩니다. 데이터베이스가 트랜잭션 ID를 사용하는 속도에 따라 이 경보는 시스템에서 autovacuum을 실행할 시간이 부족함을 나타낼 수 있습니다. 이 경우 즉시 이를 해결하는 것이 좋습니다.

테이블이 지속적으로 이 임계값을 위반하면 autovacuum 파라미터를 추가로 수정합니다. 기본적으로 수동 VACUUM을 사용하면(비용에 따른 지연이 비활성화됨)은 기본 autovacuum을 사용할 때보다 더 적극적이지만 시스템 전체에 더 많이 침입할 수 있는 상태이기도 합니다.

다음과 같이 하는 것이 좋습니다.

- 모니터링 메커니즘을 숙지하고 활성화하여 가장 오래된 트랜잭션의 수명을 확인합니다.

  트랜잭션 ID 랩어라운드에 대해 경고하는 프로세스 생성에 대한 자세한 내용은 AWS 데이터베이스 블로그 게시물 Amazon RDS for PostgreSQL의 트랜잭션 ID 랩어라운드용 조기 경고 시스템 구현을 참조하세요.
- 더 많이 사용되는 테이블의 경우 autovacuum을 사용하는 것 이외에 유지 관리 기간 동안 수동 vacuum freeze를 정기적으로 수행합니다. 수동 vacuum freeze 수행에 대한 자세한 내용은 수동 vacuum freeze 수행 단원을 참조하십시오.

현재 Autovacuum을 수행할 수 있는 테이블 결정

vacuum을 수행해야 하는 테이블이 하나이거나 두 개인 경우가 많습니다. relfrozenxid 값이 autovacuum_freeze_max_age의 트랜잭션 수보다 큰 테이블은 항상 Autovacuum의 대상이 됩니다. 그렇지 않은 경우 VACUUM이 "vacuum 임계값"을 초과하여 튜플 수가 더 이상 사용되지 않는 경우 테이블이 vacuum됩니다.

autovacuum 임계값은 다음과 같이 정의되어 있습니다.

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

여기서 vacuum base threshold는 autovacuum_vacuum_threshold이고, vacuum scale factor는 autovacuum_vacuum_scale_factor이며, number of tuples는 pg_class.reltuples입니다.

데이터베이스에 연결되어 있는 상태에서 다음 쿼리를 실행하여 autovacuum이 vacuum 가능한 대상으로 분류하는 테이블 목록을 확인합니다.

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
```

```
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

현재 Autovacuum이 실행 중인지 여부 및 실행 기간 확인

테이블을 수동으로 vacuum해야 하는 경우 autovacuum이 현재 실행 중인지 확인합니다. 실행 중이면 더 효율적으로 실행되도록 파라미터를 수정하거나 VACUUM을 수동으로 실행할 수 있도록 일시적으로 autovacuum을 종료해야 합니다.

다음 쿼리를 사용하여 autovacuum이 실행 중인지 여부와 얼마 동안 실행되고 있는지, 다른 세션에 대해 대기하고 있는지 확인합니다.

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

쿼리를 실행하면 다음과 유사한 출력이 표시됩니다.

```

 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+--------------------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
 mydb    | rdsadmin |   618 | active |            | 00:00:00                |
 SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query+
         |          |       |        |            |                         | FROM
 pg_stat_activity
           +
```

```
        |        |      |      |          |                    | WHERE
query like '%VACUUM%'
          +
        |        |      |      |          |                    | ORDER BY
xact_start;
        +
```

몇 가지 문제로 인해 autovacuum 세션이 오래(며칠간) 실행될 수 있습니다. 이 문제는 대부분 `maintenance_work_mem` 파라미터 값이 테이블 크기 또는 업데이트 속도에 대해 너무 낮게 설정된 경우입니다.

다음 공식을 사용하여 `maintenance_work_mem` 파라미터 값을 설정하는 것이 좋습니다.

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

짧은 기간 동안 실행되는 autovacuum 세션에서도 문제를 표시할 수 있습니다.

- 워크로드에 `autovacuum_max_workers`가 충분하지 않다고 표시될 수 있습니다. 이 경우 작업자 수를 명시해야 합니다.
- 인덱스 손상(autovacuum에 충돌이 발생하여 동일한 관계에서 다시 시작되지만 진행되지 않음)이 있다고 표시될 수 있습니다. 이 경우에는 `vacuum freeze verbose table` 매뉴얼을 실행하여 정확한 원인을 확인합니다.

수동 vacuum freeze 수행

vacuum 프로세스가 실행되고 있는 테이블에서 수동 vacuum을 수행하려는 경우가 있습니다. 이 작업은 수명이 20억 개 트랜잭션에 도달하거나 모니터링 중인 임계값을 초과한 테이블을 파악해 둔 경우 유용합니다.

다음 단계는 지침으로 이 프로세스를 여러 가지로 변형할 수 있습니다. 예를 들어 테스트 중에 `maintenance_work_mem` 파라미터 값이 너무 작게 설정되었고 테이블에 작업을 즉시 수행해야 한다고 가정해보겠습니다. 그러나 지금은 인스턴스를 반송하고 싶지 않을 수도 있습니다. 이전 세션의 쿼리를 사용하여 어떤 테이블이 문제이고 오랜 기간 동안 실행 중인 autovacuum 세션이 있는지 확인합니다. `maintenance_work_mem` 파라미터 설정도 변경해야 하지만 즉시 조치를 취해 문제가 되는 테이블을 vacuum해야 하기도 합니다. 이 경우 어떤 작업을 수행해야 하는지가 다음 절차에 나와 있습니다.

vacuum freeze를 수동으로 수행하려면

1. vacuum할 테이블이 포함되어 있는 데이터베이스에 세션 두 개를 엽니다. 두 번째 세션의 경우 "screen"을 사용하거나 연결이 끊긴 경우 세션을 유지하는 다른 유틸리티를 사용합니다.
2. 첫 번째 세션에서는 테이블에서 실행 중인 autovacuum 세션의 프로세스 ID(PID)를 가져옵니다.

   다음 쿼리를 실행하여 autovacuum 세션의 PID를 가져옵니다.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
   xact_start;
   ```
3. 세션 2에서 이 작업에 필요한 메모리 양을 계산합니다. 이 예제에서는 이 작업에 메모리를 최대 2GB까지 사용할 수 있는 것으로 보고 현재 세션의 `maintenance_work_mem`을 2GB로 설정합니다.

   ```
   SET maintenance_work_mem='2 GB';
   SET
   ```
4. 세션 2에서 테이블에 대한 `vacuum freeze verbose` 명령을 실행하십시오. 현재 PostgreSQL에서 이 작업에 대한 진행률 보고가 없음에도 작업을 확인할 수 있기 때문에 상세 정보 표시 설정이 유용하게 사용됩니다.

   ```
   \timing on
   Timing is on.
   vacuum freeze verbose pgbench_branches;
   ```

   ```
   INFO:  vacuuming "public.pgbench_branches"
   INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
   DETAIL:  0 index row versions were removed.
   0 index pages have been deleted, 0 are currently reusable.
   CPU 0.00s/0.00u sec elapsed 0.00 sec.
   INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
   DETAIL:  0 index row versions were removed.
   0 index pages have been deleted, 0 are currently reusable.
   CPU 0.00s/0.00u sec elapsed 0.00 sec.
   INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
        in 43 out of 43 pages
   DETAIL:  0 dead row versions cannot be removed yet.
   There were 9347 unused item pointers.
   ```

```
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. 세션 1에서 autovacuum이 vacuum 세션을 차단한 경우 `pg_stat_activity`에서 vacuum 세션에 대한 대기를 나타내는 T를 확인할 수 있습니다. 이 경우 다음과 같이 autovacuum 프로세스를 종료합니다.

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> Amazon Aurora의 일부 하위 버전은 위의 명령을 사용하여 autovacuum 프로세스를 종료할 수 없으며 `ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227` 오류와 함께 작업이 실패합니다. 패치된 PostgreSQL 버전을 찾으려면 Amazon Aurora PostgreSQL 업데이트에서 다음 글머리 기호를 검색합니다.
>
> ```
> Allow rds_superuser to terminate backends which are not explicitly
>  associated with a role
> ```

이때 세션이 시작됩니다. 이 테이블이 작업 목록에서 가장 상위에 있을 것이므로 autovacuum이 즉시 재시작됩니다.

6. 세션 2에서 `vacuum freeze verbose` 명령을 시작한 다음 세션 1에서 autovacuum 프로세스를 종료합니다.

Autovacuum이 실행 중인 경우 테이블 인덱스 다시 지정

인덱스가 손상되면 autovacuum은 계속해서 테이블을 처리하려 하고 실패합니다. 이 경우 수동 vacuum을 시도하면 다음과 비슷한 오류 메시지가 표시됩니다.

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

인덱스가 손상된 상태에서 테이블에 대해 autovacuum을 실행하려고 하면 이미 실행 중인 autovacuum 세션이 있음을 확인하게 됩니다. REINDEX 명령을 실행하면 테이블에 대한 단독 잠금을 해제합니다. 쓰기 작업과 해당 특정 인덱스를 사용하는 읽기 작업도 차단됩니다.

테이블에서 autovacuum을 실행할 때 테이블 인덱스를 다시 지정하려면

1. vacuum할 테이블이 포함되어 있는 데이터베이스에 세션 두 개를 엽니다. 두 번째 세션의 경우 "screen"을 사용하거나 연결이 끊긴 경우 세션을 유지하는 다른 유틸리티를 사용합니다.
2. 첫 번째 세션에서는 테이블에서 실행 중인 autovacuum 세션의 PID를 가져옵니다.

   다음 쿼리를 실행하여 autovacuum 세션의 PID를 가져옵니다.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
   xact_start;
   ```
3. 세션 2에서 reindex 명령을 실행합니다.

   ```
   \timing on
   Timing is on.
   reindex index pgbench_branches_test_index;
   REINDEX
     Time: 9.966 ms
   ```
4. 세션 1에서 autovacuum이 프로세스를 차단한 경우 `pg_stat_activity`에서 vacuum 세션에 대한 대기를 나타내는 "T"를 확인할 수 있습니다. 이 경우에는 autovacuum 프로세스를 종료합니다.

   ```
   SELECT pg_terminate_backend('the_pid');
   ```

   이때 세션이 시작됩니다. 이 테이블이 작업 목록에서 가장 상위에 있을 것이므로 autovacuum이 즉시 다시 시작된다는 점을 알아 두어야 합니다.
5. 세션 2에서 명령을 시작한 다음 세션 1에서 autovacuum 프로세스를 종료합니다.

대용량 인덱스를 사용하여 autovacuum 관리

작업 중에 autovacuum은 테이블에서 실행되는 동안 여러 vacuum 단계를 수행합니다. 테이블을 정리하기 전에 먼저 모든 인덱스에 vacuum을 실행합니다. 여러 개의 대용량 인덱스를 제거할 경우, 이 단계

는 상당한 시간과 리소스를 사용합니다. 따라서 테이블의 인덱스 수를 제어하고 사용하지 않는 인덱스를 제거하는 것이 가장 좋습니다.

이 프로세스에서는 먼저 전체 인덱스 크기를 확인합니다. 그런 다음, 다음 예제에 나온 것처럼 제거할 수 있는 사용하지 않는 인덱스가 있는지 확인합니다.

테이블 및 해당 인덱스의 크기를 확인하려면

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
pg_size_pretty
11 GB
(1 row)
```

이 예제에서는 인덱스 크기가 테이블보다 큽니다. 이러한 차이로 인해 인덱스가 팽창하거나 사용되지 않아 성능 문제가 발생하여 autovacuum 및 삽입 작업에 영향을 미칠 수 있습니다.

사용하지 않는 인덱스를 확인하려면

[pg_stat_user_indexes](https://www.postgresql.org/docs/current/monitoring-stats.html) 보기를 사용하면 인덱스가 `idx_scan` 열에 사용되는 빈도를 확인할 수 있습니다. 다음 예제를 보면 사용하지 않은 인덱스는 `idx_scan` 값이 `0`입니다.

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+---------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
```

```
16433   | 16453       | public      | pgbench_accounts | index_e                | 0
 | 0             | 0
16433   | 16451       | public      | pgbench_accounts | index_c                | 0
 | 0             | 0
16433   | 16449       | public      | pgbench_accounts | index_a                | 0
 | 0             | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

Note

이러한 통계는 통계가 재설정된 시점부터 증분됩니다. 사업 분기 말에만 사용되거나 특정 보고서에만 사용되는 인덱스가 있는 경우를 가정해 보겠습니다. 통계가 재설정된 이후로 이 인덱스가 사용되지 않았을 수 있습니다. 자세한 내용은 Statistics Functions(통계 함수)를 참조하세요. 고유성을 적용하는 데 사용되는 인덱스는 스캔이 실행되지 않으므로 사용하지 않는 인덱스로 식별해선 안 됩니다. 사용하지 않는 인덱스를 식별하려면 애플리케이션 및 해당 쿼리에 대한 심층적인 지식이 있어야 합니다.

데이터베이스의 통계가 마지막으로 재설정된 시간을 확인하려면 `pg_stat_database`를 사용하세요.

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+--------------------------------
```

```
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

테이블에 최대한 신속하게 Vacuum을 실행하는 방법

RDS for PostgreSQL 12 이상

대형 테이블에 인덱스가 너무 많으면 DB 인스턴스가 트랜잭션 ID 랩어라운드(XID)에 가까워질 수 있는데, 이때 XID 카운터가 0으로 래핑됩니다. 이 옵션을 선택하지 않으면 데이터가 손실될 수 있습니다. 그러나 인덱스를 정리하지 않고도 테이블에 신속하게 vacuum을 실행할 수 있습니다. RDS for PostgreSQL 12에서는 INDEX_CLEANUP 절과 함께 VACUUM을 사용할 수 있습니다.

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

autovacuum 세션이 이미 실행 중인 경우, 해당 세션을 종료하여 수동 VACUUM을 시작해야 합니다. 수동 vacuum freeze 수행에 대한 자세한 내용은 수동 vacuum freeze 수행 섹션을 참조하세요.

> **Note**
>
> 주기적인 인덱스 정리를 건너뛰면 인덱스 팽창이 발생하여 스캔 성능이 다운그레이드될 수 있습니다. 인덱스는 데드 행을 유지하고 테이블은 데드 라인 포인터를 유지합니다. 따라서 pg_stat_all_tables.n_dead_tup는 인덱스 정리가 포함된 수동 VACUUM 또는 autovacuum이 실행될 때까지 증가합니다. 가장 좋은 방법은 이 프로시저만 사용하여 트랜잭션 ID 랩어라운드를 방지하는 것입니다.

RDS for PostgreSQL 11 이상

그러나 RDS for PostgreSQL 11 이하 버전에서 vacuum을 더 신속하게 완료할 수 있는 유일한 방법은 테이블의 인덱스 수를 줄이는 것입니다. 인덱스를 삭제하면 쿼리 계획에 영향을 미칠 수 있습니다. 사

용하지 않는 인덱스를 먼저 삭제한 다음, XID 랩어라운드가 매우 가까워졌을 때 인덱스를 삭제하는 것이 좋습니다. vacuum 프로세스가 완료되면 이러한 인덱스를 다시 생성할 수 있습니다.

Autovacuum에 영향을 주는 기타 파라미터

이 쿼리를 사용하면 autovacuum 및 해당 동작에 직접 영향을 주는 일부 파라미터 값이 표시됩니다. autovacuum 파라미터는 PostgreSQL 설명서에 자세히 설명되어 있습니다.

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

모두 autovacuum에 영향을 주지만 가장 중요한 사항 몇 가지는 다음과 같습니다.

- maintenance_work_mem
- autovacuum_freeze_max_age
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

테이블 수준 Autovacuum 파라미터 설정

Autovacuum이 관련된 스토리지 파라미터를 테이블 수준에서 설정할 수 있습니다. 이 방법은 전체 데이터베이스의 동작을 변경하는 방법보다 더 나을 수 있습니다. 큰 테이블에 적극적인 설정을 지정해야

하지만 autovacuum이 모든 테이블에서 이와 같은 방식으로 작동하지 않도록 하려는 경우가 있을 수 있습니다.

이 쿼리를 사용하면 현재 어떤 테이블에 테이블 수준 옵션을 사용 중인지가 표시됩니다.

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

이 쿼리가 유용한 경우는 테이블 하나가 나머지 테이블보다 훨씬 더 큰 경우입니다. 300GB 테이블 하나와 1GB 미만의 테이블 30개가 있다고 가정하십시오. 이 경우 더 큰 테이블에 특정 파라미터를 설정하여 전체 시스템의 동작이 변경되지 않도록 할 수 있습니다.

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

이렇게 하면 시스템의 리소스를 더 많이 사용하는 대신 이 테이블의 비용에 따른 autovacuum 지연이 비활성화됩니다. 일반적으로 매시간 autovacuum_cost_limit에 도달한 autovacuum_vacuum_cost_delay의 autovacuum이 일시 중지됩니다. 자세한 내용은 비용에 따른 vacuum 수행에 대한 PostgreSQL 설명서에서 확인할 수 있습니다.

autovacuum 및 vacuum 활동 로그

autovacuum 활동에 대한 정보는 rds.force_autovacuum_logging_level 파라미터에 지정된 레벨을 기반으로 postgresql.log에 전송됩니다. 다음은 이 파라미터와 PostgreSQL 버전에 허용되는 해당 값이 기본 설정인 값입니다.

- disabled(PostgreSQL 10, PostgreSQL 9.6)
- debug5, debug4, debug3, debug2, debug1
- info(PostgreSQL 12, PostgreSQL 11)
- notice
- warning(PostgreSQL 13 이상)
- error, 로그, fatal, panic

rds.force_autovacuum_logging_level은 log_autovacuum_min_duration 파라미터와 함께 작동합니다. log_autovacuum_min_duration 파라미터 값은 autovacuum 작업이 기록되는 임계값(밀리초) 이상입니다. -1로 설정하면 아무것도 기록하지 않지만 0으로 설

정하면 모든 작업이 기록됩니다. `rds.force_autovacuum_logging_level`과 마찬가지로 `log_autovacuum_min_duration`의 기본값은 다음과 같이 버전에 따라 다릅니다.

- `10000 ms` - PostgreSQL 14, PostgreSQL 13, PostgreSQL 12, PostgreSQL 11
- `(empty)` - PostgreSQL 10 및 PostgreSQL 9.6 기본값 없음

`rds.force_autovacuum_logging_level`을 `WARNING`로 설정하는 것이 좋습니다. 또한 `log_autovacuum_min_duration`을 1000에서 5000까지의 값으로 설정하는 것이 좋습니다. 5,000 밀리초 이상 걸리는 5000개의 기록 활동 설정. -1을 제외한 모든 설정은 충돌하는 잠금 또는 동시에 삭제된 관계로 인해 autovacuum 작업을 건너뛴 경우에도 메시지를 기록합니다. 자세한 내용은 PostgreSQL 설명서의 [자동 Vacuuming](#)을 참조하세요.

문제를 해결하려면 `rds.force_autovacuum_logging_level` 매개 변수를 상세 표시 정보에 대해 debug1부터 debug5까지의 디버그 레벨 중 하나로 변경할 수 있습니다. 디버그 설정은 단기 문제 해결 목적으로만 사용하는 것이 좋습니다. 자세한 내용은 PostgreSQL의 [로그 시기](#) 문서를 참조하세요.

> **Note**
>
> PostgreSQL을 사용하면 `rds_superuser` 계정에서 `pg_stat_activity`의 autovacuum 세션을 볼 수 있습니다. 예: 명령의 실행을 차단하거나, 수동으로 실행한 vacuum 명령보다 느리게 실행되는 autovacuum 세션을 식별 및 종료 가능

잘못된 데이터베이스를 사용한 autovacuum 동작 이해

DROP DATABASE 작업 도중에 중단된 데이터베이스를 유효하지 않은 것으로 나타내는 새 `-2` 값이 `pg_database` 카탈로그의 `datconnlimit` 열에 도입되었습니다.

이 새 값은 다음과 같은 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.4 이상의 모든 버전
- 14.9 이상 버전
- 13.12 이상 버전
- 12.16 이상 버전
- 11.21 이상 버전

잘못된 데이터베이스는 유효한 데이터베이스의 기능을 중지하는 autovacuum 기능에 영향을 주지 않습니다. autovacuum은 잘못된 데이터베이스를 무시합니다. 따라서 PostgreSQL 환경의 모든 유효한 데이터베이스에 대해 정기적인 vacuum 작업이 계속해서 적절하고 효율적으로 작동합니다.

주제

- [트랜잭션 ID 모니터링]
- [모니터링 쿼리 조정]
- [잘못된 데이터베이스 문제 해결]

트랜잭션 ID 모니터링

`age(datfrozenxid)` 함수는 일반적으로 데이터베이스의 트랜잭션 ID(XID) 수명을 모니터링하여 트랜잭션 ID 랩어라운드를 방지하는 데 사용됩니다.

유효하지 않은 데이터베이스는 autovacuum에서 제외되므로, 트랜잭션 ID(XID) 횟수가 최대값인 2 `billion`에 도달하여 `- 2 billion`로 순환하며 이 주기를 무한정 계속할 수 있습니다. 트랜잭션 ID 랩어라운드를 모니터링하는 일반적인 쿼리는 다음과 같습니다.

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

그러나 `datconnlimit`에 대한 -2 값이 도입되면 잘못된 데이터베이스가 이 쿼리의 결과를 왜곡할 수 있습니다. 이러한 데이터베이스는 유효하지 않으며 정기적인 유지 관리 검사에 포함되어서는 안 되므로, 오탐을 유발할 수 있어 실제보다 `age(datfrozenxid)`가 높다고 믿게 됩니다.

모니터링 쿼리 조정

정확한 모니터링을 위해서는 잘못된 데이터베이스를 제외하도록 모니터링 쿼리를 조정해야 합니다. 다음 권장 쿼리를 따르세요.

```
SELECT
    max(age(datfrozenxid))
FROM
    pg_database
WHERE
    datconnlimit <> -2;
```

이 쿼리를 사용하면 PostgreSQL 환경 전체의 트랜잭션 ID 사용 기간을 정확하게 반영하여 유효한 데이터베이스만 `age(datfrozenxid)` 계산에 고려합니다.

잘못된 데이터베이스 문제 해결

잘못된 데이터베이스에 연결하려고 하면 다음과 같은 오류 메시지가 표시될 수 있습니다.

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

또한, `log_min_messages` 파라미터가 `DEBUG2` 이상으로 설정된 경우 autovacuum 프로세스가 잘못된 데이터베이스를 건너뛰고 있음을 나타내는 다음과 같은 로그 항목을 확인할 수 있습니다.

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

이 문제를 해결하려면 연결 시도 중에 제공된 `HINT`를 따르세요. RDS 마스터 계정 또는 `rds_superuser` 역할이 있는 데이터베이스 계정을 사용하여 유효한 데이터베이스에 연결하고 잘못된 데이터베이스를 삭제합니다.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

Aurora PostgreSQL에서 공격적인 vacuum 블로커 식별 및 해결

PostgreSQL에서는 vacuum 작업이 스토리지를 회수하고 트랜잭션 ID 랩어라운드 문제를 방지하기 때문에 데이터베이스의 좋은 상태를 위해 vacuum 작업이 중요합니다. 그러나 vacuum 작업이 원하는 대로 작동하지 않게 되는 경우가 있어 성능이 저하되고, 스토리지가 팽창하고, 트랜잭션 ID 랩어라운드로 인해 DB 인스턴스의 가용성이 영향을 받을 수 있습니다. 따라서 최적의 데이터베이스 성능과 가용성을 위해서는 이러한 문제를 식별하고 해결하는 것이 필수적입니다. Autovacuum에 대해 자세히 알아보려면 Understanding autovacuum in Amazon RDS for PostgreSQL environments를 읽어보세요.

`postgres_get_av_diag()` 함수는 공격적 vacuum 진행 상황을 방지하거나 지연시키는 문제를 식별하는 데 도움이 됩니다. 제안이 제공되며, 여기에는 문제를 식별할 수 있는 경우 문제 해결을 위한 명령이, 문제를 식별할 수 없는 경우 추가 진단을 위한 지침이 포함될 수 있습니다. 공격적 vacuum 블로커는 연령이 RDS의 적응형 autovacuum 임곗값인 트랜잭션 ID 5억 개를 초과할 때 보고됩니다.

트랜잭션 ID의 연령이란 무엇인가요?

트랜잭션 ID에 대한 `age()` 함수는 데이터베이스(`pg_database.datfrozenxid`) 또는 테이블(`pg_class.relfrozenxid`)에 대해 가장 오래된 고정되지 않은 트랜잭션 ID 이후 발생한 트랜잭션 수를 계산합니다. 이 값은 마지막 공격적 vacuum 작업 이후의 데이터베이스 활동을 나타내며 향후 VACUUM 프로세스에 대해 가능한 워크로드를 강조합니다.

공격적 vacuum이란 무엇인가요?

공격적인 VACUUM 작업은 일반적인 VACUUM에서는 대개 건너 뛰는 페이지를 포함하여 테이블 내에 있는 모든 페이지에 대한 종합적인 스캔을 수행합니다. 이 철저한 스캔의 목표는 최대 연령에 근접한 트랜잭션 ID를 '고정'하여 트랜잭션 ID 랩어라운드라고 하는 상황을 효과적으로 방지하는 것을 목표로 합니다.

`postgres_get_av_diag()`가 블로커를 보고하려면 블로커의 연령이 최소 5억 개의 트랜잭션이어야 합니다.

주제

- Aurora PostgreSQL에 자동 정리 모니터링 및 진단 도구 설치
- Aurora PostgreSQL의 postgres_get_av_diag() 기능
- Aurora PostgreSQL에서 식별 가능한 vacuum 블로커 해결
- Aurora PostgreSQL에서 식별 불가능한 vacuum 블로커 해결
- Aurora PostgreSQL의 vacuum 성능 문제 해결

- [Aurora PostgreSQL의 NOTICE 메시지 설명]

Aurora PostgreSQL에 자동 정리 모니터링 및 진단 도구 설치

`postgres_get_av_diag()` 함수는 현재 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 17.4 이상의 17 버전
- 16.7 이상의 16 버전
- 15.11 이상의 15 버전
- 14.16 이상의 14 버전
- 13.19 이상의 13 버전

`postgres_get_av_diag()`를 사용하려면 `rds_tools` 확장을 만듭니다.

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

확장이 설치되어 있는지 확인합니다.

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                       Description
 ----------+---------+-----------
+--------------------------------------------------------
 rds_tools |   1.9   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

함수가 만들어졌는지 확인합니다.

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
```

```
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

Aurora PostgreSQL의 postgres_get_av_diag() 기능

`postgres_get_av_diag()` 함수는 Aurora PostgreSQL 데이터베이스에서 차단하거나 지연되는 자동 정리 프로세스에 대한 진단 정보를 검색합니다. 정확한 결과를 얻으려면 가장 오래된 트랜잭션 ID로 데이터베이스에서 쿼리를 실행해야 합니다. 가장 오래된 트랜잭션 ID로 데이터베이스를 사용하는 방법에 대한 자세한 내용은 Not connected to the database with the age of oldest transaction ID를 참조하세요.

```
SELECT
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

`postgres_get_av_diag()` 함수는 다음 정보를 반환합니다.

blocker

Vacuum을 차단하는 데이터베이스 활동의 범주를 보여줍니다.

- 활성 문
- 트랜잭션의 유휴 상태
- 준비된 트랜잭션
- 논리적 복제 슬롯
- 리더 인스턴스
- 임시 테이블

데이터베이스

적용 가능하고 지원되는 경우 데이터베이스의 이름을 보여줍니다. 활동이 진행 중이고 autovacuum을 차단 중이거나 차단할 데이터베이스입니다. 연결하고 조치를 취해야 하는 데이터베이스입니다.

blocker_identifier

Autovacuum을 차단 중이거나 차단할 활동의 식별자를 보여줍니다. 식별자는 SQL 문, 준비된 트랜잭션, 읽기 전용 복제본의 IP 주소, 논리적 또는 물리적 복제 슬롯의 이름과 함께 프로세스 ID일 수 있습니다.

wait_event

차단 세션의 [대기 이벤트]를 지정하고 다음 블로커에 적용할 수 있습니다.

- 활성 문
- 트랜잭션의 유휴 상태

autovacum_lagging_by

범주당 백로그 작업에서 autovacuum이 지연되는 트랜잭션 수를 보여줍니다.

suggestion

블로커를 해결하기 위한 제안을 보여줍니다. 이러한 지침에는 해당하는 경우 활동이 존재하는 데이터베이스의 이름, 해당하는 경우 세션의 프로세스 ID(PID) 및 수행할 작업이 포함됩니다.

suggested_action

블로커를 해결하기 위해 수행해야 하는 작업을 제안합니다.

Aurora PostgreSQL에서 식별 가능한 vacuum 블로커 해결

Autovacuum은 공격적인 vacuum을 수행하고 트랜잭션 ID의 연령을 RDS 인스턴스의 `autovacuum_freeze_max_age` 파라미터에 의해 지정된 임곗값 미만으로 낮춥니다. 이 연령은 Amazon CloudWatch 지표 `MaximumUsedTransactionIDs`를 사용하여 추적할 수 있습니다.

Amazon RDS 인스턴스에 대한 `autovacuum_freeze_max_age`의 설정(기본값은 2억 개의 트랜잭션 ID)을 찾으려면 다음 쿼리를 사용할 수 있습니다.

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
```

```
    name = 'autovacuum_freeze_max_age';
```

postgres_get_av_diag()는 연령이 Amazon RDS의 적응형 autovacuum 임곗값인 트랜잭션 ID 5억 개를 초과할 때만 공격적 vacuum 블로커를 확인합니다. postgres_get_av_diag()가 블로커를 감지하려면 블로커의 연령이 최소 5억 개의 트랜잭션이어야 합니다.

postgres_get_av_diag() 함수는 다음과 같은 유형의 블로커를 식별합니다.

주제

- 활성 문
- 트랜잭션의 유휴 상태
- 준비된 트랜잭션
- 논리적 복제 슬롯
- 리더 인스턴스
- 임시 테이블

활성 문

PostgreSQL에서 활성 문은 현재 데이터베이스에서 실행 중인 SQL 문입니다. 여기에는 쿼리, 트랜잭션 또는 진행 중인 작업이 포함됩니다. pg_stat_activity를 통해 모니터링할 때 상태 열은 해당 PID가 있는 프로세스가 활성 상태임을 나타냅니다.

postgres_get_av_diag() 함수는 활성 문인 문을 식별할 때 다음과 유사한 출력을 표시합니다.

```
blocker                | Active statement
database               | my_database
blocker_identifier     | SELECT pg_sleep(20000);
wait_event             | Timeout:PgSleep
autovacuum_lagging_by  | 568,600,871
suggestion             | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action       | {"SELECT pg_terminate_backend (29621);"}
```

권장 조치

suggestion 열의 지침에 따라 사용자는 활성 문이 있는 데이터베이스에 연결할 수 있으며 suggested_action 열에 지정된 대로 세션을 종료하는 옵션을 주의 깊게 검토하는 것이 좋습니다. 종료가 안전한 경우 pg_terminate_backend() 함수를 사용하여 세션을 종료할 수 있습니다. 이 작

업은 관리자(예: RDS 마스터 계정) 또는 필요한 `pg_terminate_backend()` 권한이 있는 사용자가 수행할 수 있습니다.

> **Warning**
>
> 종료된 세션은 해당 세션이 만든 (ROLLBACK) 변경 사항을 취소합니다. 요구 사항에 따라 문을 다시 실행할 수 있습니다. 그러나 autovacuum 프로세스가 공격적인 vacuum 작업을 완료한 후에만 다시 실행하는 것이 좋습니다.

트랜잭션의 유휴 상태

트랜잭션 문의 유휴는 명시적 트랜잭션을 열고(예: `BEGIN` 명령문을 발급하여), 일부 작업을 수행했으며, 이제 클라이언트가 더 많은 작업을 전달하거나 `COMMIT`, `ROLLBACK`또는 `END`(암시적 `COMMIT`을 유발함)를 발급하여 트랜잭션 종료 신호를 보내기를 기다리는 세션을 나타냅니다.

`postgres_get_av_diag()` 함수는 `idle in transaction` 문을 블로커로 식별할 때 다음과 유사한 출력을 표시합니다.

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

권장 조치

`suggestion` 열에 표시된 대로 트랜잭션 세션에 유휴가 있는 데이터베이스에 연결하고 `pg_terminate_backend()` 함수를 사용하여 세션을 종료할 수 있습니다. 사용자는 관리자(RDS 마스터 계정) 사용자이거나 `pg_terminate_backend()` 권한이 있는 사용자일 수 있습니다.

> **Warning**
>
> 종료된 세션은 해당 세션이 만든 (ROLLBACK) 변경 사항을 취소합니다. 요구 사항에 따라 문을 다시 실행할 수 있습니다. 그러나 autovacuum 프로세스가 공격적인 vacuum 작업을 완료한 후에만 다시 실행하는 것이 좋습니다.

준비된 트랜잭션

PostgreSQL은 [준비된 트랜잭션](https://www.postgresql.org/docs/current/sql-prepare-transaction.html)이라고 하는 두 단계 커밋 전략의 일부인 트랜잭션을 허용합니다. max_prepared_transactions 파라미터를 0이 아닌 값으로 설정하면 활성화됩니다. 준비된 트랜잭션은 데이터베이스 충돌, 재시작 또는 클라이언트 연결 해제 후에도 트랜잭션이 내구성을 유지하고 계속 가능하도록 하기 위해 설계되었습니다. 일반 트랜잭션과 마찬가지로 트랜잭션 ID가 할당되며 autovacuum에 영향을 미칠 수 있습니다. 준비된 상태로 두면 autovacuum이 고정을 수행할 수 없으며 트랜잭션 ID 랩어라운드로 이어질 수 있습니다.

트랜잭션 관리자가 확인하지 않고 트랜잭션을 무기한으로 준비된 상태로 두면 고립된 준비 트랜잭션이 됩니다. 이 문제를 해결하는 유일한 방법은 COMMIT PREPARED 또는 ROLLBACK PREPARED 명령을 각각 사용하여 트랜잭션을 커밋하거나 롤백하는 것입니다.

> **Note**
>
> 준비된 트랜잭션 중에 가져온 백업에는 복원 후에도 해당 트랜잭션이 계속 포함되어 있습니다. 이러한 트랜잭션을 찾고 닫는 방법은 다음 정보를 참조하세요.

준비된 트랜잭션인 블로커를 식별하면 postgres_get_av_diag() 함수에 다음 출력이 표시됩니다.

```
blocker                | Prepared transaction
database               | my_database
blocker_identifier     | myptx
wait_event             | Not applicable
autovacuum_lagging_by  | 1,805,802,632
suggestion             | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action       | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

권장 조치

제안 열에 언급된 대로 준비된 트랜잭션이 있는 데이터베이스에 연결합니다. suggested_action 열을 기반으로 COMMIT과 ROLLBACK 중 무엇을 수행할지 주의 깊게 검토하고 작업을 적절히 적용합니다.

준비된 트랜잭션을 일반적으로 모니터링하기 위해 PostgreSQL은 pg_prepared_xacts라는 카탈로그 보기를 제공합니다. 다음 쿼리를 사용하여 준비된 트랜잭션을 찾을 수 있습니다.

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
ORDER BY
    age(transaction) DESC;
```

논리적 복제 슬롯

복제 슬롯의 목적은 사용되지 않은 변경 사항이 대상 서버에 복제될 때까지 이를 유지하는 것입니다. 자세한 내용은 PostgreSQL의 [Logical replication](#)을 참조하세요.

논리적 복제 슬롯에는 두 가지 유형이 있습니다.

비활성 논리적 복제 슬롯

복제가 종료되면 소비되지 않은 트랜잭션 로그를 제거할 수 없으며 복제 슬롯이 비활성화됩니다. 비활성 논리적 복제 슬롯은 현재 구독자가 사용하지 않지만 서버에 남아 있어 WAL 파일이 보존되고 이전 트랜잭션 로그를 제거할 수 없습니다. 이렇게 되면 LSN 정보를 덮어쓰지 않도록 시스템이 보존해야 하므로 디스크 사용량이 증가하고 특히 autovacuum이 내부 카탈로그 테이블을 정리하지 못하게 됩니다. 해결하지 않으면 카탈로그 팽창, 성능 저하 및 랩어라운드 vacuum 위험 증가로 이어질 수 있으며 이로 인해 트랜잭션 가동 중지 시간이 발생할 수도 있습니다.

활성이지만 느린 논리적 복제 슬롯

논리적 복제의 성능 저하로 인해 카탈로그의 데드 튜플 제거가 지연되는 경우가 있습니다. 이러한 복제 지연으로 인해 `catalog_xmin`의 업데이트 속도가 느려지고 카탈로그 팽창 및 랩어라운드 vacuum이 발생할 수 있습니다.

`postgres_get_av_diag()` 함수는 논리적 복제 슬롯을 블로커로 찾았을 때 다음과 유사한 출력을 표시합니다.

```
blocker               | Logical replication slot
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
```

```
suggestion             | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
suggested_action       | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

권장 조치

이 문제를 해결하려면 대상 스키마 또는 적용 프로세스를 종료할 수 있는 데이터에 문제가 있는지 복제 구성을 확인합니다. 가장 일반적인 이유는 다음과 같습니다.

- 열 누락
- 호환되지 않는 데이터 유형
- 데이터 불일치
- 누락된 테이블

문제가 인프라 문제와 관련된 경우:

- 네트워크 문제 - 호환되지 않는 네트워크 상태의 Amazon RDS DB 관련 문제를 해결하려면 어떻게 해야 하나요?
- 데이터베이스 또는 DB 인스턴스를 사용할 수 없는 이유는 다음과 같습니다.
  - 복제본 인스턴스의 스토리지가 부족함 - 스토리지 추가에 대한 자세한 내용은 Amazon RDS DB instances run out of storage를 검토하세요.
  - 호환되지 않는 파라미터 - 문제를 해결하는 방법에 대한 자세한 내용은 호환되지 않는 파라미터 상태에서 멈춘 Amazon RDS DB 인스턴스를 해결하려면 어떻게 해야 하나요?를 검토하세요.

인스턴스가 AWS 네트워크 외부 또는 AWS EC2에 있는 경우 관리자에게 가용성 또는 인프라 관련 문제를 해결하는 방법을 문의하세요.

비활성 슬롯 삭제

> **Warning**
>
> 주의: 복제 슬롯을 삭제하기 전에 복제가 진행 중이지 않고 비활성 상태이며 복구할 수 없는 상태인지 주의 깊게 확인합니다. 슬롯을 성급히 삭제하면 복제가 중단되거나 데이터가 손실될 수 있습니다.

복제 슬롯이 더 이상 필요하지 않음을 확인한 후 autovacuum이 계속되도록 슬롯을 삭제합니다. `active = 'f'` 조건은 비활성 슬롯만 삭제되도록 합니다.

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

리더 인스턴스

hot_standby_feedback 설정이 활성화되면 라이터 인스턴스의 자동 정리가 리더 인스턴스에서 실행 중인 쿼리에 여전히 필요할 수 있는 데드 행을 제거하는 것을 방지합니다. 리더 인스턴스(Aurora 글로벌 데이터베이스의 리더 인스턴스에도 해당)에서 실행되는 쿼리는 쿼리 충돌 및 취소를 방지하는 라이터 인스턴스에서 해당 행을 계속 사용할 수 있어야 하기 때문에 이 동작이 필요합니다.

> **Note**
>
> `hot_standby_feedback`은 Aurora PostgreSQL에서 기본적으로 활성화되며 수정할 수 없습니다.

`postgres_get_av_diag()` 함수는 물리적 복제 슬롯을 사용하는 읽기 전용 복제본을 블로커로 찾았을 때 다음과 유사한 출력을 표시합니다.

```
blocker                | Oldest query running on aurora reader
database               | Not applicable
blocker_identifier     | my-aurora-reader-2
wait_event             | Not applicable
autovacuum_lagging_by  | 540,122,859
suggestion             | Run the following query on the reader "my-aurora-reader-2" to
 find the long running query:

                       | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 523476310;

                       | Review carefully and you may consider terminating the query on
 reader using suggested_action.
suggested_action       | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 523476310;","

                       | [OR]


                       | ","Delete the reader if not needed"}
```

suggested_action 열에서 권장하는 대로 이러한 옵션을 주의 깊게 검토하여 autovacuum 차단을 해제합니다.

- 쿼리 종료 - 제안 열의 지침에 따라 suggested_action 열에 지정된 대로 읽기 전용 복제본에 연결할 수 있습니다. 세션을 종료하는 옵션을 주의 깊게 검토하는 것이 좋습니다. 종료가 안전한 것으로 간주되는 경우 pg_terminate_backend() 함수를 사용하여 세션을 종료할 수 있습니다. 이 작업은 관리자(예: RDS 마스터 계정) 또는 필요한 pg_terminate_backend() 권한이 있는 사용자가 수행할 수 있습니다.

  읽기 전용 복제본에서 다음 SQL 명령을 실행하여 기본의 vacuum이 오래된 행을 정리하지 못하게 하는 쿼리를 종료할 수 있습니다. backend_xmin의 값은 함수의 출력에 보고됩니다.

  ```
  SELECT
      pg_terminate_backend(pid)
  FROM
      pg_catalog.pg_stat_activity
  WHERE
      backend_xmin::text::bigint = backend_xmin;
  ```

- 필요하지 않은 경우 리더 인스턴스 삭제 - 리더 인스턴스가 더 이상 필요하지 않은 경우 삭제할 수 있습니다. 이렇게 하면 연결된 복제 오버헤드가 제거되고 프라이머리 인스턴스가 인스턴스에 의해 지연되지 않고 트랜잭션 로그를 재활용할 수 있습니다.

임시 테이블

TEMPORARY 키워드를 사용하여 만든 임시 테이블은 pg_temp_xxx와 같은 임시 스키마에 상주하며 해당 테이블을 만든 세션에서만 액세스할 수 있습니다. 세션이 종료되면 임시 테이블이 삭제됩니다. 그러나 이러한 테이블은 PostgreSQL의 autovacuum 프로세스에는 보이지 않으며 테이블을 만든 세션에서 수동으로 vacuum 처리해야 합니다. 다른 세션에서 임시 테이블을 정리하려고 해도 아무런 효과가 없습니다.

비정상적인 상황에서는 임시 테이블이 활성 세션 없이 존재합니다. 치명적인 충돌, 네트워크 문제 또는 유사한 이벤트로 인해 소유 세션이 예기치 않게 종료되는 경우 임시 테이블이 정리되지 않아 '고립된' 테이블로 남겨질 수 있습니다. PostgreSQL autovacuum 프로세스가 고립된 임시 테이블을 감지하면 다음 메시지를 로깅합니다.

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

postgres_get_av_diag() 함수는 임시 테이블을 블로커로 식별할 때 다음과 유사한 출력을 표시합니다. 함수가 임시 테이블과 관련된 출력을 올바르게 표시하려면 해당 테이블이 있는 것과 동일한 데이터베이스 내에서 실행해야 합니다.

```
blocker                 | Temporary table
database                | my_database
blocker_identifier      | pg_temp_14.ttemp
wait_event              | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion              | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action        | {"DROP TABLE ttemp;"}
```

권장 조치

출력의 suggestion 열에 제공된 지침에 따라 autovacuum 실행을 막는 임시 테이블을 식별하고 제거합니다. 다음 명령을 사용하여 postgres_get_av_diag()에서 보고한 임시 테이블을 삭제합니다. postgres_get_av_diag() 함수에서 제공하는 출력을 기반으로 테이블 이름을 바꿉니다.

```
DROP TABLE my_temp_schema.my_temp_table;
```

다음 쿼리를 사용하여 임시 테이블을 식별할 수 있습니다.

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
ORDER BY
    age(relfrozenxid) DESC;
```

Aurora PostgreSQL에서 식별 불가능한 vacuum 블로커 해결

이 섹션에서는 vacuum 작업이 진행되지 않는 다른 이유를 살펴봅니다. 이러한 문제는 현재 postgres_get_av_diag() 함수를 사용하여 직접 식별할 수 없습니다.

주제

- 인덱스 불일치
- 매우 높은 트랜잭션 속도

인덱스 불일치

논리적으로 일치하지 않는 인덱스는 autovacuum이 진행되는 것을 방지할 수 있습니다. 다음 오류 또는 유사한 오류는 인덱스의 vacuum 단계 동안 또는 SQL 문에서 인덱스에 액세스할 때 로깅됩니다.

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

지침

인덱스를 다시 구축하거나 수동 `VACUUM FREEZE`에서 `INDEX_CLEANUP`을 사용하여 인덱스를 건너뜁니다.

- CONCURRENTLY 옵션 사용 – PostgreSQL 버전 12 전에는 인덱스를 다시 구축하려면 테이블에 대한 액세스를 제한하는 배타적인 테이블 잠금이 필요했습니다. PostgreSQL 버전 12 이상에서는 CONCURRENTLY 옵션을 사용하여 행 수준 잠금을 허용하므로 테이블의 가용성이 크게 향상됩니다. 다음은 명령입니다.

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  CONCURRENTLY는 중단이 적지만 사용량이 많은 테이블에서는 속도가 느릴 수 있습니다. 가능하면 트래픽이 적은 기간에 인덱스를 구축하는 것이 좋습니다. 자세한 내용은 PostgreSQL 설명서의 REINDEX를 참조하세요.
- INDEX_CLEANUP FALSE 옵션 사용 - 인덱스가 크고 완료하는 데 상당한 시간이 필요할 것으로 예상되는 경우 인덱스를 제외하면서 수동 VACUUM FREEZE를 실행하여 autovacuum 차단을 해제할 수 있습니다. 이 기능은 PostgreSQL 버전 12 이상에서 사용할 수 있습니다.

  인덱스를 우회하면 일관되지 않은 인덱스의 vacuum 프로세스를 건너뛰고 랩어라운드 문제를 완화할 수 있습니다. 그러나 이렇게 해도 기본 유효하지 않은 페이지 문제는 해결되지 않습니다. 유효하지 않은 페이지 문제를 완전히 해결하려면 인덱스를 다시 구축해야 합니다.

매우 높은 트랜잭션 속도

PostgreSQL에서 높은 트랜잭션 속도는 autovacuum의 성능에 상당한 영향을 미쳐 데드 튜플의 정리가 느려지고 트랜잭션 ID 랩어라운드 위험이 증가할 수 있습니다. 두 기간의 `max(age(datfrozenxid))` 차이(일반적으로 초당)를 측정하여 트랜잭션 속도를 모니터링할 수 있습니다. 또한 RDS 성능 개선 도우미의 다음 카운터 지표를 사용하여 총 트랜잭션 수인 트랜잭션 속도(xact_commit과 xact_rollback의 합계)를 측정할 수 있습니다.

카운터	유형	Unit	지표
xact_commit	트랜잭션	초당 커밋 수	db.Transactions.xact_commit
xact_rollback	트랜잭션	초당 롤백 수	db.Transactions.xact_rollback

빠른 증가는 트랜잭션 로드가 높음을 나타내며, 이는 autovacuum에 부담을 주어 팽창, 잠금 경합 및 잠재적 성능 문제를 일으킬 수 있습니다. 이는 다음과 같은 몇 가지 방법으로 autovacuum 프로세스에 부정적인 영향을 미칠 수 있습니다.

- 테이블 활동: Vacuum 처리 중인 특정 테이블에 대량의 트랜잭션이 발생하여 지연이 발생할 수 있습니다.
- 시스템 리소스: 전체 시스템에 과부하가 발생하여 autovacuum이 효율적으로 작동하는 데 필요한 리소스에 액세스하기 어려울 수 있습니다.

Autovacuum이 더 효과적으로 작동하고 작업을 따라잡을 수 있도록 하려면 다음 전략을 고려하세요.

1. 가능하면 트랜잭션 속도를 줄입니다. 가능한 경우 유사한 트랜잭션을 배치 처리하거나 그룹화하는 것을 고려합니다.
2. 사용량이 적은 시간에 야간, 주간 또는 격주로 수동 `VACUUM FREEZE` 작업을 사용하여 자주 업데이트되는 테이블을 타게팅합니다.
3. 인스턴스 클래스를 스케일 업하여 높은 트랜잭션 볼륨과 autovacuum을 처리할 수 있도록 더 많은 시스템 리소스를 할당하는 것을 고려합니다.

Aurora PostgreSQL의 vacuum 성능 문제 해결

이 섹션에서는 느린 vacuum 성능에 영향을 미치는 요인과 이러한 문제를 해결하는 방법을 설명합니다.

주제

- 큰 인덱스 vacuum
- 테이블 또는 데이터베이스가 너무 많아 vacuum 작업을 할 수 없음
- 공격적 vacuum(랩어라운드 방지용) 실행 중

큰 인덱스 vacuum

VACUUM은 초기화, 힙 스캔, 인덱스 및 힙 vacuum, 인덱스 정리, 힙 잘라내기 및 최종 정리와 같은 순차적 단계를 거칩니다. 힙 스캔 중에 프로세스는 페이지를 정리하고 조각 모음을 하여 고정합니다. 힙 스캔이 완료되면 VACUUM은 인덱스를 정리하고 빈 페이지를 운영 체제로 반환하며 빈 공간 맵 vacuum 및 통계 업데이트와 같은 최종 정리 작업을 수행합니다.

인덱스를 vacuum할 때 `maintenance_work_mem`(또는 `autovacuum_work_mem`)이 인덱스를 처리하기에 충분하지 않은 경우 여러 번 전달해야 할 수 있습니다. PostgreSQL 16 이하에서는 데드 튜플 ID 저장을 위한 1GB 메모리 제한이 있어 큰 인덱스에서는 여러 번의 전달이 강제될 때가 많았습니다. PostgreSQL 17에는 단일 할당 배열을 사용하는 대신 메모리를 동적으로 할당하는 `TidStore`가 도입되었습니다. 이렇게 하면 1GB 제약이 제거되고, 메모리를 더 효율적으로 사용하며, 인덱스당 여러 인덱스 스캔의 필요성이 줄어듭니다.

사용 가능한 메모리가 전체 인덱스 처리를 한 번에 수용할 수 없는 경우에도 큰 인덱스는 PostgreSQL 17에서 여러 번 전달해야 할 수 있습니다. 일반적으로 큰 인덱스에는 여러 번 전달해야 하는 데드 튜플이 더 많이 포함되어 있습니다.

느린 vacuum 작업 탐지

`postgres_get_av_diag()` 함수는 메모리 부족으로 인해 vacuum 작업이 느리게 실행되는 시기를 탐지할 수 있습니다. 이 함수에 대한 자세한 내용은 Aurora PostgreSQL에 자동 정리 모니터링 및 진단 도구 설치 단원을 참조하십시오.

사용 가능한 메모리가 단일 패스에서 인덱스 vacuuming을 완료하기에 충분하지 않은 경우 `postgres_get_av_diag()` 함수는 다음 알림을 발행합니다.

`rds_tools` 1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
        memory. The suggested value is an estimate based on the current number of dead
 tuples for the table being vacuumed, which might not fully reflect the latest state.
 Additionally, review the possibility of manual
        vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
 TRUE) table_name;). For more information, see

        Working with PostgreSQL autovacuum in the Amazon Aurora User Guide.
```

> **Note**
>
> 이 `postgres_get_av_diag()` 함수는 인덱스 vacuum 작업에 필요한 메모리의 양을 추정하기 위해 `pg_stat_all_tables.n_dead_tup`에 의존합니다.

`postgres_get_av_diag()` 함수가 `autovacuum_work_mem` 부족으로 인해 여러 인덱스 스캔이 필요한 느린 vacuum 작업을 식별하면 다음 메시지가 생성됩니다.

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
        For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

지침

수동 `VACUUM FREEZE`를 사용해 다음 해결 방법을 적용하여 테이블 고정의 속도를 높일 수 있습니다.

Vacuum 작업을 위한 메모리 증가

`postgres_get_av_diag()` 함수에서 제안한 대로 인스턴스 수준에서 잠재적 메모리 제약을 해결하기 위해 `autovacuum_work_mem` 파라미터를 늘리는 것이 좋습니다. `autovacuum_work_mem`은 동적 파라미터이지만 새 메모리 설정이 적용되려면 autovacuum 대몬이 작업자를 다시 시작해야 합니다. 이를 수행하는 방법은 다음과 같습니다.

1. 새 설정이 있는지 확인합니다.

2. 현재 autovacuum을 실행 중인 프로세스를 종료합니다.

이 접근 방식을 사용하면 조정된 메모리 할당이 새 autovacuum 작업에 적용됩니다.

보다 즉각적인 결과를 얻으려면 세션 내에서 `maintenance_work_mem` 설정이 증가된 VACUUM FREEZE 작업을 수동으로 수행하는 것을 고려하세요.

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

Amazon RDS를 사용하고 있고 `maintenance_work_mem` 또는 `autovacuum_work_mem`에 대해 더 높은 값을 지원하는 데 추가 메모리가 필요한 경우 메모리가 더 많은 인스턴스 클래스로 업그레이드하는 것을 고려해 보세요. 이를 통해 수동 및 자동 vacuum 작업을 모두 개선하는 데 필요한 리소스를 제공하여 전반적인 vacuum 및 데이터베이스 성능을 개선할 수 있습니다.

INDEX_CLEANUP 비활성화

PostgreSQL 버전 12 이상의 수동 VACUUM에서는 인덱스 정리 단계를 건너뛸 수 있고, PostgreSQL 버전 14 이상의 긴급 autovacuum에서는 [`vacuum_failsafe_age`](#) 파라미터를 기반으로 자동으로 이 작업을 수행합니다.

> **Warning**
>
> 인덱스 정리를 건너뛰면 인덱스 팽창이 발생하여 쿼리 성능에 부정적인 영향을 미칠 수 있습니다. 이를 완화하려면 영향을 받는 인덱스를 유지 관리 기간 동안 다시 인덱싱하거나 vacuum하는 것이 좋습니다.

큰 인덱스 처리에 대한 추가 지침은 [대용량 인덱스를 사용하여 autovacuum 관리](#) 의 설명서를 참조하세요.

병렬 인덱스 vacuum 작업

PostgreSQL 13부터 인덱스는 각 인덱스에 하나의 vacuum 작업자 프로세스가 할당된 수동 VACUUM을 사용하여 기본적으로 병렬로 vacuum 및 정리가 가능합니다. 그러나 vacuum 작업이 병렬 실행에 적합한지 PostgreSQL이 확인하려면 다음과 같은 특정 기준을 충족해야 합니다.

- 인덱스가 최소 2개 있어야 합니다.
- `max_parallel_maintenance_workers` 파라미터 값을 최소 2로 설정합니다.

- 인덱스 크기는 `min_parallel_index_scan_size` 한도를 초과해야 하며, 기본값은 512KB입니다.

Amazon RDS 인스턴스에서 사용할 수 vCPU의 수와 테이블의 인덱스 수에 따라 `max_parallel_maintenance_workers` 설정을 조정하여 vacuum 작업 처리 시간을 최적화할 수 있습니다.

자세한 내용은 [Parallel vacuuming in Amazon RDS for PostgreSQL and Amazon Aurora PostgreSQL](https://aws.amazon.com/)을 참조하세요.

테이블 또는 데이터베이스가 너무 많아 vacuum 작업을 할 수 없음

PostgreSQL의 The Autovacuum Daemon 설명서에서 언급한 것처럼 autovacuum 대몬은 여러 프로세스를 통해 작동합니다. 여기에는 시스템 내 각 데이터베이스에 대해 autovacuum 작업자 프로세스를 시작하는 영구 autovacuum 런처가 포함됩니다. 런처는 데이터베이스당 약 `autovacuum_naptime`초마다 이러한 작업자가 시작하도록 일정을 예약합니다.

데이터베이스가 'N'개이면 새 작업자가 대략 [`autovacuum_naptime`/N초]마다 시작합니다. 그러나 총 동시 작업자 수는 `autovacuum_max_workers` 설정에 따라 제한됩니다. Vacuum 작업이 필요한 데이터베이스 또는 테이블 수가 이 한도를 초과하는 경우 작업자를 사용할 수 있게 되는 즉시 다음 데이터베이스 또는 테이블이 처리됩니다.

많은 대형 테이블 또는 데이터베이스에서 동시에 vacuum 작업이 필요한 경우 사용 가능한 모든 autovacuum 작업자가 장기간 점유되어 다른 테이블 및 데이터베이스에 대한 유지 관리가 지연될 수 있습니다. 트랜잭션 속도가 높은 환경에서는 이 병목 현상이 빠르게 에스컬레이션되어 Amazon RDS 인스턴스 내에서 vacuum 랩어라운드 문제가 발생할 수 있습니다.

`postgres_get_av_diag()`가 많은 수의 테이블 또는 데이터베이스를 감지하면 다음과 같은 권장 사항을 제공합니다.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

지침

autovacuum_max_workers 증가

Vacuum 작업의 속도를 높이기 위해 더 많은 동시 autovacuum 작업자를 허용하도록 `autovacuum_max_workers` 파라미터를 조정하는 것이 좋습니다. 성능 병목 현상이 지속되면 Amazon RDS 인스턴스를 더 많은 vCPU가 있는 클래스로 스케일 업하는 것이 좋습니다. 이렇게 하면 병렬 처리 기능을 더욱 개선할 수 있습니다.

공격적 vacuum(랩어라운드 방지용) 실행 중

PostgreSQL의 데이터베이스(MaximumUsedTransactionIDs) 연령은 공격적 vacuum(랩어라운드 방지용)이 성공적으로 완료될 때만 감소합니다. 이 vacuum 작업이 완료될 때까지 트랜잭션 속도에 따라 연령이 계속 증가합니다.

`postgres_get_av_diag()` 함수는 공격적인 vacuum을 감지하면 다음 `NOTICE`를 생성합니다. 그러나 vacuum의 활성 상태가 최소 2분간 지속된 후에만 이 출력이 트리거됩니다.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

공격적 vacuum에 대한 자세한 내용은 When an aggressive vacuum is already running 섹션을 참조하세요.

다음 쿼리를 사용하여 공격적인 vacuum이 진행 중인지 확인할 수 있습니다.

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
    a.wait_event,
    v.pid,
    v.phase,
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
```

```
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

출력의 쿼리 열을 확인하여 공격적인 vacuum(랩어라운드 방지용)인지 확인할 수 있습니다. '랩어라운드 방지용'이라는 문구는 공격적인 vacuum임을 나타냅니다.

```
query                  | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

예를 들어, 트랜잭션 연령이 10억인 블로커가 있고 동일한 트랜잭션 연령에서 랩어라운드 방지용으로 공격적인 vacuum이 필요한 테이블이 있다고 가정해 보겠습니다. 또한 트랜잭션 연령이 7억 5,000만인 또 다른 블로커가 있습니다. 트랜잭션 연령이 1억인 블로커를 정리한 후에 트랜잭션 연령이 7억 5,000만으로 즉시 줄어들지 않습니다. 공격적인 vacuum이 필요한 테이블 또는 연령이 7억 5,000만이 넘는 트랜잭션이 완료될 때까지 트랜잭션 연령이 높게 유지됩니다. 이 기간 동안 PostgreSQL 클러스터의 트랜잭션 연령이 계속 증가합니다. Vacuum 프로세스가 완료되면 트랜잭션 연령이 7억 5,000만으로 감소하지만 추가 vacuum 작업이 완료될 때까지 다시 증가하기 시작합니다. 이 주기는 이러한 조건이 지속되는 한, 트랜잭션 연령이 결국 `autovacuum_freeze_max_age`에서 지정된 Amazon RDS 인스턴스에 대해 구성된 수준으로 떨어질 때까지 계속됩니다.

Aurora PostgreSQL의 NOTICE 메시지 설명

`postgres_get_av_diag()` 함수는 다음 NOTICE 메시지를 제공합니다.

연령이 모니터링 임곗값에 아직 도달하지 않은 경우

> `postgres_get_av_diag()`가 블로커를 식별하기 위한 모니터링 임곗값은 기본적으로 5억 개의 트랜잭션입니다. `postgres_get_av_diag()`가 다음 NOTICE를 생성하면 트랜잭션 연령이 아직 이 임곗값에 도달하지 않았음을 나타냅니다.

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

연령이 가장 많은 트랜잭션 ID를 가진 데이터베이스에 연결되지 않음

`postgres_get_av_diag()` 함수는 연령이 가장 많은 트랜잭션 ID를 가진 데이터베이스에 연결할 때 가장 정확한 출력을 제공합니다. `postgres_get_av_diag()`에서 보고한 연령이 가장 많은 트랜잭션 ID를 가진 데이터베이스는 사용자의 경우 'my_database'와 다릅니다. 올바른 데이터베이스에 연결되지 않은 경우 다 NOTICE가 생성됩니다.

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

연령이 가장 많은 트랜잭션을 가진 데이터베이스에 연결하는 것은 다음과 같은 이유로 중요합니다.

- 임시 테이블 블로커 식별: 임시 테이블의 메타데이터는 각 데이터베이스마다 다르기 때문에 일반적으로 테이블이 만들어진 데이터베이스에서 찾을 수 있습니다. 그러나 임시 테이블이 최상위 블로커이고 수명이 가장 많은 트랜잭션을 가진 데이터베이스에 있는 경우 이는 오해의 소지가 있을 수 있습니다. 올바른 데이터베이스에 연결하면 임시 테이블 블로커를 정확하게 식별할 수 있습니다.
- 느린 vacuum 진단: 인덱스 메타데이터 및 테이블 수 정보는 데이터베이스별로 다르며 느린 vacuum 문제를 진단하는 데 필요합니다.

연령을 기준으로 가장 오래된 트랜잭션을 가진 데이터베이스가 rdsadmin 또는 template0 데이터베이스임

경우에 따라 `rdsadmin` 또는 `template0` 데이터베이스가 연령이 가장 많은 트랜잭션 ID를 가진 데이터베이스로 식별될 수 있습니다. 이 경우 `postgres_get_av_diag()`는 다음 NOTICE를 발행합니다.

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

나열된 블로커가 이 두 데이터베이스 중 하나에서 시작되지 않는지 확인합니다. `rdsadmin` 또는 `template0`에 블로커가 있는 것으로 보고되는 경우 이러한 데이터베이스는 사용자가 액세스할 수 없으며 개입이 필요하므로 지원팀에 문의하세요.

`rdsadmin` 또는 `template0` 데이터베이스가 최상위 블로커를 포함할 가능성은 매우 낮습니다.

공격적 vacuum이 이미 실행 중인 경우

postgres_get_av_diag() 함수는 공격적인 vacuum 프로세스가 실행 중일 때 보고하도록 설계되었지만, vacuum이 최소 1분 동안 활성 상태인 경우에만 이 출력을 트리거합니다. 이러한 의도적인 지연은 오탐지의 가능성을 줄이는 데 도움이 됩니다. 이 함수는 대기함으로써 효과적이고 중요한 vacuum만 보고되도록 하므로 vacuum 활동을 보다 정확하고 안정적으로 모니터링할 수 있습니다.

postgres_get_av_diag() 함수는 진행 중인 공격적인 vacuum이 하나 이상인 것을 감지하면 다음 NOTICE를 생성합니다.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

NOTICE에 표시된 대로 vacuum의 성능을 계속 모니터링합니다. 공격적 vacuum에 대한 자세한 내용은 [공격적 vacuum(랩어라운드 방지용) 실행 중]() 섹션을 참조하세요.

Autovacuum이 꺼져 있는 경우

데이터베이스 인스턴스에서 autovacuum이 비활성화된 경우 postgres_get_av_diag() 함수는 다음 NOTICE를 생성합니다.

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

자동 정리는 원활한 데이터베이스 작업을 보장하는 Aurora PostgreSQL DB 인스턴스의 중요한 기능입니다. 오래된 행 버전을 자동으로 제거하고, 스토리지 공간을 회수하고, 테이블 팽창을 방지하여 테이블과 인덱스를 효율적으로 유지함으로써 최적의 성능을 발휘하도록 합니다. 또한 Amazon RDS 인스턴스에서 트랜잭션을 중지할 수 있는 트랜잭션 ID 랩어라운드를 방지합니다. Autovacuum을 비활성화하면 데이터베이스 성능과 안정성이 장기적으로 저하될 수 있습니다. 항상 켜두는 것이 좋습니다. 자세한 내용은 [Understanding autovacuum in Aurora PostgreSQL environments]()를 참조하세요.

> **Note**
>
> Autovacuum을 끄더라도 공격적인 vacuum이 중지되지 않습니다. 테이블이 autovacuum_freeze_max_age 임곗값에 도달하면 여전히 공격적 vacuum이 발생합니다.

남은 트랜잭션 수가 매우 적음

`postgres_get_av_diag()` 함수는 랩어라운드 vacuum이 임박하면 다음 NOTICE를 생성합니다. 이 NOTICE는 Amazon RDS 인스턴스가 잠재적으로 새 트랜잭션을 거부하기까지 1억 개의 트랜잭션이 남았을 때 발행됩니다.

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

데이터베이스 가동 중지 시간을 방지하려면 즉각적인 조치가 필요합니다. Vacuum 작업을 면밀히 모니터링하고 영향을 받는 데이터베이스에서 `VACUUM FREEZE`를 수동으로 시작하여 트랜잭션 실패를 방지해야 합니다.

Babelfish for Aurora PostgreSQL 사용

Babelfish for Aurora PostgreSQL에서는 SQL Server 클라이언트의 데이터베이스 연결을 수락하는 기능을 사용하여 Aurora PostgreSQL DB 클러스터를 확장합니다. Babelfish를 사용하면 원래 SQL Server용으로 구축된 애플리케이션이 기존 마이그레이션에 비해 코드를 거의 변경할 필요가 없고 데이터베이스 드라이버를 수정하지 않고도 Aurora PostgreSQL과 직접 작동할 수 있습니다. 마이그레이션에 대한 자세한 내용은 [SQL Server 데이터베이스를 Babelfish for Aurora PostgreSQL로 마이그레이션](#) 섹션을 참조하세요.

Babelfish는 Aurora PostgreSQL 데이터베이스 클러스터에 대한 추가 엔드포인트를 제공하여 SQL Server 와이어 수준 프로토콜 및 일반적으로 사용되는 SQL Server 문을 이해하도록 할 수 있습니다. Tabular Data Stream(TDS) 와이어 프로토콜을 사용하는 클라이언트 애플리케이션은 Aurora PostgreSQL의 TDS 리스너 포트에 기본적으로 연결할 수 있습니다. TDS에 대한 자세한 내용은 Microsoft 웹 사이트의 [MS-TDS]: Tabular Data Stream 프로토콜을 참조하세요.

> **Note**
>
> Babelfish for Aurora PostgreSQL은 TDS 버전 7.1에서 7.4까지를 지원합니다.

또한 Babelfish는 PostgreSQL 연결을 사용하여 데이터에 대한 액세스를 제공합니다. 기본적으로 Babelfish에서 지원하는 두 SQL 언어는 다음 포트에서 기본 와이어 프로토콜을 통해 사용할 수 있습니다.

- SQL Server 언어(T-SQL), 클라이언트는 포트 1433에 연결합니다.
- PostgreSQL 언어(PL/pgSQL), 클라이언트는 포트 5432에 연결합니다.

Babelfish는 몇 가지 차이가 있는 Transact-SQL(T-SQL) 언어를 실행합니다. 자세한 내용은 Babelfish for Aurora PostgreSQL과 SQL Server의 차이점 섹션을 참조하세요.

다음 섹션에서는 Babelfish for Aurora PostgreSQL DB 클러스터를 설정하고 사용하는 방법에 대한 정보를 찾아볼 수 있습니다.

주제

Babelfish 제한 사항

현재 Babelfish for Aurora PostgreSQL에 다음과 같은 제한 사항이 적용됩니다.

- 업그레이드할 때 동일한 스키마 내에서 객체 소유권이 다를 수도 있습니다. 업그레이드 전 객체는 기존 소유자가 소유하지만, 업그레이드 후 객체는 스키마 소유자가 소유하며, 이 소유자는 다른 사용자일 수 있습니다. 이를 해결하기 위해 `sys.generate_alter_ownership_statements()` 함수를 도입했습니다.

  객체 소유권 불일치를 해결하려면 PostgreSQL 엔드포인트를 사용하여 클러스터에 연결하고, sys.generate_alter_ownership_statements() 함수를 실행하고, 생성된 SQL 문을 실행합니다.

  객체 소유권 변경과 관련된 다음과 같은 중요한 제한 사항을 고려하세요.

  - PostgreSQL 엔드포인트를 통해 CREATE 권한을 부여받은 사용자는 TDS 엔드포인트를 통해 객체를 생성할 수 없습니다. PostgreSQL 엔드포인트를 통해 T-SQL 객체에 대한 권한을 변경하지 않는 것이 좋습니다. 잘못된 T-SQL 동작이 유발될 수 있기 때문입니다.
  - 객체 액세스 권한이 변경될 수 있습니다. 예를 들어 'sch_own'이 소유한 스키마에 업그레이드 전에 'dbo'가 소유했던 객체가 포함된 경우, 사용자가 SELECT 또는 INSERT와 같은 작업을 수행할 수 있는지는 업그레이드 전 객체('dbo' 소유)와 업그레이드 후 객체('sch_own' 소유) 간에 다를 수 있습니다.
- 현재 Babelfish는 다음 Aurora 기능을 지원하지 않습니다.
  - AWS Identity and Access Management
  - 데이터베이스 활동 스트림(DAS)
  - Aurora PostgreSQL Serverless v2를 사용하고 프로비저닝된 RDS 데이터 API
  - RDS for SQL Server를 사용하는 RDS 프록시
  - Salted challenge response authentication mechanism(SCRAM)
  - 쿼리 편집기
- Babelfish는 다음과 같은 클라이언트 드라이버 API 지원을 제공하지 않습니다.
  - Microsoft Distributed Transaction Coordinator(MSDTC) 관련 연결 속성이 있는 API 요청은 지원되지 않습니다. 여기에는 SQL 서버 JDBC 드라이버의 SQLServerXAResource 클래스에 의한 XA 호출이 포함됩니다.
- Babelfish는 현재 다음 Aurora PostgreSQL 확장을 지원하지 않습니다.
  - `bloom`
  - `btree_gin`
  - `btree_gist`

  - citext
  - cube
  - hstore
  - hypopg
  - pglogical를 이용한 논리적 복제
  - ltree
  - pgcrypto
  - apg_plan_mgmt를 사용하는 쿼리 계획 관리

  PostgreSQL 확장에 대한 자세한 내용은 확장 및 외부 데이터 래퍼 작업 섹션을 참조하세요.
- Microsoft JDBC 드라이버의 대안으로 설계된 오픈 소스 jTDS 드라이버는 지원되지 않습니다.

Babelfish 아키텍처 및 구성 이해

Aurora DB 클러스터와 마찬가지로 Babelfish를 실행하는 Aurora PostgreSQL-Compatible Edition DB 클러스터를 관리합니다. 즉 확장성, 장애 조치를 지원하는 고가용성 및 Aurora DB 클러스터가 제공하는 기본 제공 복제본의 이점을 누릴 수 있습니다. 이러한 기능에 대해 자세히 알아보려면 Aurora DB 클러스터의 성능 및 확장 관리, Amazon Aurora의 고가용성 및 Amazon Aurora를 사용한 복제 섹션을 참조하세요. 다음과 같이 다른 많은 AWS 도구 및 유틸리티에 대한 액세스 권한도 있습니다.

- Amazon CloudWatch는 데이터 및 실행 가능한 인사이트를 제공하는 모니터링 및 관찰 가능 서비스입니다. 자세한 내용은 Amazon CloudWatch로 Amazon Aurora 지표 모니터링 단원을 참조하십시오.
- 성능 개선 도우미는 데이터베이스의 로드를 빠르게 평가할 수 있는 데이터베이스 성능 튜닝 및 모니터링 기능입니다. 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링을 참조하십시오.
- Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있으므로 대기 시간이 짧은 글로벌 읽기를 지원하며, 드물게 발생하여 전체 AWS 리전에 영향을 미칠 수 있는 중단을 신속하게 복구할 수 있습니다. 자세한 내용은 Amazon Aurora Global Database 사용 단원을 참조하십시오.
- 사용 가능한 경우 자동 소프트웨어 패치를 사용하면 최신 보안 및 기능 패치로 데이터베이스를 최신 상태로 유지할 수 있습니다.
- Amazon RDS 이벤트는 이메일 또는 SMS 메시지를 통해 사용자에게 자동 장애 조치와 같은 중요한 데이터베이스 이벤트에 대한 알림을 제공합니다. 자세한 내용은 Amazon Aurora 이벤트 모니터링 단원을 참조하십시오.

아래에서 Babelfish 아키텍처와 마이그레이션하는 SQL Server 데이터베이스를 Babelfish에서 처리하는 방법에 대해 알아볼 수 있습니다. Babelfish DB 클러스터를 생성할 때 단일 데이터베이스 또는 다중 데이터베이스, 데이터 정렬 및 기타 세부 사항에 대해 몇 가지 결정을 내려야 합니다.

주제

Babelfish 아키텍처

Babelfish가 켜진 상태에서 Aurora PostgreSQL 클러스터를 생성하면 Aurora는 `babelfish_db`라는 PostgreSQL 데이터베이스로 클러스터를 프로비저닝합니다. 이 데이터베이스에는 마이그레이션된 모든 SQL Server 객체 및 구조가 상주합니다.

> **Note**
>
> Aurora PostgreSQL 클러스터에서 `babelfish_db` 데이터베이스 이름은 Babelfish용으로 예약되어 있습니다. Babelfish DB 클러스터에 자체 'babelfish_db' 데이터베이스를 생성하면 Aurora가 Babelfish를 성공적으로 프로비저닝하지 못합니다.

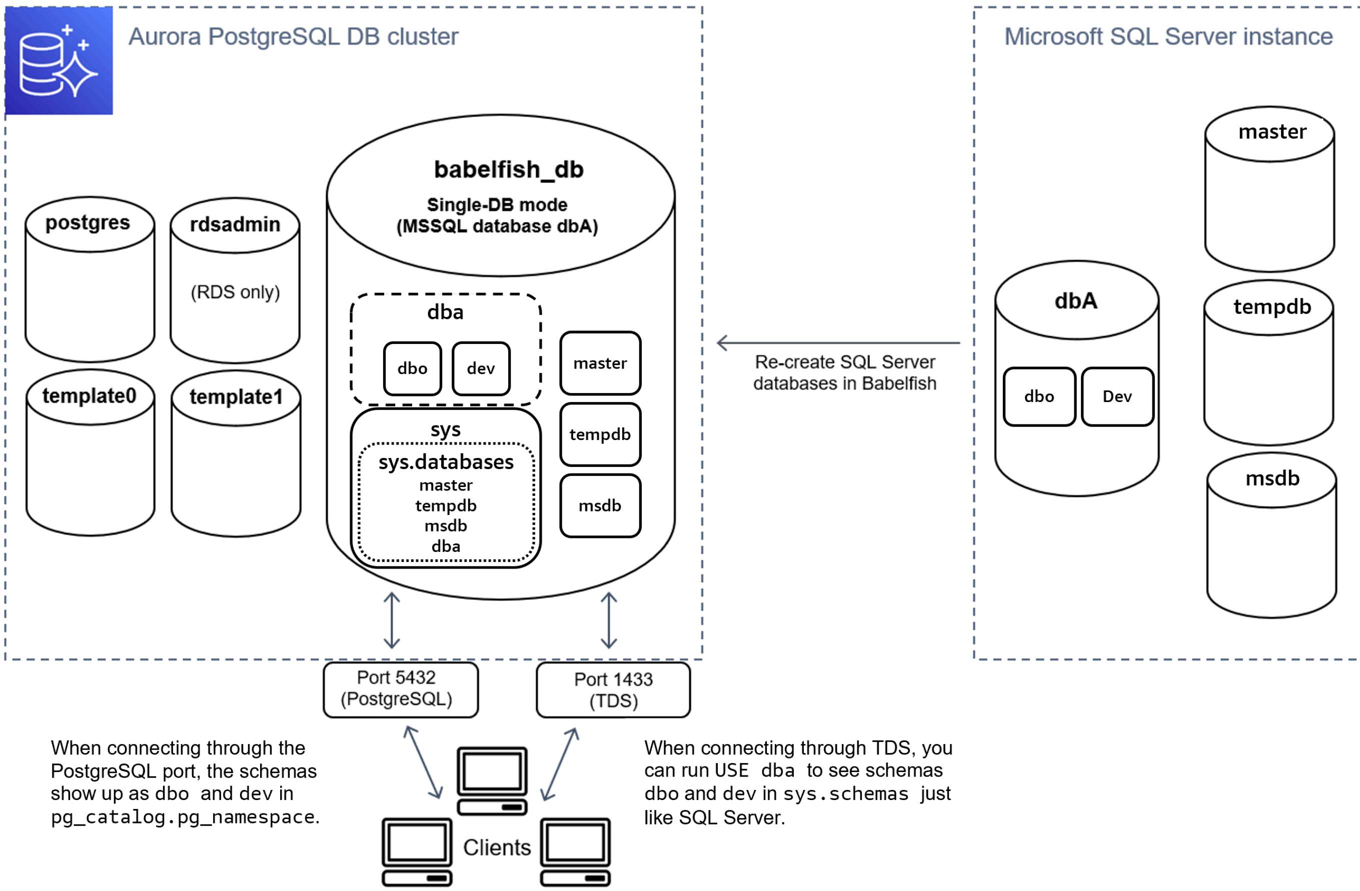
TDS 포트에 연결하면 세션이 `babelfish_db` 데이터베이스에 배치됩니다. T-SQL에서 구조는 SQL Server 인스턴스에 연결되는 것과 비슷합니다. `master`, `msdb`, `tempdb` 데이터베이스와 `sys.databases` 카탈로그가 표시됩니다. USE 문을 사용하여 추가 사용자 데이터베이스를 생성하고 데이터베이스 간에 전환할 수 있습니다. SQL Server 사용자 데이터베이스를 생성하면 `babelfish_db` PostgreSQL 데이터베이스에 병합됩니다. 데이터베이스는 SQL Server에서 제공하는 것과 같거나 유사한 교차 데이터베이스 구문 및 의미 체계를 유지합니다.

단일 데이터베이스 또는 여러 데이터베이스에서 Babelfish 사용

Babelfish와 함께 사용할 Aurora PostgreSQL 클러스터를 생성하면 자체적으로 단일 SQL Server 데이터베이스를 사용하거나 여러 SQL Server 데이터베이스를 함께 사용할 것인지 선택할 수 있습니다. 선택 사항은 `babelfish_db` 데이터베이스 내부의 SQL Server 스키마의 이름이 Aurora PostgreSQL에서 나타나는 방식에 영향을 미칩니다. 마이그레이션 모드는 `migration_mode` 파라미터에 저장됩니

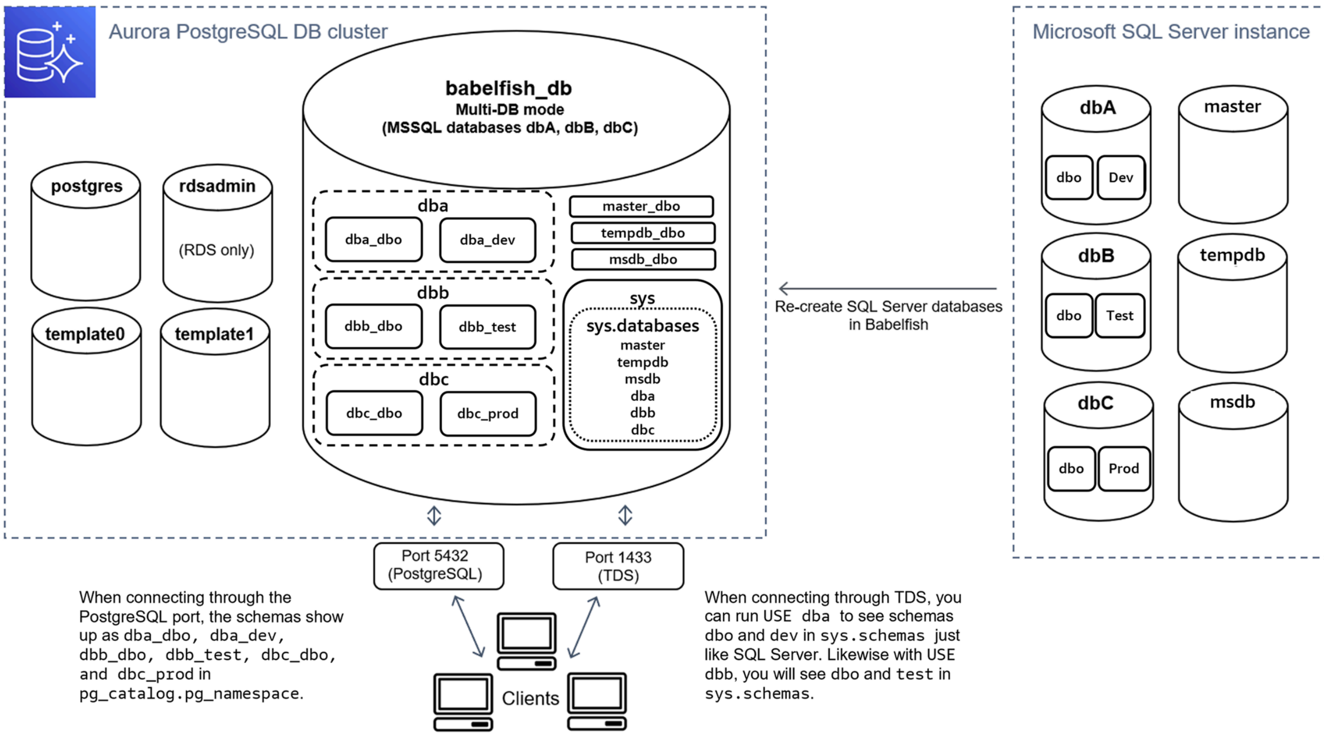
다. 이전에 생성한 모든 SQL 객체에 대한 액세스 권한이 손실될 수 있으므로, 클러스터를 생성한 후에는 이 파라미터를 변경하지 않아야 합니다.

단일 데이터베이스 모드에서는 SQL Server 데이터베이스의 스키마 이름이 PostgreSQL의 `babelfish_db` 데이터베이스에서도 그대로 유지됩니다. 단일 데이터베이스만 마이그레이션하도록 선택하면, 마이그레이션된 사용자 데이터베이스의 스키마 이름을 SQL Server에서 사용한 것과 동일한 이름으로 PostgreSQL에서 참조할 수 있습니다. 예를 들어 dbo 및 `smith` 스키마는 dbA 데이터베이스 내부에 상주합니다.



TDS를 통해 연결할 때 SQL 서버에서와 마찬가지로 T-SQL에서 dbo 및 `dev` 스키마를 보려면 USE dba를 실행할 수 있습니다. 변경되지 않은 스키마 이름은 PostgreSQL에서 볼 수 있습니다.

다중 데이터베이스 모드에서 사용자 데이터베이스의 스키마 이름은 PostgreSQL에서 액세스하는 경우 `dbname_schemaname`이 됩니다. T-SQL에서 액세스하는 경우에도 스키마 이름은 동일하게 유지됩니다.



이미지에서 볼 수 있듯이 다중 데이터베이스 모드와 단일 데이터베이스 모드는 TDS 포트를 통해 연결하고 T-SQL을 사용할 때 SQL Server와 동일합니다. 예를 들어, `USE dbA`은 SQL Server에서와 같이 스키마 `dbo` 및 `dev`를 나열합니다. 매핑된 스키마 이름(예: `dba_dbo` 및 `dba_dev`)은 PostgreSQL에서 볼 수 있습니다.

각 데이터베이스에는 여전히 스키마가 포함되어 있습니다. 각 데이터베이스의 이름이 SQL Server 스키마 이름 앞에 추가되며 밑줄을 구분 기호로 사용합니다. 예를 들면 다음과 같습니다.

- `dba`가 `dba_dbo` 및 `dba_dev`를 포함합니다.
- `dbb`가 `dbb_dbo` 및 `dbb_test`를 포함합니다.
- `dbc`가 `dbc_dbo` 및 `dbc_prod`를 포함합니다.

`babelfish_db` 데이터베이스 내부에서 T-SQL 사용자는 계속 `USE dbname`을 실행하여 데이터베이스 컨텍스트를 변경하므로 모양과 느낌이 SQL Server와 비슷하게 유지됩니다.

마이그레이션 모드 선택

각 마이그레이션 모드에는 장점과 단점이 있습니다. 보유한 사용자 데이터베이스 수와 마이그레이션 계획에 따라 마이그레이션 모드를 선택합니다. Babelfish와 함께 사용할 클러스터를 생성한 후에는 마이그레이션 모드를 변경하지 않아야 합니다. 이전에 생성한 모든 SQL 객체에 대한 액세스 권한을 잃을

수 있습니다. 마이그레이션 모드를 선택할 때는 사용자 데이터베이스 및 클라이언트의 요구 사항을 고려하세요.

Babelfish와 함께 사용할 클러스터를 만들면 Aurora PostgreSQL은 시스템 데이터베이스 `master` 및 `tempdb`를 생성합니다. 시스템 데이터베이스(`master` 또는 `tempdb`)에서 객체를 생성하거나 수정한 경우 새 클러스터에서 해당 객체를 다시 생성해야 합니다. SQL 서버와 달리 Babelfish는 클러스터 재부팅 후 `tempdb`를 다시 초기화하지 않습니다.

다음과 같은 경우 단일 데이터베이스 마이그레이션 모드를 사용합니다.

- 단일 SQL Server 데이터베이스를 마이그레이션하는 경우. 단일 데이터베이스 모드의 경우 PostgreSQL에서 액세스될 때 마이그레이션된 스키마 이름은 원래 SQL Server 스키마 이름과 동일합니다. 따라서 PostgreSQL 연결에서 실행되도록 기존 SQL 쿼리를 최적화하려는 경우 기존 SQL 쿼리에 대한 코드 변경이 줄어듭니다.
- 최종 목표가 기본 Aurora PostgreSQL로 완전히 마이그레이션하는 경우. 마이그레이션하기 전에 스키마를 단일 스키마(dbo)로 통합한 다음, 단일 클러스터로 마이그레이션하여 필요한 변경 사항을 줄입니다.

다음과 같은 경우 여러 데이터베이스 마이그레이션 모드를 사용합니다.

- 동일한 인스턴스에서 여러 사용자 데이터베이스를 사용하는 기본 SQL Server 경험을 원하는 경우
- 여러 사용자 데이터베이스를 함께 마이그레이션해야 하는 경우

Babelfish용 DB 클러스터 파라미터 그룹 설정

Aurora PostgreSQL DB 클러스터를 생성하고 Babelfish에서 설정(Turn on Babelfish)을 선택한 경우 새로 생성(Create new)을 선택하면 DB 클러스터 파라미터 그룹이 자동으로 생성됩니다. 이 DB 클러스터 파라미터 그룹은 설치를 위해 선택한 Aurora PostgreSQL 버전(예: Aurora PostgreSQL 버전 14)의 Aurora PostgreSQL DB 클러스터 파라미터 그룹을 기반으로 합니다. 다음과 같은 일반 패턴을 사용하여 이름을 지정합니다.

```
custom-aurora-postgresql14-babelfish-compat-3
```

클러스터 생성 프로세스 중에 다음 설정을 변경할 수 있지만, 일부 설정은 사용자 지정 파라미터 그룹에 저장되면 변경할 수 없으므로 신중하게 지정해야 합니다.

- 단일 데이터베이스 또는 다중 데이터베이스
- 기본 데이터 정렬 로캘
- 데이터 정렬 이름
- DB 파라미터 그룹

기존 Aurora PostgreSQL DB 클러스터 버전 13 이상의 파라미터 그룹을 사용하려면 그룹을 편집하고 `babelfish_status` 파라미터를 on으로 설정합니다. Aurora PostgreSQL 클러스터를 생성하기 전에 Babelfish 옵션을 지정합니다. 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#)를 참조하세요.

다음 파라미터는 Babelfish 기본 설정을 제어합니다. 설명(Description)에 달리 명시되지 않는 한 파라미터를 수정할 수 있습니다. 기본값이 설명에 포함됩니다. 파라미터에 사용할 수 있는 값을 보려면 다음을 수행합니다.

> **Note**
>
> 새 DB 파라미터 그룹을 DB 인스턴스와 연결하면 수정된 정적 파라미터 및 동적 파라미터는 DB 인스턴스가 재부팅된 후에만 적용됩니다. 그러나 DB 파라미터 그룹을 DB 인스턴스에 연결한 후 DB 파라미터 그룹에서 동적 파라미터를 수정하면 이러한 변경 사항이 재부팅 없이 즉시 적용됩니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 메뉴에서 파라미터 그룹(Parameter groups)을 선택합니다.

3. 목록에서 `default.aurora-postgresql14` DB 클러스터 파라미터 그룹을 선택합니다.
4. 검색 필드에 파라미터 이름을 입력합니다. 예를 들면, 검색 필드에 `babelfishpg_tsql.default_locale`을 입력하여 이 파라미터와 기본값 및 허용되는 설정을 표시합니다.

> **Note**
>
> Babelfish for Aurora PostgreSQL 글로벌 데이터베이스는 보조 리전에 다음 파라미터가 설정된 경우에만 보조 리전에서 작동합니다.

파라미터	설명	적용 유형	수정 가능
babelfishpg_tsql.apg_enable_correlated_scalar_transform	플래너가 Babelfish에서 상관관계가 있는 스칼라 하위 쿼리를 변환할 수 있습니다. (기본값: on)(허용: on, off)	dynamic	true
babelfishpg_tsql.apg_enable_subquery_cache	Babelfish에서 상관관계가 있는 스칼라 하위 쿼리에 캐시를 사용할 수 있도록 지원합니다. (기본값: on)(허용: on, off)	dynamic	true
babelfishpg_tds.tds_default_numeric_scale	엔진에서 지정하지 않은 경우 TDS 열 메타데이터에서 전송할 숫자 유형의 기본 배율을 설정합니다. (기본값: 8)(허용: 0~38)	dynamic	true
babelfishpg_tds.tds_default_numeric_precision	엔진에서 지정하지 않은 경우 TDS 열 메타데이터에서 전송할 숫자 유형의 기본 정밀도	dynamic	true

파라미터	설명	적용 유형	수정 가능
	를 설정하는 정수입니다. (기본값: 38)(허용: 1~38)		
babelfishpg_tds.tds_default_packet_size	SQL Server 클라이언트 연결을 위한 기본 패킷 크기를 설정하는 정수입니다. (기본값: 4096)(허용: 512~32767)	dynamic	true
babelfishpg_tds.tds_default_protocol_version	클라이언트 연결을 위한 기본 TDS 프로토콜 버전을 설정하는 정수입니다. (기본값: DEFAULT) (허용: TDSv7.0, TDSv7.1, TDSv7.1.1, TDSv7.2, TDSv7.3A, TDSv7.3B, TDSv7.4, DEFAULT)	dynamic	true
babelfishpg_tds.default_server_name	Babelfish 서버의 기본 이름을 식별하는 문자열입니다. (기본값: Microsoft SQL Server) (허용: null)	dynamic	true
babelfishpg_tds.tds_debug_log_level	TDS에서 로깅 수준을 설정하는 정수로, 0은 로깅을 끕니다. (기본값: 1)(허용: 0, 1, 2, 3)	dynamic	true

파라미터	설명	적용 유형	수정 가능
babelfishpg_tds.listen_addresses	TDS를 수신할 호스트 이름, IP 주소 또는 주소를 설정하는 문자열입니다. Babelfish DB 클러스터를 생성한 후에는 이 파라미터를 수정할 수 없습니다. (기본값: *)(허용: null)	–	false
babelfishpg_tds.port	SQL Server 구문의 요청에 사용되는 TCP 포트를 설정하는 정수입니다. (기본값: 1433)(허용: 1~65535)	정적	true
babelfishpg_tds.tds_ssl_encrypt	TDS 리스너 포트를 통과하는 데이터에 대해 암호화를 켜거나(0) 끄는(1) 부울입니다. 클라이언트 연결을 위해 SSL 사용법에 대한 자세한 내용은 Babelfish SSL 설정 및 클라이언트 연결 섹션을 참조하세요. (기본값: 0)(허용: 0, 1)	dynamic	true
babelfishpg_tds.tds_ssl_max_protocol_version	TDS 세션에 사용할 가장 높은 SSL/TLS 프로토콜 버전을 지정하는 문자열입니다. (기본값: 'TLSv1.2')(허용: 'TLSv1', 'TLSv1.1', 'TLSv1.2')	dynamic	true

파라미터	설명	적용 유형	수정 가능
babelfishpg_tds.tds_ssl_min_protocol_version	TDS 세션에 사용할 최소 SSL/TLS 프로토콜 버전을 지정하는 문자열입니다. (기본값: Aurora PostgreSQL 버전 16의 'TLSv1.2', Aurora PostgreSQL 버전 16 이전 버전의 경우 'TLSv1') (허용 가능: 'TLSv1', 'TLSv1.1', 'TLSv1.2')	dynamic	true
babelfishpg_tds.unix_socket_directories	TDS 서버 Unix 소켓 디렉터리를 식별하는 문자열입니다. Babelfish DB 클러스터를 생성한 후에는 이 파라미터를 수정할 수 없습니다. (기본값: /tmp)(허용: null)	–	false
babelfishpg_tds.unix_socket_group	TDS 서버 Unix 소켓 그룹을 식별하는 문자열입니다. Babelfish DB 클러스터를 생성한 후에는 이 파라미터를 수정할 수 없습니다. (기본값: rdsdb)(허용: null)	–	false

파라미터	설명	적용 유형	수정 가능
babelfishpg_tsql.default_locale	Babelfish 데이터 정렬에 사용되는 기본 로캘을 지정하는 문자열입니다. 기본 로캘은 로캘일 뿐이며 한정자는 포함하지 않습니다. Babelfish DB 클러스터를 프로비저닝할 때 이 파라미터를 설정합니다. DB 클러스터를 프로비저닝한 후 이 파라미터에 대한 변경은 무시됩니다. (기본값: en_US)(허용: 테이블 참조)	정적	true

파라미터	설명	적용 유형	수정 가능
babelfishpg_tsql.migration_mode	단일 또는 다중 사용자 데이터베이스에 대한 지원을 지정하는 수정할 수 없는 목록입니다. Babelfish DB 클러스터를 프로비저닝할 때 이 파라미터를 설정합니다. DB 클러스터를 프로비저닝한 후에는 이 파라미터의 값을 수정할 수 없습니다. (기본값: Aurora PostgreSQL 버전 16의 multi-db, Aurora PostgreSQL 버전 16 이전 버전의 경우 single-db) (허용: single-db, multi-db, null)	정적	true
babelfishpg_tsql.server_collation_name	서버 수준 작업에 사용되는 데이터 정렬의 이름을 지정하는 문자열입니다. Babelfish DB 클러스터를 프로비저닝할 때 이 파라미터를 설정합니다. DB 클러스터를 프로비저닝한 후에는 이 파라미터의 값을 수정하지 마세요. (기본값: bbf_unicode_general_ci_as)(허용: 테이블 참조)	정적	true

파라미터	설명	적용 유형	수정 가능
babelfishpg_tsql.version	@@VERSION 변수의 출력을 설정하는 문자열입니다. Aurora PostgreSQL DB 클러스터에 대해 이 값을 수정하지 마세요. (기본값: null)(허용: 기본값)	dynamic	true
rds.babelfish_status	Babelfish 기능의 상태를 설정하는 문자열입니다. 이 파라미터가 `datatypesonly` 로 설정된 경우 Babelfish는 꺼져 있지만 SQL Server 데이터 유형은 계속 사용할 수 있습니다. (기본값: off)(허용: on, off, datatypesonly)	정적	true
unix_socket_permissions	TDS 서버 Unix 소켓 권한을 설정하는 정수입니다. Babelfish DB 클러스터를 생성한 후에는 이 파라미터를 수정할 수 없습니다. (기본값: 0700)(허용: 0~511)	–	false

Babelfish SSL 설정 및 클라이언트 연결

Babelfish for Aurora PostgreSQL DB 클러스터에 대한 SSL/TLS 연결을 요구하려면 `rds.force_ssl` 파라미터를 사용합니다.

- SSL/TLS 연결을 요구하려면 `rds.force_ssl` 파라미터 값을 1(켜짐)로 설정합니다.
- SSL/TLS 연결 요구를 끄려면 `rds.force_ssl` 파라미터 값을 0(꺼짐)으로 설정합니다.

이 파라미터의 기본값은 Aurora PostgreSQL 버전에 따라 다릅니다.

- Aurora PostgreSQL 버전 17 이상: 기본값은 1(켜짐)입니다.
- Aurora PostgreSQL 버전 16 이상: 기본값은 0(꺼짐)입니다.

> **Note**
>
> Aurora PostgreSQL 버전 16 이하에서 버전 17 이상으로 메이저 버전 업그레이드를 수행하면 파라미터의 기본값이 0(꺼짐)에서 1(켜짐)로 변경됩니다. 이 변경으로 인해 SSL용으로 구성되지 않은 애플리케이션에 대한 연결이 실패할 수 있습니다. 이 파라미터를 0(꺼짐)으로 설정하여 이전 기본 동작으로 되돌릴 수 있습니다.

드라이버별 세부 정보는 Babelfish DB 클러스터에 연결 섹션을 참조하세요.

클라이언트가 TDS 포트(기본값 `1433`)에 연결하는 경우 Babelfish는 클라이언트 핸드셰이크 중에 전송된 보안 소켓 계층(SSL) 설정을 Babelfish SSL 파라미터 설정(`tds_ssl_encrypt`)과 비교합니다. 그런 다음, Babelfish는 연결이 허용되는지 여부를 결정합니다. 연결이 허용되면 파라미터 설정과 클라이언트가 제공하는 암호화 지원에 따라 암호화 동작이 적용되거나 적용되지 않습니다.

다음 테이블은 Babelfish가 각 조합에 대해 어떻게 동작하는지 보여줍니다.

클라이언트 SSL 설정	Babelfish SSL 설정	rds.force_ssl	연결이 허용되나요?	클라이언트에 반환된 값
ENCRYPT_ON	모두	임의	허용됨, 전체 연결 암호화	ENCRYPT_ON
ENCRYPT_OFF	tds_ssl_encrypt=1	임의	허용됨, 전체 연결 암호화	ENCRYPT_REQ

클라이언트 SSL 설정	Babelfish SSL 설정	rds.force_ssl	연결이 허용되나요?	클라이언트에 반환된 값
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=0	허용됨, 로그인 패킷 암호화	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=1	아니요. 연결 종료	ENCRYPT_OFF
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=0	예	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	임의	아니요. 연결 종료	ENCRYPT_REQ
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=1	아니요. 연결 종료	ENCRYPT_NOT_SUP
ENCRYPT_CLIENT_CERT	모두	임의	아니요. 연결 종료	지원되지 않음

Babelfish for Aurora PostgreSQL의 데이터 정렬 이해

Babelfish를 사용하여 Aurora PostgreSQL DB 클러스터를 생성하는 경우 데이터의 데이터 정렬을 선택하게 됩니다. 데이터 정렬은 주어진 문자 언어로 텍스트 또는 문자를 생성하는 정렬 순서와 비트 패턴을 지정합니다. 데이터 정렬에는 지정된 비트 패턴 세트에 대한 데이터를 비교하는 규칙이 포함됩니다. 데이터 정렬은 현지화와 관련이 있습니다. 서로 다른 로캘은 문자 매핑, 정렬 순서 등에 영향을 줍니다. 데이터 정렬 속성은 다양한 데이터 정렬의 이름에 반영됩니다. 속성에 대한 자세한 내용은 Babelfish collation attributes table 섹션을 참조하세요.

Babelfish는 SQL Server 데이터 정렬을 Babelfish에서 제공하는 유사한 데이터 정렬에 매핑합니다. Babelfish는 문화적으로 민감한 문자열 비교 및 정렬 순서를 사용하여 유니코드 데이터 정렬을 미리 정의합니다. 또한 Babelfish는 SQL Server DB에 있는 데이터 정렬을 가장 r근접하게 일치하는 Babelfish 데이터 정렬로 변환하는 방법을 제공합니다. 로캘별 데이터 정렬은 다양한 언어 및 지역에 대해 제공됩니다.

일부 데이터 정렬은 클라이언트 측 인코딩에 해당하는 코드 페이지를 지정합니다. Babelfish는 각 출력 열의 데이터 정렬에 따라 서버 인코딩에서 클라이언트 인코딩으로 자동 변환됩니다.

Babelfish는 Babelfish supported collations table에 나열된 데이터 정렬을 지원합니다. Babelfish는 SQL Server 데이터 정렬을 Babelfish에서 제공하는 유사한 데이터 정렬에 매핑합니다.

Babelfish는 International Components for Unicode(ICU) 데이터 정렬 라이브러리 버전 153.80을 사용합니다. ICU 데이터 정렬의 자세한 내용은 ICU 설명서의 데이터 정렬을 참조하세요. PostgreSQL 및 데이터 정렬에 대한 자세한 내용은 PostgreSQL 설명서의 Collation Support(데이터 정렬 지원)을 참조하세요.

주제

데이터 정렬 및 로캘을 제어하는 DB 클러스터 파라미터

다음 파라미터는 데이터 정렬 동작에 영향을 줍니다.

babelfishpg_tsql.default_locale

이 파라미터는 데이터 정렬에 사용되는 기본 로캘을 지정합니다. 또한, Babelfish collation attributes table에 나열된 속성과 함께 사용되어 특정 언어 및 지역에 대한 데이터 정렬을 사용자 지정합니다. 이 파라미터의 기본값은 `en-US`입니다.

기본 로캘은 BBF로 시작하는 모든 Babelfish 데이터 정렬 이름과 Babelfish 데이터 정렬에 매핑되는 모든 SQL Server 데이터 정렬에 적용됩니다. 기존 Babelfish DB 클러스터의 이 파라미터에 대한 설정을 변경해도 기존 데이터 정렬의 로캘에는 영향을 주지 않습니다. 데이터 정렬 목록은 Babelfish supported collations table 섹션을 참조하세요.

babelfishpg_tsql.server_collation_name

이 파라미터는 서버(Aurora PostgreSQL DB 클러스터 인스턴스)와 데이터베이스의 기본 데이터 정렬을 지정합니다. 기본값은 `sql_latin1_general_cp1_ci_as`입니다. T-SQL에서 서버 데이터 정렬이 식별자를 비교하는 방법을 결정하므로 `server_collation_name`은 `CI_AS` 데이터 정렬이어야 합니다.

Babelfish DB 클러스터를 생성할 때 선택 가능한 목록에서 데이터 정렬 이름(Collation name)을 선택합니다. 여기에는 Babelfish supported collations table에 나열된 데이터 정렬이 포함됩니다. Babelfish 데이터베이스가 생성된 후 `server_collation_name`을 수정하지 마세요.

Babelfish for Aurora PostgreSQL DB 클러스터를 생성할 때 선택하는 설정은 이러한 파라미터의 클러스터와 연결된 DB 클러스터 파라미터 그룹에 저장되고 데이터 정렬 동작을 설정합니다.

Babelfish의 결정적 및 비결정적 데이터 정렬

Babelfish는 결정적이고 비결정적인 데이터 정렬을 지원합니다.

- 결정적 데이터 정렬은 동일한 바이트 시퀀스를 가진 문자를 동등하게 평가합니다. 이는 x 및 X가 결정적 데이터 정렬에서는 동일하지 않다는 뜻입니다. 결정적 데이터 정렬에는 대소문자 구분(CS) 및 악센트 구분(AS)이 적용됩니다.
- 비결정적 데이터 정렬은 똑같이 일치할 필요가 없습니다. 비결정적 데이터 정렬은 x 및 X를 동일한 것으로 확인합니다. 비결정적 데이터 정렬은 대소문자 구분하지 않음(CI)이나 악센트 구분하지 않음(AI) 또는 둘 다가 적용됩니다.

다음 테이블에서 비결정적 데이터 정렬을 사용할 때 Babelfish와 PostgreSQL 간의 동작 차이점을 확인할 수 있습니다.

Babelfish	PostgreSQL
CI_AS 데이터 정렬에 대한 LIKE 절을 지원합니다.	비결정적 데이터 정렬에서는 LIKE 절을 지원하지 않습니다.
Babelfish 버전 4.2.0의 다음 AI 데이터 정렬에서만 LIKE 절을 지원합니다. • bbf_unicode_cp1250_ci_ai • bbf_unicode_cp1250_cs_ai • bbf_unicode_cp1257_ci_ai • bbf_unicode_cp1257_cs_ai • bbf_unicode_cp1_ci_ai • bbf_unicode_cp1_cs_ai • estonian_ci_ai • finnish_swedish_ci_ai • french_ci_ai • latin1_general_ci_ai • latin1_general_cs_ai • modern_spanish_ci_ai • polish_ci_ai • sql_latin1_general_cp1_ci_ai • sql_latin1_general_cp1_cs_ai • traditional_spanish_ci_ai	비결정적 데이터 정렬에서는 LIKE 절을 지원하지 않습니다.

SQL Server 및 PostgreSQL과 비교하여 Babelfish의 기타 제한 사항 및 동작 차이점에 대한 목록은 데이터 정렬 제한 및 동작 차이점 섹션을 참조하세요.

Babelfish와 SQL Server는 다음 테이블과 같이 데이터 정렬 속성을 설명하는 데이터 정렬에 대한 명명 규칙을 따릅니다.

속성	설명
AI	악센트 구분하지 않음.
AS	악센트 구분.
BIN2	BIN2는 코드 포인트 순서대로 데이터를 정렬하도록 요청합니다. 유니코드 코드 포인트 순서는 UTF-8, UTF-16 및 UCS-2 인코딩의 문자 순서와 동일합니다. 코드 포인트 순서는 빠른 결정적 데이터 정렬입니다.
CI	대소문자 구분하지 않음.
CS	대소문자 구분.
PREF	소문자 앞에 대문자를 정렬하려면 PREF 데이터 정렬을 사용합니다. 비교가 대소문자를 구분하지 않는 경우 다른 구분이 없으면 대문자 버전은 소문자 버전 앞에 정렬됩니다. ICU 라이브러리는 `colCaseFirst=upper` 의 대문자 기본 설정을 지원하지만 CI_AS 데이터 정렬은 지원하지 않습니다. PREF는 CS_AS 결정적 데이터 정렬에만 적용될 수 있습니다.

Babelfish의 데이터베이스 수준에서 지원되는 데이터 정렬

다음 데이터 정렬은 Babelfish의 데이터베이스 수준에서 지원됩니다.

- bbf_unicode_bin2
- bbf_unicode_cp1_ci_ai
- bbf_unicode_cp1_ci_as
- bbf_unicode_cp1250_ci_ai
- bbf_unicode_cp1250_ci_as
- bbf_unicode_cp1257_ci_ai
- bbf_unicode_cp1257_ci_as
- estonian_ci_ai
- estonian_ci_as
- finnish_swedish_ci_ai
- finnish_swedish_ci_as

- french_ci_ai
- french_ci_as
- latin1_general_bin2
- latin1_general_ci_ai
- latin1_general_ci_as
- latin1_general_90_bin2
- latin1_general_100_bin2
- latin1_general_140_bin2
- modern_spanish_ci_ai
- modern_spanish_ci_as
- polish_ci_ai
- polish_ci_as
- sql_latin1_general_cp1_ci_ai
- sql_latin1_general_cp1_ci_as
- sql_latin1_general_cp1250_ci_as
- sql_latin1_general_cp1251_ci_as
- sql_latin1_general_cp1257_ci_as
- traditional_spanish_ci_ai
- traditional_spanish_ci_as

> **Note**
>
> 데이터베이스 수준에서 다른 데이터 정렬을 사용하려면 서버 수준 데이터 정렬과 일치하는지 확인합니다. 자세한 내용은 Babelfish의 서버 및 객체 데이터 정렬 단원을 참조하세요.

Babelfish의 서버 및 객체 데이터 정렬

다음 데이터 정렬을 서버 데이터 정렬 또는 객체 데이터 정렬로 사용합니다.

데이터 정렬 ID	참고

데이터 정렬 ID	참고
bbf_unicode_general_ci_as	대소문자를 구분하지 않는 비교와 LIKE 연산자를 지원합니다.
bbf_unicode_cp1_ci_as	비결정적 데이터 정렬은 CP1252라고도 합니다.
bbf_unicode_CP1250_ci_as	비결정적 데이터 정렬은 라틴 문자를 사용하는 중부 유럽 및 동유럽 언어로 텍스트를 표현하는 데 사용됩니다.
bbf_unicode_CP1251_ci_as	비결정적 데이터 정렬은 키릴 문자를 사용하는 언어에 사용됩니다.
bbf_unicode_cp1253_ci_as	비결정적 데이터 정렬은 현대 그리스어를 표현하는 데 사용됩니다.
bbf_unicode_cp1254_ci_as	비결정적 데이터 정렬은 터키어를 지원합니다.
bbf_unicode_cp1255_ci_as	비결정적 데이터 정렬은 히브리어를 지원합니다.
bbf_unicode_cp1256_ci_as	비결정적 데이터 정렬은 아랍 문자를 사용하는 언어를 작성하는 데 사용됩니다.
bbf_unicode_cp1257_ci_as	비결정적 데이터 정렬은 에스토니아어, 라트비아어 및 리투아니아어를 지원하는 데 사용됩니다.
bbf_unicode_cp1258_ci_as	비결정적 데이터 정렬은 베트남 문자를 작성하는 데 사용됩니다.
bbf_unicode_cp874_ci_as	비결정적 데이터 정렬은 태국 문자를 작성하는 데 사용됩니다.
sql_latin1_general_cp1250_ci_as	비결정적 싱글바이트 문자 인코딩은 라틴 문자를 표현하는 데 사용됩니다.

데이터 정렬 ID	참고
sql_latin1_general_cp1251_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1253_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1254_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1255_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1256_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1257_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
sql_latin1_general_cp1258_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
chinese_prc_ci_as	비결정적 데이터 정렬은 중국어(PRC)를 지원합니다.
cyrillic_general_ci_as	비결정적 데이터 정렬은 키릴어를 지원합니다.
finnish_swedish_ci_as	비결정적 데이터 정렬은 핀란드어를 지원합니다.

데이터 정렬 ID	참고
french_ci_as	비결정적 데이터 정렬은 프랑스어를 지원합니다.
japanese_ci_as	비결정적 데이터 정렬은 일본어를 지원합니다. Babelfish 2.1.0 이상 릴리스에서 지원됩니다.
korean_wansung_ci_as	비결정적 데이터 정렬은 한국어를 지원합니다(사전 정렬 사용).
latin1_general_ci_as	비결정적 데이터 정렬은 라틴 문자를 지원합니다.
modern_spanish_ci_as	비결정적 데이터 정렬은 현대 스페인어를 지원합니다.
polish_ci_as	비결정적 데이터 정렬은 폴란드어를 지원합니다.
thai_ci_as	비결정적 데이터 정렬은 태국어를 지원합니다.
traditional_spanish_ci_as	비결정적 데이터 정렬은 스페인어(전통에 따라 정렬)를 지원합니다.
turkish_ci_as	비결정적 데이터 정렬은 터키어를 지원합니다.
ukrainian_ci_as	비결정적 데이터 정렬은 우크라이나어를 지원합니다.
vietnamese_ci_as	비결정적 데이터 정렬은 베트남어를 지원합니다.

다음 데이터 정렬을 객체 데이터 정렬로 사용할 수 있습니다.

언어	결정적 옵션	비결정적 옵션
아랍어	Arabic_CS_AS	Arabic_CI_AS

언어	결정적 옵션	비결정적 옵션
		Arabic_CI_AI
아랍어 문자	BBF_Unicode_CP1256_CS_AS BBF_Unicode_Pref_CP1256_CS_AS	BBF_Unicode_CP1256_CI_AI BBF_Unicode_CP1256_CS_AI
바이너리	latin1_general_bin2 BBF_Unicode_BIN2	–
라틴 문자를 사용하는 중부 유럽 및 동유럽 언어	BBF_Unicode_CP1250_CS_AS BBF_Unicode_Pref_CP1250_CS_AS	BBF_Unicode_CP1250_CI_AI BBF_Unicode_CP1250_CS_AI
중국어	Chinese_PRC_CS_AS	Chinese_PRC_CI_AS Chinese_PRC_CI_AI
키릴 문자_일반	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS Cyrillic_General_CI_AI
키릴 문자	BBF_Unicode_CP1251_CS_AS BBF_Unicode_Pref_CP1251_CS_AS	BBF_Unicode_CP1251_CI_AI BBF_Unicode_CP1251_CS_AI
에스토니아어	Estonian_CS_AS	Estonian_CI_AS Estonian_CI_AI
에스토니아어, 라트비아어, 리투아니아어	BBF_Unicode_CP1257_CS_AS BBF_Unicode_Pref_CP1257_CS_AS	BBF_Unicode_CP1257_CI_AI BBF_Unicode_CP1257_CS_AI

언어	결정적 옵션	비결정적 옵션
필란드어_스웨덴어	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS Finnish_Swedish_CI_AI
프랑스어	French_CS_AS	French_CI_AS French_CI_AI
그리스어	Greek_CS_AS	Greek_CI_AS Greek_CI_AI
히브리어	BBF_Unicode_CP1255_CS_AS BBF_Unicode_Pref_CP1255_CS_AS Hebrew_CS_AS	BBF_Unicode_CP1255_CI_AI BBF_Unicode_CP1255_CS_AI Hebrew_CI_AS Hebrew_CI_AI
일본어(Babelfish 2.1.0 이상)	Japanese_CS_AS	Japanese_CI_AI Japanese_CI_AS
한국어_완성형	Korean_Wamsung_CS_AS	Korean_Wamsung_CI_AS Korean_Wamsung_CI_AI

언어	결정적 옵션	비결정적 옵션
코드 페이지 CP1252의 라틴 문자	latin1_general_cs_as BBF_Unicode_General_CS_AS BBF_Unicode_General_Pref_CS_AS BBF_Unicode_Pref_CP1_CS_AS BBF_Unicode_CP1_CS_AS	latin1_general_ci_as latin1_general_ci_ai latin1_general_cs_ai BBF_Unicode_General_CI_AI BBF_Unicode_General_CS_AI BBF_Unicode_CP1_CI_AI BBF_Unicode_CP1_CS_AI
현대 그리스어	BBF_Unicode_CP1253_CS_AS BBF_Unicode_Pref_CP1253_CS_AS	BBF_Unicode_CP1253_CI_AI BBF_Unicode_CP1253_CS_AI
현대_스페인어	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS Modern_Spanish_CI_AI
몽골어	Mongolian_CS_AS	Mongolian_CI_AS Mongolian_CI_AI
폴란드어	Polish_CS_AS	Polish_CI_AS Polish_CI_AI
태국어	BBF_Unicode_CP874_CS_AS BBF_Unicode_Pref_CP874_CS_AS Thai_CS_AS	BBF_Unicode_CP874_CI_AI BBF_Unicode_CP874_CS_AI Thai_CI_AS, Thai_CI_AI

언어	결정적 옵션	비결정적 옵션
정통_스페인어	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS Traditional_Spanish_CI_AI
터키어	BBF_Unicode_CP1254_CS_AS BBF_Unicode_Pref_CP1254_CS_AS Turkish_CS_AS	BBF_Unicode_CP1254_CI_AI BBF_Unicode_CP1254_CS_AI Turkish_CI_AS, Turkish_CI_AI
우크라이나어	Ukranian_CS_AS	Ukranian_CI_AS Ukranian_CI_AI
베트남어	BBF_Unicode_CP1258_CS_AS BBF_Unicode_Pref_CP1258_CS_AS Vietnamese_CS_AS	BBF_Unicode_CP1258_CI_AI BBF_Unicode_CP1258_CS_AI Vietnamese_CI_AS Vietnamese_CI_AI

Babelfish의 기본 데이터 정렬 동작

이전에는 `pg_catalog.default`가 정렬 가능한 데이터 형식의 기본 데이터 정렬이었습니다. 이러한 데이터 형식에 의존하는 데이터 형식 및 객체는 대/소문자 구분 데이터 정렬을 따릅니다. 이 조건은 대소문자를 구분하지 않는 데이터 정렬을 따르는 데이터 세트의 T-SQL 객체에 잠재적으로 영향을 미칩니다. Babelfish 2.3.0부터 정렬 가능한 데이터 형식(TEXT 및 NTEXT 제외)의 기본 데이터 정렬은 `babelfishpg_tsql.server_collation_name` 파라미터의 데이터 정렬과 동일합니다. Babelfish 2.3.0으로 업그레이드하면 DB 클러스터 생성 시 기본 데이터 정렬이 자동으로 선택되므로 눈에 띄는 영향이 없습니다.

데이터 정렬 관리

ICU 라이브러리는 데이터 정렬 버전 추적을 제공하여 새 버전의 ICU를 사용할 수 있으면 데이터 정렬에 종속된 인덱스를 다시 인덱싱할 수 있도록 합니다. 현재 데이터베이스에 새로 고침이 필요한 데이터 정렬이 있는지 확인하려면 `psql` 또는 `pgAdmin`을 사용하여 연결한 후 다음 쿼리를 활용합니다.

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

이 쿼리는 다음과 같은 출력을 반환합니다.

```
 Collation | Object
-----------+--------
(0 rows)
```

이 예에서는 데이터 정렬을 업데이트할 필요가 없습니다.

Babelfish 데이터베이스에서 미리 정의된 데이터 정렬 목록을 확인하려면 다음 쿼리와 함께 `psql` 또는 `pgAdmin`을 사용하면 됩니다.

```
SELECT * FROM pg_collation;
```

미리 정의된 데이터 정렬은 `sys.fn_helpcollations` 테이블에 저장됩니다. 다음 명령을 사용하여 데이터 정렬에 대한 정보(예: lcid, style 및 collate flags)를 표시할 수 있습니다. `sqlcmd`를 사용하여 모든 데이터 정렬 목록을 가져오려면 T-SQL 포트(기본값 1433)에 연결하고 다음 쿼리를 실행합니다.

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                     description
---------------------------------------- ----------------------------------------
arabic_cs_as                             Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                             Arabic, case-insensitive, accent-insensi
arabic_ci_as                             Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                         Unicode-General, case-sensitive, accent-
```

```
bbf_unicode_cp1250_ci_ai                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai                Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as                Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as           Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai                Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as                Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai                Default locale, code page 1254, case-ins
...
(124 rows affected)
```

이 예에 표시된 1행과 2행은 문서 가독성을 위해 출력 범위를 좁혔습니다.

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
---------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

데이터 정렬 제한 및 동작 차이점

Babelfish는 데이터 정렬 지원을 위해 ICU 라이브러리를 사용합니다. PostgreSQL은 특정 버전의 ICU로 빌드되며 최대 한 버전의 데이터 정렬을 일치시킬 수 있습니다. 시간이 흐름에 따라 언어가 발전하면서 사소한 변형이 있는 것처럼 버전 간 변형은 피할 수 없습니다. 다음 목록에는 Babelfish 데이터 정렬의 알려진 제한 사항 및 동작 변형이 나와 있습니다.

- 인덱스 및 데이터 정렬 유형 종속성 – International Components for Unicode(ICU) 데이터 정렬 라이브러리(Babelfish에서 사용하는 라이브러리)를 따르는 사용자 지정 유형의 인덱스는 라이브러리 버전이 변경되어도 무효화되지 않습니다.
- COLLATIONPROPERTY 함수 – 데이터 정렬 속성은 지원되는 Babelfish BBF 데이터 정렬에 대해서만 구현됩니다. 자세한 내용은 [Babelfish supported collations table]을 참조하세요.
- 유니코드 정렬 규칙 차이점 - SQL Server의 SQL 데이터 정렬은 유니코드로 인코딩된 데이터(`nchar` 및 `nvarchar`)를 유니코드로 인코딩되지 않은 데이터(`char` 및 `varchar`)와 다르게 정렬합니다. Babelfish 데이터베이스는 항상 UTF-8으로 인코딩되며, `char` 또는 `varchar`에 대한 정렬 순서가 `nchar` 또는 `nvarchar`에 대한 정렬 순서와 같도록 데이터 유형에 관계없이 항상 유니코드 정렬 규칙을 일관되게 적용합니다.

- 2차와 동일한 데이터 정렬 및 정렬 동작 – 기본 ICU 유니코드 2차와 동일한(CI_AS) 데이터 정렬은 문장 부호 및 기타 영숫자가 아닌 문자를 숫자 문자 앞에 정렬하고 숫자 문자는 알파벳 문자 앞에 정렬합니다. 그러나 문장 부호 및 기타 특수 문자의 순서는 다릅니다.
- 3차 데이터 정렬, ORDER BY용 차선책 – SQL 데이터 정렬(예: SQL_Latin1_General_Pref_CP1_CI_AS)은 TERTIARY_WEIGHTS 함수 및 CI_AS 데이터 정렬에서 균등하게 비교되는 문자열을 정렬하는 기능을 지원하여 ABC, ABc, AbC, Abc, aBC, aBc, abC, abc와 같이 먼저 대문자로 정렬되도록 합니다. 따라서 DENSE_RANK OVER (ORDER BY column) 분석 함수는 이러한 문자열을 동일한 순위를 갖는 것으로 평가하지만 파티션 내에서 먼저 대문자를 정렬합니다.

  COLLATE 절을 @colCaseFirst=upper를 지정하는 3차 CS_AS 데이터 정렬을 지정하는 ORDER BY 절에 추가하여 Babelfish와 비슷한 결과를 얻을 수 있습니다. 그러나 colCaseFirst 한정자는 CI_AS 데이터 정렬과 같이 2차와 동일한 문자열이 아닌 3차와 동일한 문자열에만 적용됩니다. 따라서 단일 ICU 데이터 정렬을 사용하여 3차 SQL 데이터 정렬을 에뮬레이션할 수 없습니다.

  해결 방법으로 먼저 BBF_SQL_Latin1_General_CP1_CI_AS 데이터 정렬을 사용하려면 SQL_Latin1_General_Pref_CP1_CI_AS 데이터 정렬을 사용하는 애플리케이션을 수정하는 것이 좋습니다. 그런 다음, COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS를 이 열의 ORDER BY 절에 추가합니다.
- 문자 확장 – 문자 확장은 단일 문자를 기본 수준의 문자 시퀀스와 동일하게 취급합니다. SQL Server의 기본 CI_AS 데이터 정렬은 문자 확장을 지원합니다. ICU 데이터 정렬은 악센트를 구분하지 않는 데이터 정렬에 대해서만 문자 확장을 지원합니다.

  문자 확장이 필요한 경우 비교를 위해 AI 데이터 정렬을 사용합니다. 그러나 이러한 데이터 정렬은 현재 LIKE 연산자에서 지원되지 않습니다.
- char 및 varchar 인코딩 – SQL 데이터 정렬이 char 또는 varchar 데이터 유형에 사용되는 경우 ASCII 127 이전 문자의 정렬 순서는 해당 SQL 데이터 정렬에 대한 특정 코드 페이지에 따라 결정됩니다. SQL 데이터 정렬의 경우 char 또는 varchar로 선언된 문자열은 nchar 또는 nvarchar로 선언된 문자열과 다르게 정렬될 수 있습니다.

  PostgreSQL은 데이터베이스 인코딩으로 모든 문자열을 인코딩하므로 모든 문자를 UTF-8으로 변환하고 유니코드 규칙을 사용하여 정렬합니다.

  SQL 데이터 정렬은 유니코드 규칙을 사용하여 nchar 및 nvarchar 데이터 유형을 정렬하므로 Babelfish는 UTF-8을 사용하여 서버의 모든 문자열을 인코딩합니다. Babelfish는 유니코드 규칙을 사용하여 char 및 varchar 문자열을 정렬하는 것과 동일한 방식으로 nchar 및 nvarchar 문자열을 정렬합니다.

- 보조 문자 – SQL Server 함수 NCHAR, UNICODE, LEN은 유니코드 기본 다국어 평면(BMP) 외부의 코드 포인트에 대한 문자를 지원합니다. 대조적으로 SC가 아닌 데이터 정렬은 서로게이트 쌍 문자를 사용하여 보조 문자를 처리합니다. 유니코드 데이터 형식의 경우 SQL Server는 UCS-2 또는 보조 문자를 사용하는 경우 전체 유니코드 범위(1,114,114자)를 사용하여 최대 65,535자를 나타낼 수 있습니다.
- 가나 구분(KS) 데이터 정렬 - 가나 구분(KS) 데이터 정렬은 Hiragana 및 Katakana 일본어 가나 문자를 다르게 취급합니다. ICU는 일본 데이터 정렬 표준 JIS X 4061을 지원합니다. 이제 사용되지 않는 colhiraganaQ [on | off] 로캘 한정자는 KS 데이터 정렬과 동일한 기능을 제공할 수 있습니다. 그러나 SQL Server와 동일한 이름의 KS 데이터 정렬은 현재 Babelfish에서 지원하지 않습니다.
- 전자/반자 구분(WS) 데이터 정렬 – 싱글바이트 문자(반자)와 더블바이트 문자(전자)로 표현되는 동일한 문자가 다르게 처리되는 경우의 데이터 정렬을 전자/반자 구분(WS)이라고 합니다. 그러나 SQL Server와 동일한 이름의 WS 데이터 정렬은 현재 Babelfish에서 지원하지 않습니다.
- 변형 선택기 구분(VSS) 데이터 정렬 - 변형 선택기 구분(VSS) 데이터 정렬은 일본어 데이터 정렬 Japanese_Bushu_Kakusu_140 및 Japanese_XJIS_140에서 표의 문자 변형 선택기를 구분합니다. 변형 시퀀스는 기본 문자와 추가 변형 선택기로 구성됩니다. _VSS 옵션을 선택하지 않은 경우 변형 선택기는 비교 시 고려되지 않습니다.

  VSS 데이터 정렬은 현재 Babelfish에서 지원되지 않습니다.
- BIN 및 BIN2 데이터 정렬 – BIN2 데이터 정렬은 코드 포인트 순서에 따라 문자를 정렬합니다. UTF-8의 바이트별 이진 순서는 유니코드 코드 포인트 순서를 준수하므로 가장 성능이 좋은 데이터 정렬이 될 수 있습니다. 유니코드 코드 포인트 순서가 애플리케이션에서 작동하는 경우 BIN2 데이터 정렬을 사용하는 것이 좋습니다. 그러나 BIN2 데이터 정렬을 사용하면 문화적으로 예상치 못한 순서로 데이터가 클라이언트에 표시될 수 있습니다. 시간이 진행되면서 소문자에 대한 새로운 매핑이 유니코드에 추가되므로 LOWER 함수는 ICU의 다른 버전에서 다르게 수행될 수 있습니다. 이 경우는 BIN2 데이터 정렬과 관련된 문제가 아닌 좀 더 일반적인 데이터 정렬 버전 관리 문제의 특별한 경우입니다.

  Babelfish는 Babelfish 배포를 통해 BBF_Latin1_General_BIN2 데이터 정렬을 제공하여 유니코드 코드 포인트 순서로 정렬합니다. BIN 데이터 정렬에서는 첫 번째 문자만 wchar로 정렬됩니다. 나머지 문자는 해당 인코딩에 따라 코드 포인트 순서로 바이트별 정렬이 효과적으로 이뤄집니다. 이 방법은 유니코드 데이터 정렬 규칙을 따르지 않으며 Babelfish에서 지원하지 않습니다.
- 비결정적 데이터 정렬 및 CHARINDEX 제한 - 버전 2.1.0 이전 Babelfish 릴리스의 경우 CHARINDEX를 비결정적 데이터 정렬과 함께 사용할 수 없습니다. 기본적으로 Babelfish는 대소문자를 구분하지 않는(비결정적) 데이터 정렬을 사용합니다. 이전 버전의 Babelfish에 CHARINDEX를 사용하면 다음과 같은 런타임 오류가 발생합니다.

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> 이 제한 사항 및 해결 방법은 Babelfish 버전 1.x(Aurora PostgreSQL 13.x 버전)에만 적용됩니다. Babelfish 2.1.0 이상 릴리스에는 문제가 발생하지 않습니다.

이 문제를 해결하려면 다음 중 한 방법을 시도하면 됩니다.

- 표현식을 대소문자를 구분하는 데이터 정렬로 명시적으로 변환하고 LOWER 또는 UPPER를 적용하여 두 인수를 모두 케이스 폴드(case-fold)합니다. 예를 들어 `SELECT charindex('x', a) FROM t1`은 다음이 됩니다.

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- SQL 함수 f_charindex를 만들고 CHARINDEX 호출을 다음 함수에 대한 호출로 바꿉니다.

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

이스케이프 해치를 사용하여 Babelfish 오류 처리 관리

Babelfish는 제어 흐름 및 트랜잭션 상태에 대해 SQL 동작을 가능한 한 모방합니다. Babelfish에 오류가 발생하면 SQL Server 오류 코드와 유사한 오류 코드를 반환합니다. Babelfish가 오류를 SQL Server 코드에 매핑할 수 없으면 고정 오류 코드(33557097)를 반환하고 다음과 같이 오류 유형에 따라 특정 작업을 수행합니다.

- 컴파일 타임 오류인 경우 Babelfish는 트랜잭션을 롤백합니다.
- 런타임 오류인 경우 Babelfish는 배치를 종료하고 트랜잭션을 롤백합니다.
- 클라이언트와 서버 간의 프로토콜 오류인 경우 트랜잭션이 롤백되지 않습니다.

오류 코드를 동등한 코드에 매핑할 수 없고 유사한 오류에 대한 코드를 사용할 수 있는 경우 오류 코드가 대체 코드에 매핑됩니다. 예를 들어 SQL Server 코드, 8143 및 8144를 유발하는 동작이 둘 다 8143으로 매핑됩니다.

매핑할 수 없는 오류는 `TRY... CATCH` 구문을 준수하지 않습니다.

`@@ERROR`를 사용하여 SQL Server 오류 코드를 반환하거나 `@@PGERROR` 함수를 사용하여 PostgreSQL 오류 코드를 반환할 수 있습니다. `fn_mapped_system_error_list` 함수를 사용하여 매핑된 오류 코드 목록을 반환할 수 있습니다. PostgreSQL 오류 코드에 대한 자세한 내용은 [PostgreSQL 웹 사이트](https://www.postgresql.org)를 참조하세요.

Babelfish 이스케이프 해치 설정 수정

오류가 있을 수 있는 문장을 처리하기 위해 Babelfish는 이스케이프 해치라는 특정 옵션을 정의합니다. 이스케이프 해치는 지원되지 않는 기능이나 구문을 발견했을 때 Babelfish 동작을 지정하는 옵션입니다.

`sp_babelfish_configure` 저장 프로시저를 사용하여 이스케이프 해치의 설정을 제어할 수 있습니다. 스크립트를 사용하여 이스케이프 해치를 `ignore` 또는 `strict`로 설정합니다. `strict`로 설정된 경우 Babelfish는 계속하기 전에 수정해야 하는 오류를 반환합니다.

현재 세션과 클러스터 수준에 변경 사항을 적용하려면 `server` 키워드를 포함합니다.

사용법은 다음과 같습니다.

- 모든 이스케이프 해치와 해당 상태 및 사용 정보를 나열하려면 `sp_babelfish_configure`를 실행합니다.
- 명명된 이스케이프 해치와 해당 값을 나열하려면 현재 세션 또는 클러스터 전체에 대해 *`hatch_name`*이 하나 이상의 이스케이프 해치의 식별자인 `sp_babelfish_configure`

'*hatch_name*' 명령을 실행합니다. *hatch_name*에는 '%'와 같은 SQL 와일드카드를 사용할 수 있습니다.

- 하나 이상의 이스케이프 해치를 지정된 값으로 설정하려면 `sp_babelfish_configure ['hatch_name' [, 'strict'|'ignore' [, 'server']]`을 실행합니다. 클러스터 전체 수준에서 설정이 영구적이 되게 하려면 아래 나와 있는 것처럼 `server` 키워드를 포함합니다.

```
EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
```

현재 세션에 대해서만 설정하려면 `server`는 사용하지 마세요.

- 모든 이스케이프 해치를 기본값으로 재설정하려면 `sp_babelfish_configure 'default'`(Babelfish 1.2.0 이상)를 실행합니다.

해치를 식별하는 문자열에는 SQL 와일드카드가 포함될 수 있습니다. 예를 들어 다음은 모든 구문 이스케이프 해치를 Aurora PostgreSQL 클러스터의 `ignore`로 설정합니다.

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

다음 표에서는 Babelfish 사전 정의된 이스케이프 해치에 대한 설명과 기본값을 확인할 수 있습니다.

이스케이프 해치	설명	Default
`escape_hatch_checkpoint`	절차 코드에서 CHECKPOINT 문을 사용할 수 있지만 CHECKPOINT 문은 현재 구현되지 않았습니다.	무시
`escape_hatch_constraint_name_for_default`	기본 제약 조건 이름과 관련된 Babelfish 동작을 제어합니다.	무시
`escape_hatch_database_misc_options`	CREATE DATABASE: CONTAINMENT, DB_CHAINING, TRUSTWORTHY, PERSISTENT_LOG_BUFFER에서 다음 옵션과 관련된 Babelfish 동작을 제어합니다.	무시

이스케이프 해치	설명	Default
escape_hatch_for_replication	테이블을 만들거나 변경하는 경우 [NOT] FOR REPLICATION 절과 관련된 Babelfish 동작을 제어합니다.	strict
escape_hatch_fulltext	DEFAULT_FULLTEXT_L ANGUAGE in CREATE/ALTER DATABASE, CREATE FULLTEXT INDEX, 또는 sp_fulltext_database 와 같은 FULLTEXT 기능과 관련된 Babelfish 동작을 제어합니다.	무시
escape_hatch_ignore_dup_key	CREATE/ALTER TABLE 및 CREATE INDEX와 관련된 Babelfish 동작을 제어합니다. IGNORE_DUP_KEY=ON인 경우 `strict`(기본값)로 설정하면 오류가 반환되고`ignore`로 설정하면 오류가 무시됩니다(Babelfish 버전 1.2.0 이상).	strict
escape_hatch_index_clustering	인덱스 및 PRIMARY KEY 또는 UNIQUE 제약 조건에 대한 CLUSTERED 또는 NONCLUSTE RED 키워드와 관련된 Babelfish 동작을 제어합니다. 클러스터링이 무시되면 NONCLUSTERED가 지정된 것처럼 인덱스 또는 제약 조건이 계속 만들어집니다.	무시

이스케이프 해치	설명	Default
escape_hatch_index_columnstore	COLUMNSTORE 절과 관련된 Babelfish 동작을 제어합니다. ignore를 지정한 경우 Babelfish는 일반 B-tree 인덱스를 생성합니다.	strict
escape_hatch_join_hints	LOOP, HASH, MERGE, REMOTE, REDUCE, REDISTRIBUTE, REPLICATE 등 JOIN 연산자의 키워드 동작을 제어합니다.	무시
escape_hatch_language_non_english	화면 메시지에 대해 영어 이외의 언어와 관련된 Babelfish 동작을 제어합니다. Babelfish는 현재 화면 메시지에 대해 us_english 만 지원합니다. SET LANGUAGE는 언어 이름을 포함하는 변수를 사용할 수 있으므로 설정 중인 실제 언어는 런타임 시에만 감지할 수 있습니다.	strict
escape_hatch_login_hashed_password	무시하면 CREATE LOGIN과 ALTER LOGIN에 대한 HASHED 키워드의 오류를 표시하지 않습니다.	strict
escape_hatch_login_misc_options	무시하면 CREATE LOGIN 및 ALTER LOGIN에 대한 HASHED, MUST_CHANGE , OLD_PASSWORD 및 UNLOCK을 제외한 다른 키워드의 오류를 표시하지 않습니다.	strict

이스케이프 해치	설명	Default
escape_hatch_login_old_password	무시하면 CREATE LOGIN과 ALTER LOGIN에 대한 OLD_PASSWORD 키워드의 오류를 표시하지 않습니다.	strict
escape_hatch_login_password_must_change	무시하면 CREATE LOGIN과 ALTER LOGIN에 대한 MUST_CHANGE 키워드의 오류를 표시하지 않습니다.	strict
escape_hatch_login_password_unlock	무시하면 CREATE LOGIN과 ALTER LOGIN에 대한 UNLOCK 키워드의 오류를 표시하지 않습니다.	strict
escape_hatch_nocheck_add_constraint	제약 조건에 대한 WITH CHECK 또는 NOCHECK 절과 관련된 Babelfish 동작을 제어합니다.	strict
escape_hatch_nocheck_existing_constraint	FOREIGN KEY 또는 CHECK 제약 조건과 관련된 Babelfish 동작을 제어합니다.	strict
escape_hatch_query_hints	쿼리 힌트와 관련된 Babelfish 동작을 제어합니다. 이 옵션을 무시하도록 설정하면 서버는 OPTION (...) 절을 사용하여 쿼리 처리 측면을 지정하는 힌트를 무시합니다. 예에는 SELECT FROM ...이 있습니다. OPTION(MERGE JOIN HASH, MAXRECURSION 10)).	무시

이스케이프 해치	설명	Default
`escape_hatch_rowversion`	ROWVERSION 및 TIMESTAMP 데이터 유형의 동작을 제어합니다. 사용 정보는 [구현이 제한된 Babelfish 기능 사용](#) 단원을 참조하세요.	strict
`escape_hatch_schemabinding_function`	WITH SCHEMABINDING 절과 관련된 Babelfish 동작을 제어합니다. 기본적으로 WITH SCHEMABINDING 절은 CREATE 또는 ALTER FUNCTION 명령으로 지정되면 무시됩니다.	무시
`escape_hatch_schemabinding_procedure`	WITH SCHEMABINDING 절과 관련된 Babelfish 동작을 제어합니다. 기본적으로 WITH SCHEMABINDING 절은 CREATE 또는 ALTER PROCEDURE 명령으로 지정되면 무시됩니다.	무시
`escape_hatch_rowguidcol_column`	테이블을 만들거나 변경하는 경우 ROWGUIDCOL 절과 관련된 Babelfish 동작을 제어합니다.	strict
`escape_hatch_schemabinding_trigger`	WITH SCHEMABINDING 절과 관련된 Babelfish 동작을 제어합니다. 기본적으로 WITH SCHEMABINDING 절은 CREATE 또는 ALTER TRIGGER 명령으로 지정되면 무시됩니다.	무시

이스케이프 해치	설명	Default
`escape_hatch_schemabinding_view`	WITH SCHEMABINDING 절과 관련된 Babelfish 동작을 제어합니다. 기본적으로 WITH SCHEMABINDING 절은 CREATE 또는 ALTER VIEW 명령으로 지정되면 무시됩니다.	무시
`escape_hatch_session_settings`	지원되지 않는 세션 수준 SET 문에 대한 Babelfish 동작을 제어합니다.	무시
`escape_hatch_showplan_all`	SET SHOWPLAN_ALL 및 SET STATISTICS PROFILE과 관련된 Babelfish 동작을 제어합니다. 무시하도록 설정하면 SET BABELFISH_SHOWPLAN_ALL 및 SET BABELFISH_STATISTICS PROFILE처럼 동작합니다. 엄격하게 설정하면 자동으로 무시됩니다.	strict
`escape_hatch_storage_on_partition`	파티셔닝을 정의하는 경우 ON `partition_scheme column` 절과 관련된 Babelfish 동작을 제어합니다. Babelfish는 현재 파티셔닝을 구현하지 않습니다.	strict

이스케이프 해치	설명	Default
`escape_hatch_storage_options`	CREATE, ALTER DATABASE, TABLE, INDEX에 사용되는 모든 스토리지 옵션의 이스케이프 해치. 여기에는 테이블, 인덱스, 제약 조건과 데이터베이스에 대한 스토리지 위치(파티션, 파일 그룹)를 정의하는 (LOG) ON, TEXTIMAGE_ON, FILESTREAM_ON 절이 포함됩니다. 이 이스케이프 해치 설정은 이러한 모든 절(ON [PRIMARY] 및 ON 'DEFAULT' 포함)에 적용됩니다. ON partition_scheme(열)이 있는 테이블이나 인덱스에 대해 파티션이 지정된 경우는 예외입니다.	무시
`escape_hatch_table_hints`	WITH (...) 절을 사용하여 지정된 테이블 힌트의 동작을 제어합니다.	무시
`escape_hatch_unique_constraint`	엄격으로 설정하면 인덱싱된 열에서 NULL 값을 처리할 때 SQL Server와 PostgreSQL의 의미 차이가 모호한 경우 오류가 발생할 수 있습니다. 의미 차이는 비현실적인 사용 사례에서만 나타나므로 오류가 표시되지 않도록 이 이스케이프 해치를 '무시'로 설정할 수 있습니다. 다음 버전에서 더 이상 사용되지 않음: 3.6.0 이상 버전, 4.2.0 이상 버전	strict

Babelfish for Aurora PostgreSQL DB 클러스터 생성

Babelfish for Aurora PostgreSQL은 Aurora PostgreSQL 버전 13.4 이상에서 지원됩니다.

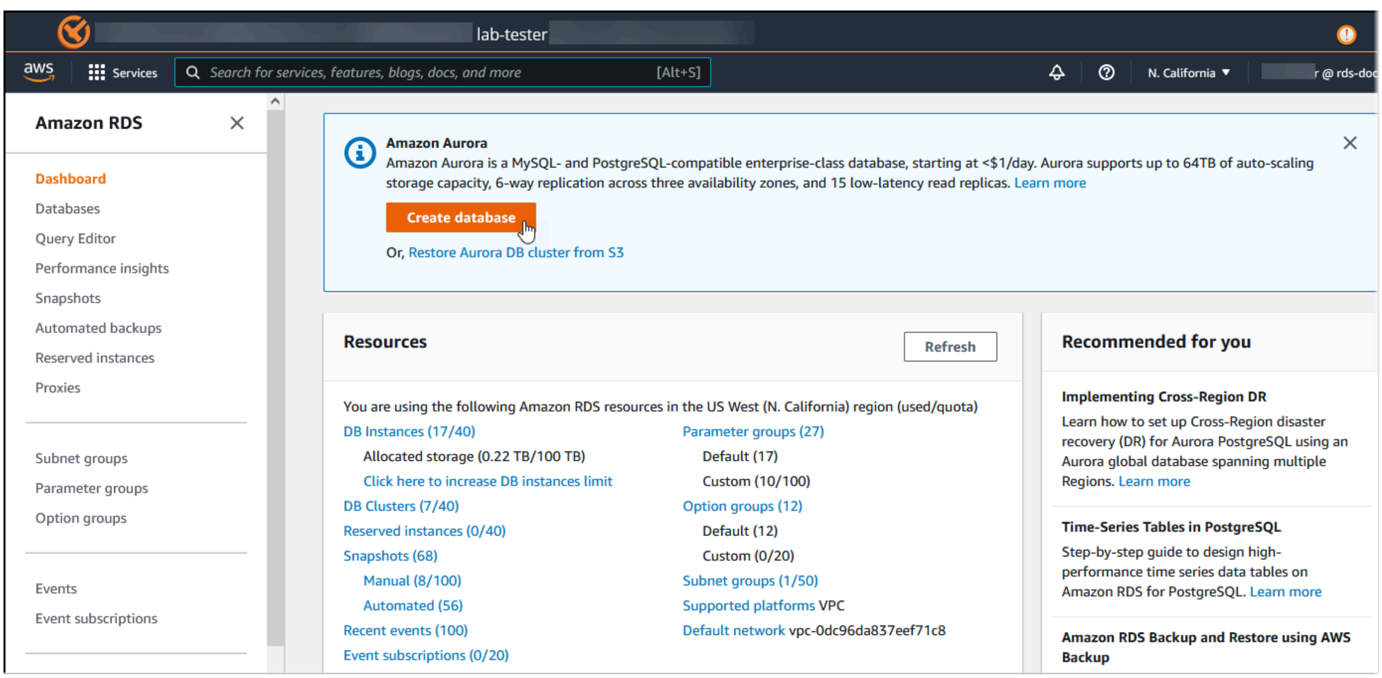
AWS Management Console 또는 AWS CLI를 사용하여 Babelfish에서 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다.

> **Note**
>
> Aurora PostgreSQL 클러스터에서 `babelfish_db` 데이터베이스 이름은 Babelfish용으로 예약되어 있습니다. Babelfish for Aurora PostgreSQL에 자체 "babelfish_db" 데이터베이스를 생성하면 Aurora가 Babelfish를 성공적으로 프로비저닝하지 못합니다.

콘솔

AWS Management Console에서 실행되는 Babelfish를 사용하여 클러스터를 만들려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 열고 데이터베이스 생성(Create database)을 선택합니다.

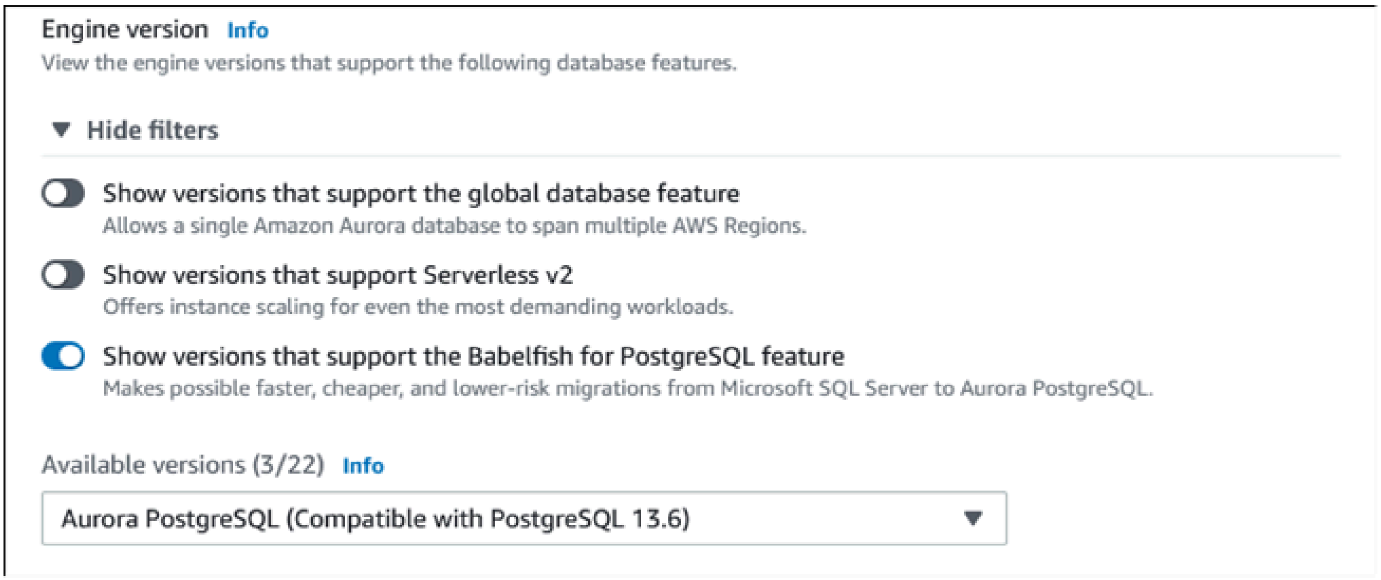


2. 데이터베이스 생성 방법 선택(Choose a database creation method)에서 다음 중 하나를 수행합니다.

   - 자세한 엔진 옵션을 지정하려면 표준 생성(Standard create)을 선택합니다.

- Aurora 클러스터에 대한 모범 사례를 지원하는 사전 구성된 옵션을 사용하려면 간편한 생성(Easy create)을 선택합니다.

3. 엔진 유형에서 Aurora(PostgreSQL 호환)를 선택합니다.
4. 필터 표시(Show filters)를 선택한 다음, Babelfish for PostgreSQL 기능을 지원하는 버전 표시(Show versions that support the Babelfish for PostgreSQL feature)를 선택하여 Babelfish를 지원하는 엔진 유형을 나열합니다. Babelfish는 현재 Aurora PostgreSQL 13.4 이상 버전에서 지원됩니다.
5. 사용 가능한 버전(Available versions)에서 Aurora PostgreSQL 버전을 선택합니다. 최신 Babelfish 기능을 사용하려면 가장 높은 Aurora PostgreSQL 메이저 버전을 선택하세요.

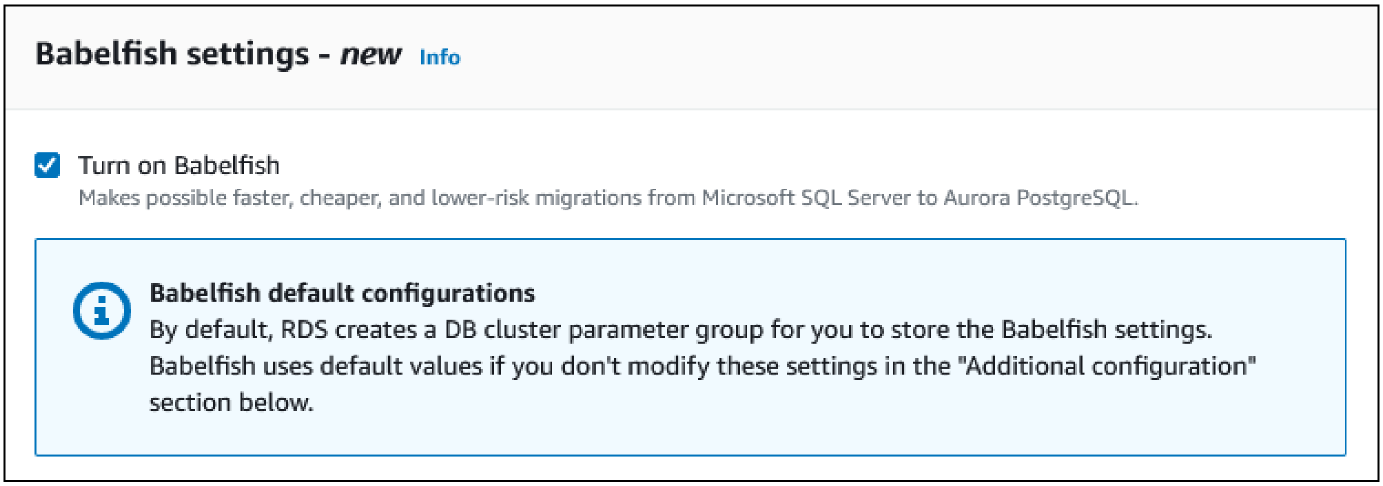


6. 템플릿(Templates)에서 사용 사례에 맞는 템플릿을 선택합니다.
7. DB 클러스터 식별자(DB cluster identifier)에서 나중에 쉽게 찾을 수 있는 이름을 DB 클러스터 목록에 입력합니다.
8. 마스터 사용자 이름(Master username)에 관리자 사용자 이름을 입력합니다. Aurora PostgreSQL의 기본값은 `postgres`입니다. 기본값을 적용하거나 다른 이름을 선택할 수 있습니다. 예를 들어, SQL Server 데이터베이스에 사용된 이름 지정 규칙을 따르려면 마스터 사용자 이름에 `sa`(시스템 관리자)를 입력하면 됩니다.

   현재 `sa`라는 이름의 사용자를 생성하지 않는 경우 선택한 클라이언트를 사용하여 나중에 사용자를 하나 만들 수 있습니다. 사용자를 생성한 후 `ALTER SERVER ROLE` 명령을 이용하여 클러스터의 `sysadmin` 그룹(역할)에 사용자를 추가합니다.

> **Warning**
>
> 마스터 사용자 이름은 항상 소문자를 사용해야 하며, 그러지 않으면 DB 클러스터가 TDS 포트를 통해 Babelfish에 연결할 수 없습니다.

9. 마스터 암호(Master password)에서 강력한 암호를 만들고 암호를 확인합니다.
10. 다음 옵션의 경우 Babelfish 설정(Babelfish settings) 섹션에서 DB 클러스터 설정을 지정합니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 섹션을 참조하세요.
11. Babelfish 기능을 사용할 수 있도록 하려면 Babelfish 켜기(Turn on Babelfish) 상자를 선택합니다.



12. DB 클러스터 파라미터 그룹(DB cluster parameter group)에서 다음 중 하나를 수행합니다.
    - 새로 생성(Create new)을 선택하여 Babelfish를 켠 상태에서 새 파라미터 그룹을 생성합니다.
    - 기존 선택(Choose existing)을 선택하여 기존 파라미터 그룹을 사용합니다. 기존 그룹을 사용하는 경우 클러스터를 생성하기 전에 그룹을 수정하고 Babelfish 파라미터에 대한 값을 추가해야 합니다. Babelfish 파라미터에 대한 자세한 내용은 Babelfish용 DB 클러스터 파라미터 그룹 설정 섹션을 참조하세요.

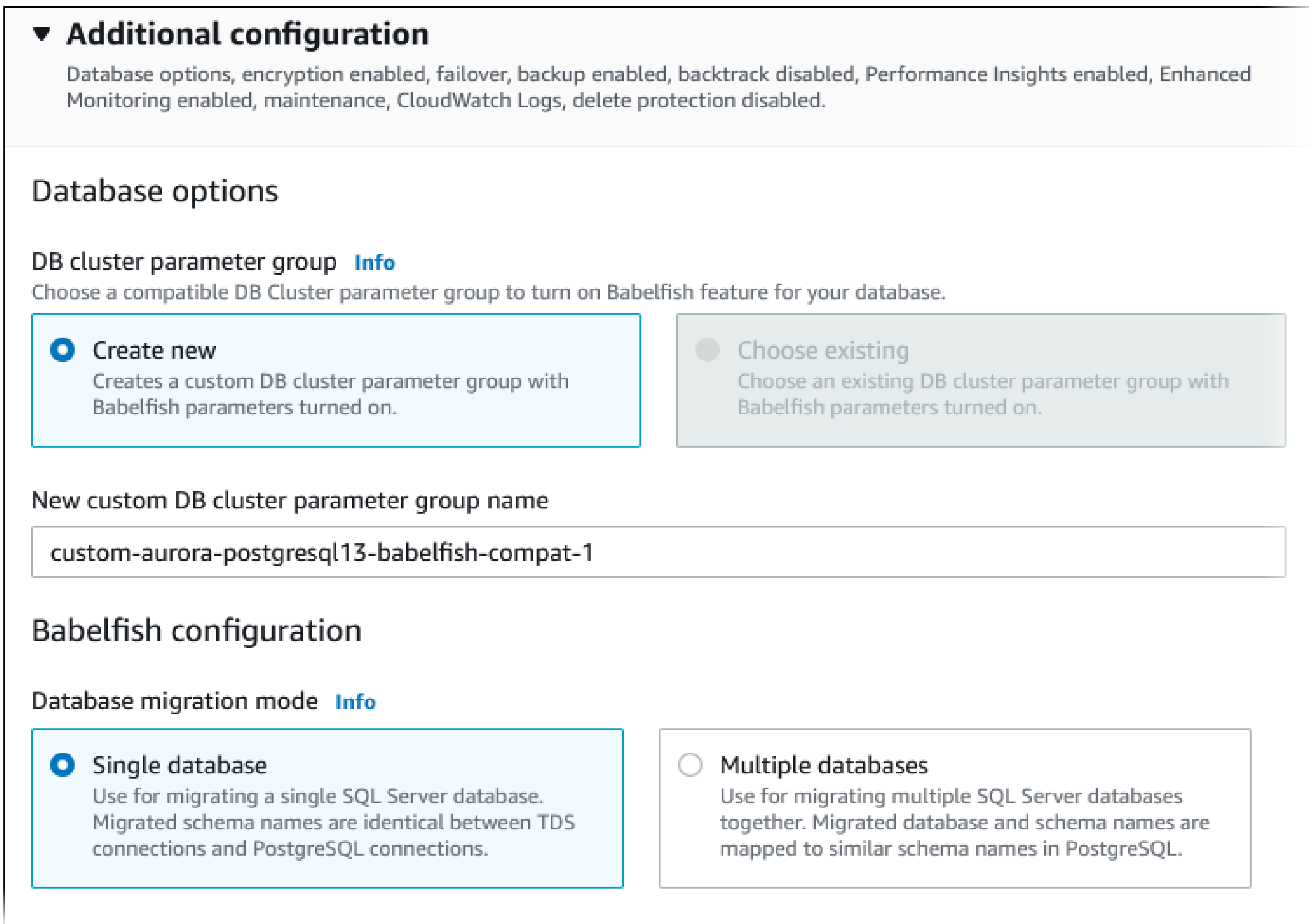
      기존 그룹을 사용하는 경우 다음 상자에 그룹 이름을 입력합니다.
13. 데이터베이스 마이그레이션 모드(Database migration mode)에서 다음 중 하나를 선택합니다.
    - 단일 SQL Server 데이터베이스를 마이그레이션할 단일 데이터베이스(Single database).

      경우에 따라 Babelfish를 사용하지 않고 기본 Aurora PostgreSQL로 완전히 마이그레이션하는 것을 목표로 여러 사용자 데이터베이스를 함께 마이그레이션할 수 있습니다. 최종 애플리케이션에 통합 스키마(단일dbo 스키마)가 필요한 경우 먼저 SQL Server 데이터베이스를 단일 SQL

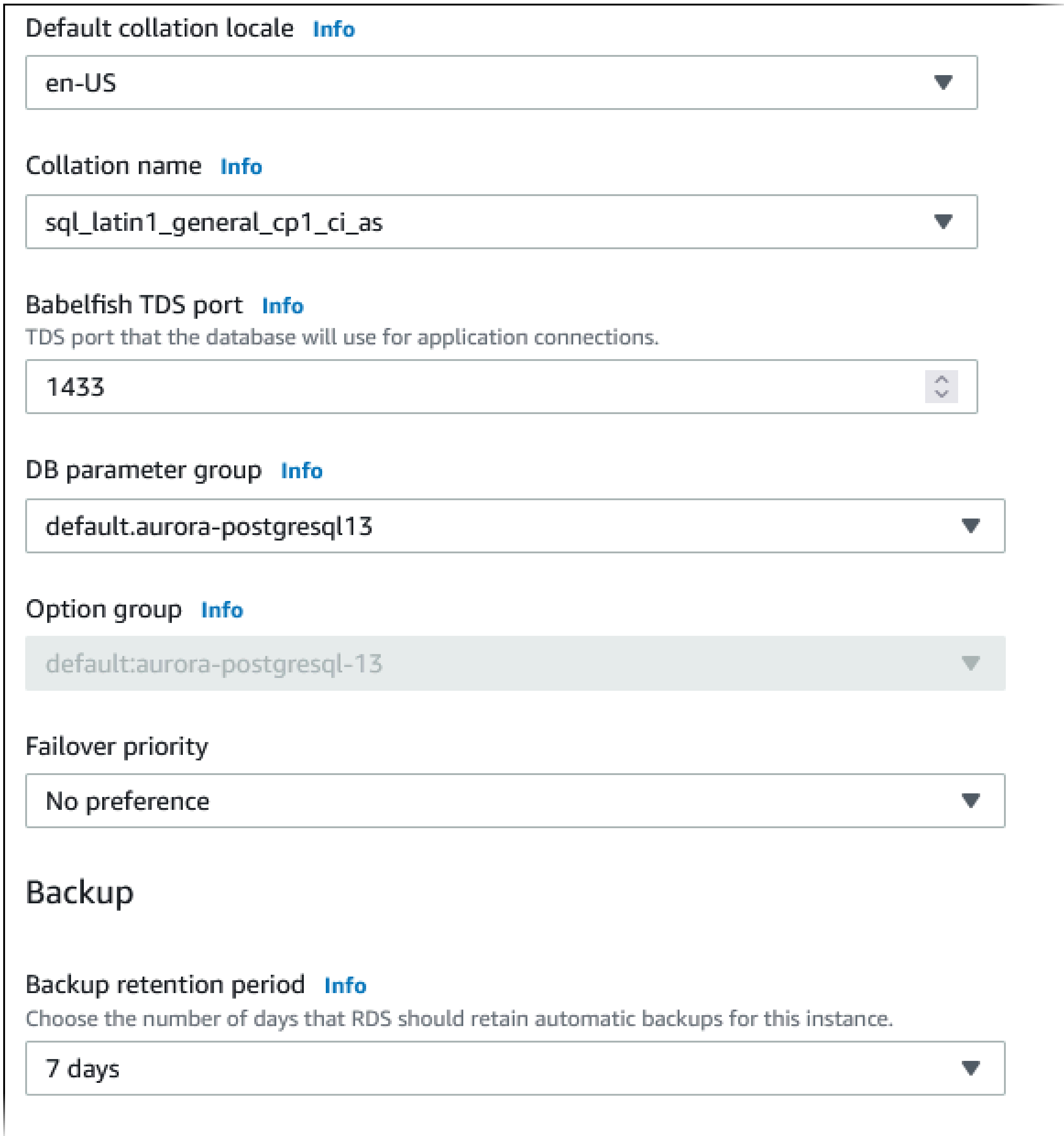
서버 데이터베이스로 통합해야 합니다. 그런 다음, 단일 데이터베이스(Single database)모드를 사용하여 Babelfish로 마이그레이션합니다.

- 여러 SQL Server 데이터베이스를 마이그레이션할 다중 데이터베이스(Multiple databases)(단일 SQL Server 설치에서 시작). 다중 데이터베이스 모드는 단일 SQL Server 설치에서 시작되지 않은 여러 데이터베이스를 통합하지 않습니다. 여러 데이터베이스 마이그레이션에 대한 자세한 내용은 단일 데이터베이스 또는 여러 데이터베이스에서 Babelfish 사용 섹션을 참조하세요.

> **Note**
>
> Aurora PostgreSQL 버전 16에서는 기본적으로 여러 데이터베이스가 데이터베이스 마이그레이션 모드로 선택됩니다.



14. 기본 데이터 정렬 로캘(Default collation locale)에서 서버 로캘을 입력합니다. 기본값은 en-US입니다. 데이터 정렬에 대한 자세한 내용은 Babelfish for Aurora PostgreSQL의 데이터 정렬 이해 섹션을 참조하세요.
15. 데이터 정렬 이름(Collation name) 필드에 기본 데이터 정렬을 입력합니다. 기본값은 `sql_latin1_general_cp1_ci_as`입니다. 자세한 내용은 Babelfish for Aurora PostgreSQL의 데이터 정렬 이해섹션을 참조하세요.
16. Babelfish TDS 포트의 경우 기본 포트인 1433을 입력합니다. 현재 Babelfish는 DB 클러스터에 1433 포트만 지원합니다.
17. DB 파라미터 그룹(DB parameter group)에서 파라미터 그룹을 선택하거나 Aurora에서 기본 설정으로 새 그룹을 생성하도록 합니다.
18. 장애 조치 우선순위(Failover priority)에서 인스턴스에 대한 장애 조치 우선순위를 선택합니다. 값을 선택하지 않은 경우 기본값은 `tier-1`입니다. 기본 인스턴스 장애로부터 복구할 때 이 우선 순위에 따라 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 섹션을 참조하세요.
19. 백업 보존 기간(Backup retention period)에서 Aurora가 데이터베이스 백업 사본을 보존하는 기간(1~35일)을 선택합니다. 백업 복사본은 데이터베이스를 마지막 특정 시점으로 복원(PITR)하는 데 사용할 수 있습니다. 기본 보존 기간은 7일입니다.



20. 스냅샷을 생성할 때 DB 인스턴스 태그를 DB 스냅샷에 복사하려면 태그를 스냅샷에 복사(Copy tags to snapshots)를 선택합니다.

> **Note**
>
> 스냅샷에서 DB 클러스터를 복원할 때는 Babelfish for Aurora PostgreSQL DB 클러스터로 복원되지 않습니다. Babelfish를 다시 활성화하려면 DB 클러스터 파라미터 그룹의 Babelfish 기본 설정을 제어하는 파라미터를 켜야 합니다. Babelfish 파라미터에 대한 자세한 내용은 Babelfish용 DB 클러스터 파라미터 그룹 설정 단원을 참조하세요.

21. 암호화 사용 설정(Enable Encryption)을 선택하면 이 DB 클러스터의 미사용 데이터 암호화(Aurora 스토리지 암호화)를 설정할 수 있습니다.

22. 성능 개선 도우미 사용 설정(Enable Performance Insights)을 선택하여 Amazon RDS 성능 개선 도우미를 설정할 수 있습니다.

23. DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 시작하려면 향상된 모니터링 사용 설정(Enable enhanced monitoring)을 선택합니다.

24. PostgreSQL 로그(PostgreSQL log)를 선택하여 Amazon CloudWatch Logs에 로그 파일을 게시합니다.

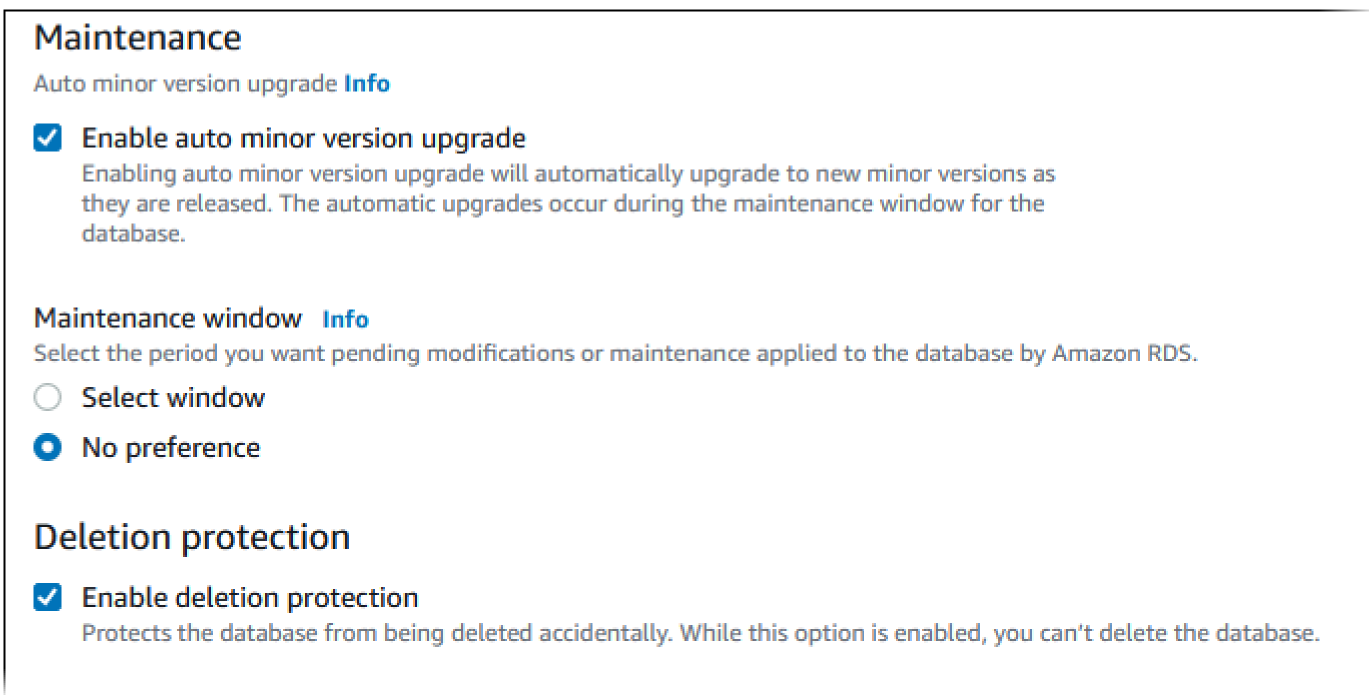
25. 마이너 버전 자동 업그레이드 사용 설정(Enable auto minor version upgrade)을 선택하여 마이너 버전 업그레이드가 사용 가능할 때 Aurora DB 클러스터를 자동으로 업데이트합니다.

26. 유지 관리 기간(Maintenance window)에서 다음을 수행합니다.

    - Amazon RDS가 수정하거나 유지 관리를 수행할 시간을 선택하려면 기간 선택(Select window)을 선택합니다.
    - 예정되지 않은 시간에 Amazon RDS 유지 관리를 수행하려면 선호 시간 없음(No preference)을 선택합니다.

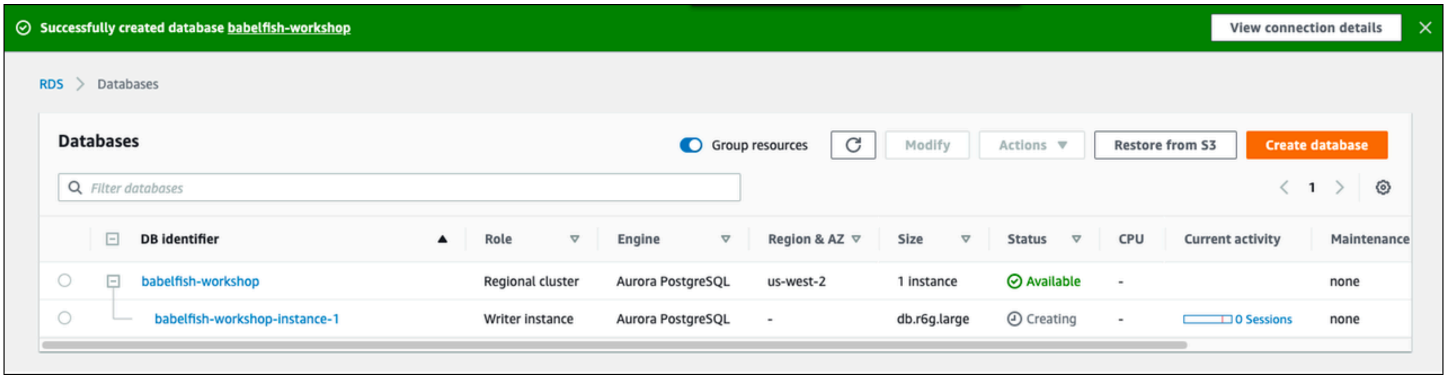
27. 삭제 방지 사용 설정(Enable deletion protection)을 선택하여 실수로 데이터베이스가 삭제되지 않도록 보호합니다.

    이 기능을 설정하면 데이터베이스를 직접 삭제할 수 없습니다. 대신, 데이터베이스를 삭제하기 전에 데이터베이스 클러스터를 수정하고 이 기능을 해제해야 합니다.



28. 데이터베이스 생성(Create database)을 선택합니다.

데이터베이스(Databases) 목록에서 Babelfish에 대해 설정된 새 데이터베이스를 찾을 수 있습니다. 상태(Status) 열은 배포가 완료되면 사용 가능(Available)이 표시됩니다.



AWS CLI

AWS CLI를 사용하여 Babelfish for Aurora PostgreSQL을 생성할 때는 클러스터에 사용할 DB 클러스터 파라미터 그룹의 이름에 명령을 전달해야 합니다. 자세한 내용은 DB 클러스터 사전 조건 섹션을 참조하세요.

AWS CLI를 사용하여 Babelfish에서 Aurora PostgreSQL DB 클러스터를 생성할 수 있기 전에 다음을 수행합니다.

- Amazon Aurora 엔드포인트 및 할당량)Amazon Aurora endpoints and quotas)의 서비스 목록에서 엔드포인트 URL을 선택합니다.
- 클러스터에 대한 파라미터 그룹을 생성합니다. 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.
- 파라미터 그룹을 수정하고 Babelfish를 설정하는 파라미터를 추가합니다.

AWS CLI를 사용하여 Babelfish가 있는 Aurora PostgreSQL DB 클러스터를 생성하려면

다음 예에서는 기본 마스터 사용자 이름(`postgres`)을 사용합니다. 필요에 따라 DB 클러스터에 대해 생성한 사용자 이름(예: `sa` 또는 기본값을 사용하지 않은 경우에 선택한 사용자 이름)으로 대체합니다.

1. 파라미터 그룹을 생성합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-parameter-group \
```

```
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--db-parameter-group-family aurora-postgresql14 \
--description "description"
```

Windows의 경우:

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url endpoint-URL ^
--db-cluster-parameter-group-name parameter-group ^
--db-parameter-group-family aurora-postgresql14 ^
--description "description"
```

2. 파라미터 그룹을 수정하여 Babelfish를 설정합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

   Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. 새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC(Virtual Private Cloud) 보안 그룹 ID를 확인한 다음, create-db-cluster 명령을 호출합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier  cluster-name\
--master-username postgres \
--manage-master-user-password \
--engine aurora-postgresql \
--engine-version 14.3            \
```

```
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

Windows의 경우:

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--manage-master-user-password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

이 예제에서는 마스터 사용자 암호를 생성하고 이를 Secrets Manager에서 관리하는 `--manage-master-user-password` 옵션을 지정합니다. 자세한 내용은 [Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리](#) 섹션을 참조하세요. 또는 `--master-password` 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

4. DB 클러스터에 대한 프라이머리 인스턴스를 명시적으로 생성합니다. 다음과 같이 [create-db-instance](#) 명령을 호출할 때 3단계에서 생성한 클러스터의 이름을 `--db-cluster-identifier` 인수에 사용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

   Windows의 경우:

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
```

```
--engine aurora-postgresql
```

SQL Server 데이터베이스를 Babelfish for Aurora PostgreSQL로 마이그레이션

Babelfish for Aurora PostgreSQL을 사용하여 SQL Server 데이터베이스에서 Amazon Aurora PostgreSQL DB 클러스터로 마이그레이션할 수 있습니다. 마이그레이션하기 전에 단일 데이터베이스 또는 여러 데이터베이스에서 Babelfish 사용 섹션을 검토하세요.

주제

마이그레이션 프로세스 개요

다음 요약에는 SQL Server 애플리케이션을 성공적으로 마이그레이션하고 Babelfish와 함께 사용하는 데 필요한 단계가 나열되어 있습니다. 내보내기 및 가져오기 프로세스에 사용할 수 있는 도구에 대한 자세한 내용과 심층적인 정보는 SQL Server에서 Babelfish로 마이그레이션하기 위한 가져오기/내보내기 도구 섹션을 참조하세요. 데이터를 로드하려면 대상 엔드포인트로 Aurora PostgreSQL DB 클러스터가 있는 AWS DMS를 사용하는 것이 좋습니다.

1. Babelfish를 설정한 새로운 Aurora PostgreSQL DB 클러스터를 생성합니다. 자세한 방법은 Babelfish for Aurora PostgreSQL DB 클러스터 생성을 참조하세요.

   SQL Server 데이터베이스에서 내보낸 다양한 SQL 아티팩트를 가져오려면 sqlcmd와 같은 SQL Server 도구를 사용하여 Babelfish 클러스터에 연결하면 됩니다. 자세한 내용은 SQL Server 클라이언트 도구를 사용하여 DB 클러스터에 연결 섹션을 참조하세요.

2. 마이그레이션할 SQL Server 데이터베이스에서 데이터 정의 언어(DDL)를 내보냅니다. DDL은 사용자 데이터(예: 테이블, 인덱스 및 뷰)와 사용자가 작성한 데이터베이스 코드(예: 저장 프로시저, 사용자 정의 함수 및 트리거)를 포함하는 데이터베이스 객체를 설명하는 SQL 코드입니다.

   자세한 내용은 SQL Server Management Studio(SSMS)를 사용하여 Babelfish로 마이그레이션 섹션을 참조하세요.

3. Babelfish가 SQL Server에서 실행 중인 애플리케이션을 효과적으로 지원할 수 있도록 평가 도구를 실행하여 변경해야 할 내용 범위를 평가합니다. 자세한 내용은 SQL Server 및 Babelfish 간의 차이점 평가 및 처리 섹션을 참조하세요.

4. AWS DMS 대상 엔드포인트 제한 사항을 검토하고 필요에 따라 DDL 스크립트를 업데이트합니다. 자세한 내용은 Babelfish for Aurora PostgreSQL을 대상으로 사용에서 Babelfish 테이블과 함께 PostgreSQL 대상 엔드포인트 사용 시 제한 사항을 참조하세요.
5. 새 Babelfish DB 클러스터에서, 지정한 T-SQL 데이터베이스 내에서 DDL을 실행하여 프라이머리 키 제약 조건이 있는 스키마, 사용자 정의 데이터 유형과 테이블만 생성합니다.
6. AWS DMS를 사용하여 SQL Server의 데이터를 Babelfish 테이블로 마이그레이션합니다. SQL Server Change Data Capture 또는 SQL Replication를 사용한 연속 복제의 경우, Babelfish 대신 Aurora PostgreSQL을 엔드포인트로 사용합니다. 방법은 AWS Database Migration Service의 대상으로 Babelfish for Aurora PostgreSQL 사용을 참조하십시오.
7. 데이터 로드가 완료되면, Babelfish 클러스터의 애플리케이션을 지원하는 나머지 T-SQL 객체를 모두 생성합니다.
8. SQL Server 데이터베이스 대신 Babelfish 엔드포인트에 연결하도록 클라이언트 애플리케이션을 재구성합니다. 자세한 내용은 Babelfish DB 클러스터에 연결 섹션을 참조하세요.
9. 필요한 경우 애플리케이션을 수정하여 다시 테스트합니다. 자세한 내용은 Babelfish for Aurora PostgreSQL과 SQL Server의 차이점 섹션을 참조하세요.

여전히 클라이언트 측 SQL 쿼리를 평가해야 합니다. SQL Server 인스턴스에서 생성된 스키마는 서버 측 SQL 코드만 변환합니다. 다음 단계를 수행하는 것이 좋습니다.

- TSQL_Replay 미리 정의된 템플릿과 함께 SQL Server 프로파일러를 사용하여 클라이언트측 쿼리를 캡처합니다. 이 템플릿은 반복적인 튜닝 및 테스트를 위해 재생할 수 있는 T-SQL 문 정보를 캡처합니다. SQL Server Management Studio의 Tools(도구) 메뉴에서 프로파일러를 시작할 수 있습니다. SQL Server 프로파일러(SQL Server Profiler)를 선택하여 프로파일러를 열고 TSQL_Replay 템플릿을 선택합니다.

  Babelfish 마이그레이션에 사용하려면 추적을 시작한 다음 기능 테스트를 사용하여 애플리케이션을 실행합니다. 프로파일러는 T-SQL 문을 캡처합니다. 테스트를 마친 후 추적을 중단합니다. 클라이언트 측 쿼리를 사용하여 XML 파일에 결과를 저장합니다(파일(File) > 다른 이름으로 저장(Save as) > XML 파일을 추적하여 재생(Trace XML File for Replay)).

  자세한 내용은 Microsoft 설명서에서 SQL Server 프로파일러를 참조하세요. TSQL_Replay 템플릿에 대한 자세한 내용은 SQL Server 프로파일러 템플릿을 참조하세요.
- 복잡한 클라이언트 측 SQL 쿼리가 있는 애플리케이션의 경우 Babelfish 호환성을 위해 Babelfish Compass를 사용하여 이러한 쿼리를 분석하는 것이 좋습니다. 분석에서 클라이언트 측 SQL 문에 지원되지 않는 SQL 기능이 포함되어 있음이 드러난 경우 클라이언트 애플리케이션의 SQL 측면을 검토하고 필요한 경우 수정합니다.

- SQL 쿼리를 확장 이벤트(.xel 형식)로 캡처할 수도 있습니다. 이렇게 하려면 SSMS XEvent Profiler를 사용해야 합니다. .xel 파일을 생성한 후, SQL 문을 Compass에서 처리할 수 있는 .xml 파일로 추출합니다. 자세한 내용은 Microsoft 설명서에서 SQL Server 프로파일러를 참조하세요.

마이그레이션된 애플리케이션에 필요한 모든 테스트, 분석 및 수정이 완료되면 프로덕션용으로 Babelfish 데이터베이스를 사용할 수 있습니다. 이렇게 하려면 원래 데이터베이스를 중지하고 라이브 클라이언트 애플리케이션을 리디렉션하여 Babelfish TDS 포트를 사용하면 됩니다.

> **Note**
>
> AWS DMS는 이제 Babelfish에서 데이터 복제를 지원합니다. 자세한 내용은 이제 AWS DMS가 소스로서 Aurora PostgreSQL용 Babelfish를 지원함을 참조하세요.

SQL Server 및 Babelfish 간의 차이점 평가 및 처리

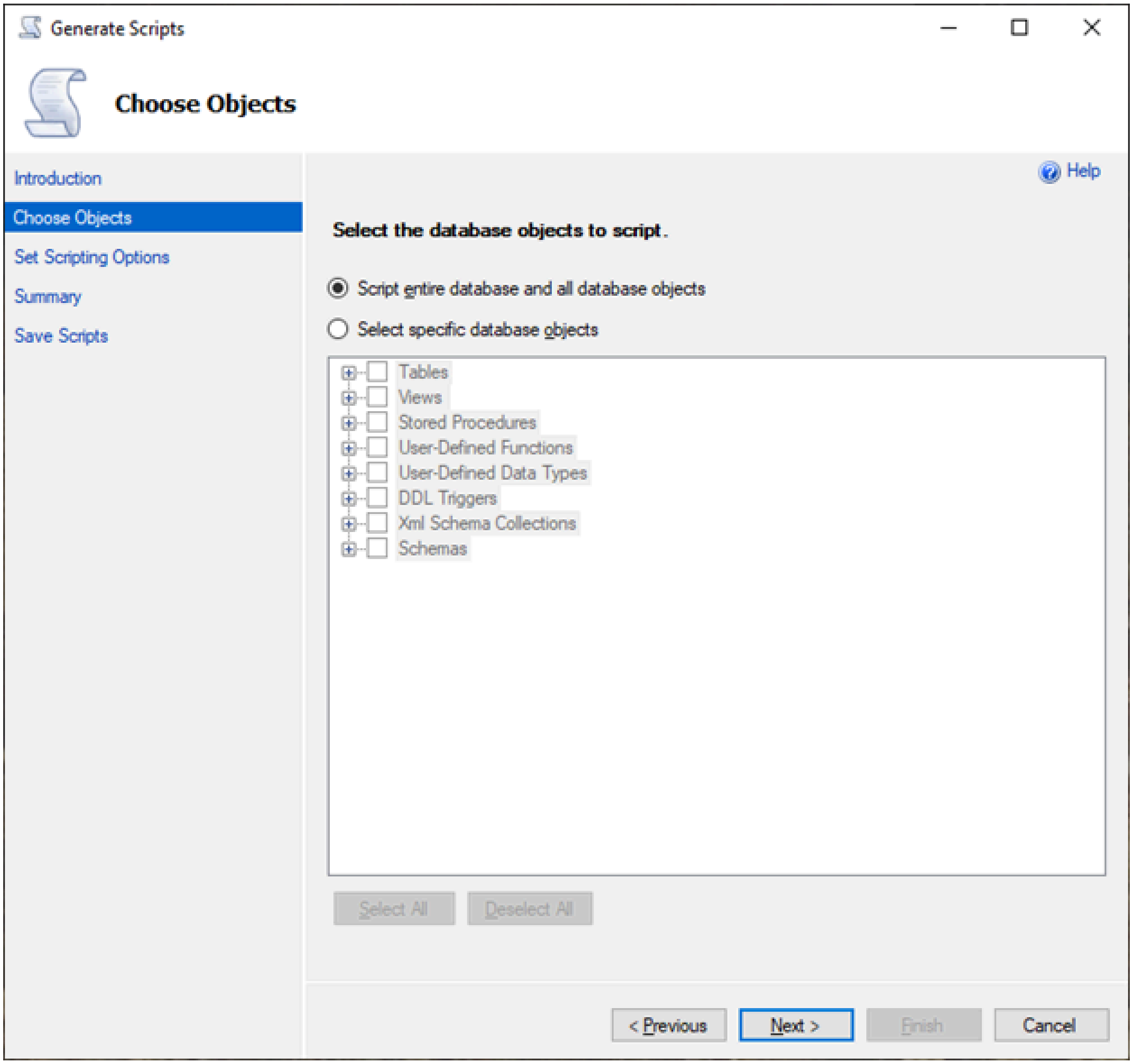
최상의 결과를 얻으려면 SQL Server 데이터베이스 애플리케이션을 Babelfish로 실제로 마이그레이션하기 전에 생성된 DDL/DML 및 클라이언트 쿼리 코드를 평가하는 것이 좋습니다. Babelfish의 버전과 애플리케이션에서 구현하는 SQL Server의 특정 기능에 따라 애플리케이션을 리팩터링하거나 Babelfish에서 아직 완전히 지원되지 않는 기능에 대한 대안을 사용해야 할 수 있습니다.

- SQL Server 애플리케이션 코드를 평가하려면, 생성된 DDL에서 Babelfish Compass를 사용하여 Babelfish에서 얼마나 많은 T-SQL 코드를 지원하는지 확인해야 합니다. Babelfish에서 실행하기 전에 수정이 필요할 수 있는 T-SQL 코드를 식별합니다. 이 도구에 대한 자세한 내용은 GitHub의 Babelfish Compass 도구를 참조하세요.

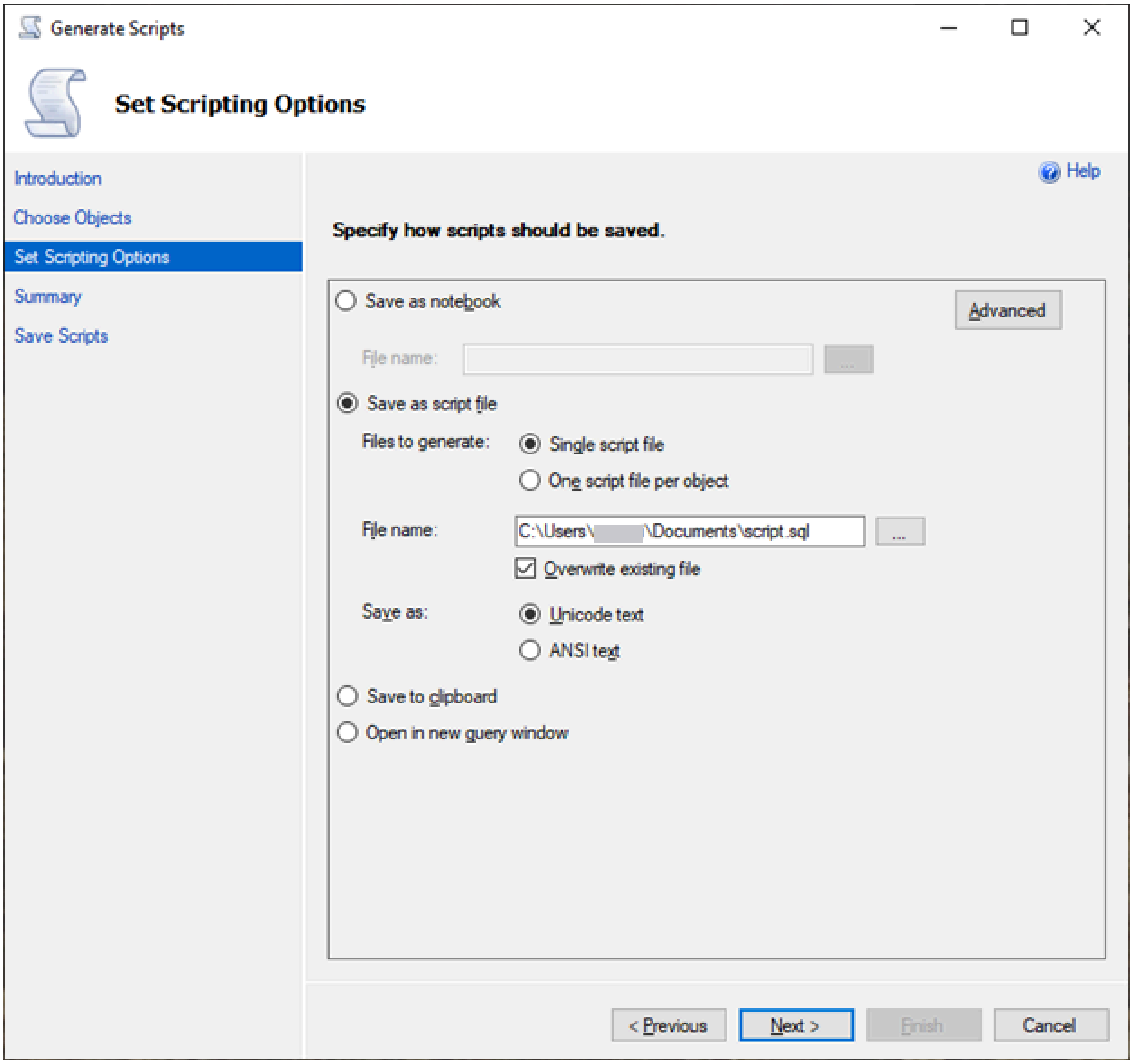
  > **Note**
  >
  > Babelfish Compass는 오픈 소스 도구입니다. AWS Support 대신 GitHub를 통해 Babelfish Compass와 관련된 모든 문제를 보고하십시오.

SQL Server Management Studio(SSMS)와 함께 스크립트 생성 마법사를 사용하여, Babelfish Compass 또는 AWS Schema Conversion Tool CLI에서 평가하는 SQL 파일을 생성할 수 있습니다. 다음 단계를 따라 평가를 간소화하는 것이 좋습니다.

1. Choose Objects(객체 선택) 페이지에서 Script entire database and all database objects(전체 데이터베이스 및 모든 데이터베이스 객체 스크립팅)를 선택합니다.

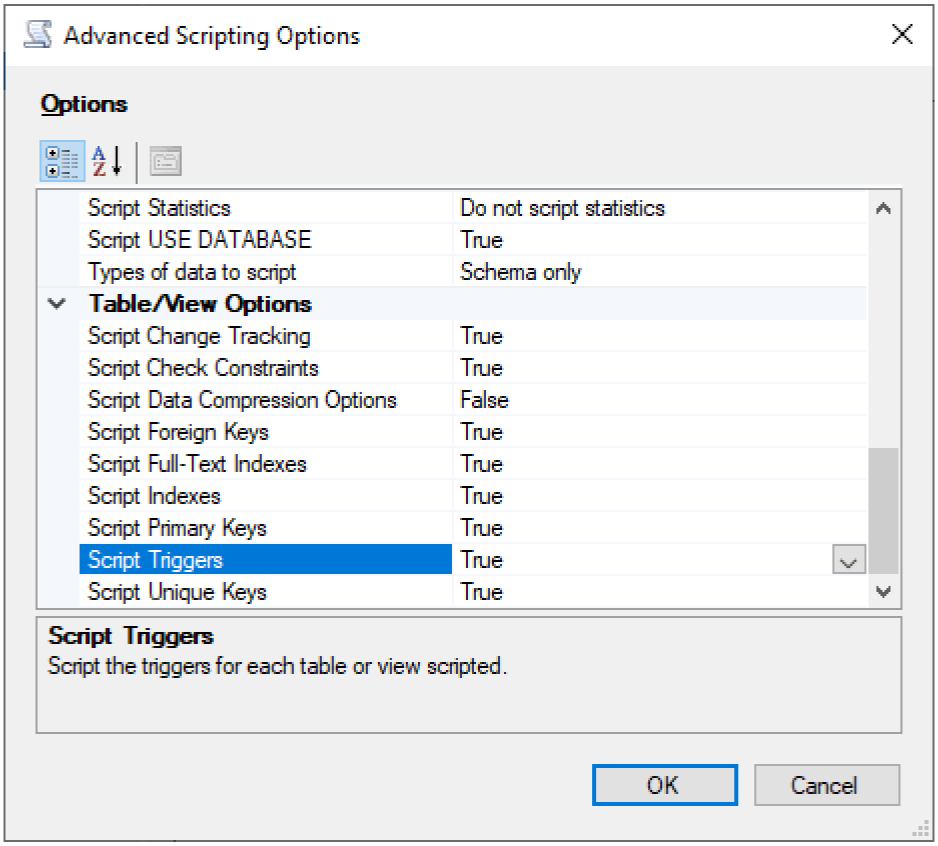


2. Set Scripting Options(스크립팅 옵션 설정)에서 Save as script file(스크립트 파일로 저장)을 Single script file(단일 스크립트 파일)로 선택합니다.



3. 전체 평가에서 일반적으로 false로 설정되는 기능을 식별하도록 기본 스크립팅 옵션을 변경하려면 Advanced(고급)를 선택합니다.

- Script Change Tracking(스크립트 변경 사항 트래킹)을 True로
- Script Full-Text Indexes(스크립트 전체 텍스트 인덱스)를 True로
- Script Triggers(스크립트 트리거)를 True롤
- Script Logins(스크립트 로그인)를 True로
- Script Owner(스크립트 소유자)를 True로
- Script Object-Level Permissions(스크립트 객체 수준 권한)를 True로
- Script Collations(스크립트 콜레이션)를 True로



4. 마법사의 나머지 단계를 수행하여 파일을 생성합니다.

SQL Server에서 Babelfish로 마이그레이션하기 위한 가져오기/내보내기 도구

SQL Server에서 Babelfish로 마이그레이션하는 기본 도구로 AWS DMS를 사용하는 것이 좋습니다. 그러나 Babelfish는 다음을 포함한 SQL Server 도구를 사용하여 데이터를 마이그레이션하는 여러 다른 방법을 지원합니다.

- 모든 Babelfish 버전을 위한 SQL Server Integration Services(SSIS) 자세한 내용은 SSIS 및 Babelfish를 사용하여 SQL Server에서 Aurora PostgreSQL로 마이그레이션을 참조하세요.
- Babelfish 버전 2.1.0 이상에서는 SSMS 가져오기/내보내기 마법사를 사용합니다. 이 도구는 SSMS를 통해 사용할 수 있지만, 독립 실행형 도구로도 지원됩니다. 자세한 내용은 Microsoft 설명서의 SQL Server 가져오기 및 내보내기 마법사 소개를 참조하세요.
- Microsoft 일괄 데이터 복사 프로그램(bcp) 유틸리티를 사용하면 데이터를 Microsoft SQL Server 인스턴스에서 지정한 형식의 데이터 파일로 복사할 수 있습니다. 자세한 내용은 Microsoft 설명서의 bcp 유틸리티를 참조하세요. 이제 Babelfish는 BCP 클라이언트를 사용한 데이터 마이그레이션을 지원하고, bcp 유틸리티는 -E 플래그(ID 열용) 및 -b 플래그(일괄 삽입용)를 지원합니다. -C, -T, -G, -K, -R, -V 및 -h를 포함한 특정 bcp 옵션은 지원되지 않습니다.

SQL Server Management Studio(SSMS)를 사용하여 Babelfish로 마이그레이션

각 특정 객체 유형별로 별도의 파일을 생성하는 것이 좋습니다. 먼저 각 DDL 문 세트에 대해 SSMS의 스크립트 생성 마법사를 사용한 다음, 객체를 그룹으로 수정하여 평가 중에 발견된 문제를 해결할 수 있습니다.

AWS DMS 또는 다른 데이터 마이그레이션 방법을 사용하여 데이터를 마이그레이션하려면 다음 단계를 수행하십시오. 이러한 생성 스크립트 유형을 먼저 실행하면 Aurora PostgreSQL의 Babelfish 테이블에 데이터를 더 빠르고 효율적으로 로드할 수 있습니다.

1. `CREATE SCHEMA` 문을 실행합니다.
2. `CREATE TYPE` 문을 실행하여 사용자 정의 데이터 유형을 생성합니다.
3. 기본 키 또는 고유한 제약 조건을 사용하여 기본 `CREATE TABLE` 문을 실행합니다.

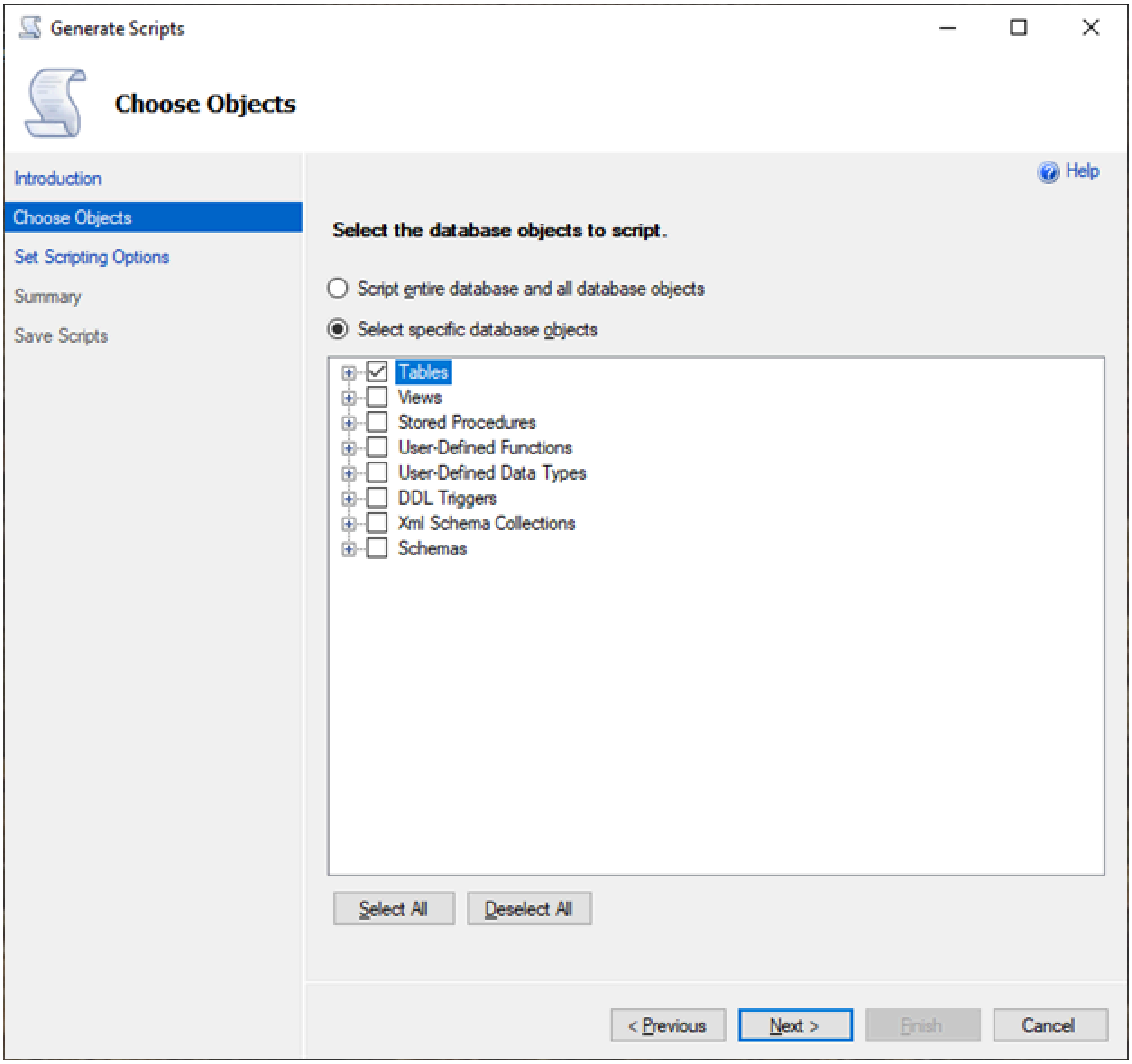
권장되는 가져오기/내보내기 도구를 사용하여 데이터 로드를 수행합니다. 다음 단계에 대해 수정된 스크립트를 실행하여 나머지 데이터베이스 객체를 추가합니다. 제약 조건, 트리거 및 인덱스를 대상으로 이러한 스크립트를 실행하려면 create table 문이 필요합니다. 스크립트가 생성되면 create table 문을 삭제합니다.

1. 검사 제약 조건, 외래 키 제약 조건, 기본 제약 조건을 대상으로 `ALTER TABLE` 문을 실행합니다.
2. `CREATE TRIGGER` 문을 실행합니다.
3. `CREATE INDEX` 문을 실행합니다.
4. `CREATE VIEW` 문을 실행합니다.
5. `CREATE STORED PROCEDURE` 문을 실행합니다.

각 객체 유형에 대한 스크립트를 생성하는 방법

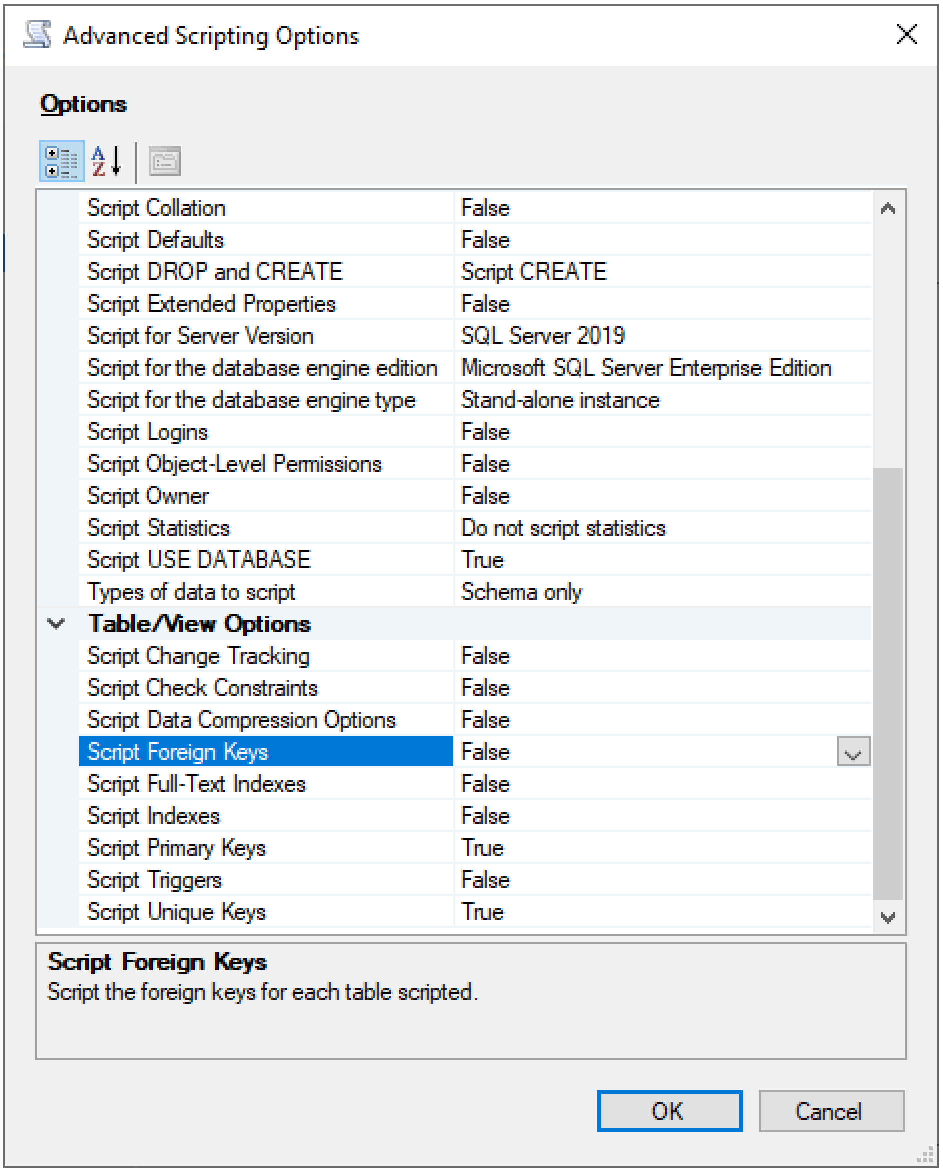
다음 단계를 수행하여, SSMS에서 스크립트 생성 마법사를 사용하여 기본 create table 문을 만듭니다. 동일한 단계를 수행하여 다양한 객체 유형에 대한 스크립트를 생성합니다.

1. 기존 SQL Server 인스턴스에 연결합니다.
2. 데이터베이스 이름의 컨텍스트 메뉴를 엽니다(마우스 오른쪽 버튼 클릭).
3. Tasks(태스크)를 선택한 다음 Generate Scripts...(스크립트 생성)를 선택합니다.
4. 객체 선택(Choose Objects) 창에서 특정 데이터베이스 객체 선택(Select specific database objects)을 선택합니다. 테이블(Tables)을 선택하고 모든 테이블을 선택합니다. 다음을 선택하여 계속 진행합니다.



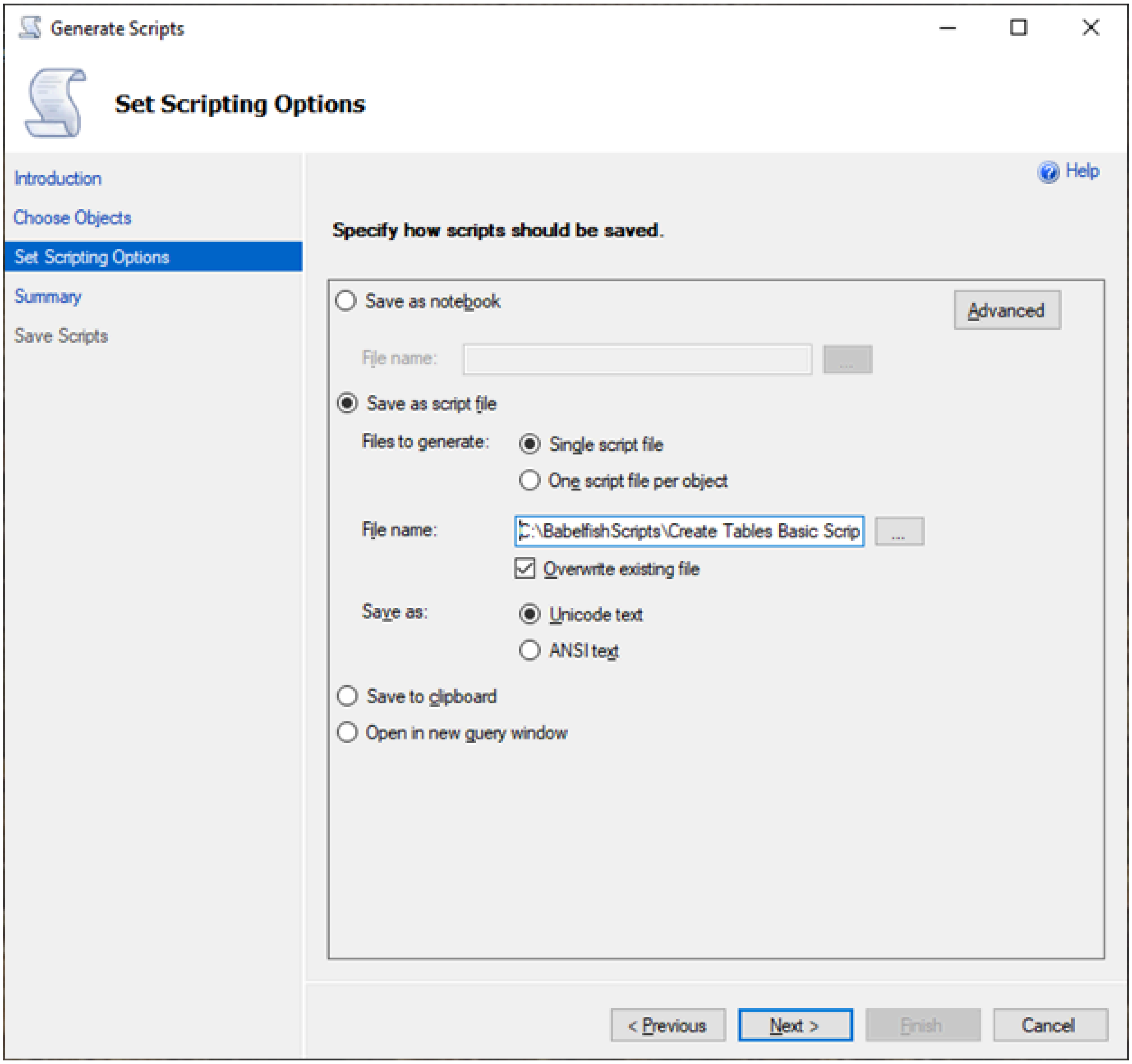
5. 스크립팅 옵션 설정(Set Scripting Options) 페이지에서 고급(Advanced)을 선택하여 옵션(Options) 설정을 엽니다. 기본 create table 명령문을 생성하려면 다음 기본값을 변경해야 합니다.

   - Script Defaults(스크립트 기본값)를 False로 변경합니다.
   - Script Extended Properties(스크립트 확장 속성)을 False로 변경합니다. Babelfish는 확장 속성을 지원하지 않습니다.
   - Script Check Constraints(스크립트 확인 제약 조건)를 False로 변경합니다. Foreign Keys(외래 키)를 False로 변경합니다.



6. 확인을 선택합니다.

7. Set Scripting Options(스크립팅 옵션 설정)에서 Save as script file(스크립트 파일로 저장)을 선택하고 Single script file(단일 스크립트 파일) 옵션을 선택합니다. File name(파일 이름)을 입력합니다.



8. Next(다음)를 선택하여 Summary wizard(요약 마법사) 페이지를 확인합니다.
9. Next(다음)를 선택하여 스크립트 생성을 시작합니다.

   마법사에서 다른 객체 유형에 대한 스크립트를 계속 생성할 수 있습니다. 파일을 저장되면 Finish(완료)를 선택하는 대신 Previous(이전) 버튼을 세 번 선택하여 Choose Objects(객체 선택) 페이지로 돌아갑니다. 그런 다음 마법사에서 단계를 반복하여 다른 객체 유형에 대한 스크립트를 생성합니다.

Babelfish for Aurora PostgreSQL을 사용하는 데이터베이스 인증

Babelfish for Aurora PostgreSQL은 데이터베이스 사용자를 인증하는 두 가지 방법을 지원합니다. 암호 인증은 기본적으로 모든 Babelfish DB 클러스터에 사용할 수 있습니다. 같은 DB 클러스터에 대해 Kerberos 인증을 추가할 수도 있습니다.

주제

Babelfish를 사용하는 비밀번호 인증

Babelfish for Aurora PostgreSQL은 암호 인증을 지원합니다. 암호는 디스크에 암호화된 형태로 저장됩니다. Aurora PostgreSQL 클러스터의 인증에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 사용한 보안 섹션을 참조하세요.

Babelfish에 연결할 때마다 자격 증명을 묻는 메시지가 표시될 수 있습니다. Aurora PostgreSQL로 마이그레이션되거나 생성된 모든 사용자는 SQL Server 포트와 PostgreSQL 포트 모두에서 동일한 자격 증명을 사용할 수 있습니다. Babelfish는 암호 정책을 시행하지 않지만, 다음을 수행하는 것이 좋습니다.

- 8자 이상의 복잡한 암호여야 합니다.
- 암호 만료 정책을 적용합니다.

전체 데이터베이스 사용자 목록을 검토하려면 `SELECT * FROM pg_user;` 명령을 사용합니다.

Babelfish를 사용하는 Kerberos 인증

Babelfish for Aurora PostgreSQL 15.2 버전은 Kerberos를 사용하여 DB 클러스터에 대한 인증을 지원합니다. 이 방법은 사용자가 Babelfish 데이터베이스에 접속하려고 할 때 Windows 인증을 사용하여 사용자를 인증할 수 있도록 합니다. 이를 위해서는 먼저 Kerberos 인증을 위한 AWS Directory Service for Microsoft Active Directory를 사용하도록 DB 클러스터를 구성해야 합니다. 자세한 내용은 AWS Directory Service 관리 가이드의 AWS Directory Service란 무엇입니까?를 참조하세요.

Kerberos 인증 설정

Babelfish for Aurora PostgreSQL DB 클러스터는 서로 다른 두 개의 포트를 사용하여 연결할 수 있지만 Kerberos 인증 설정은 한 번만 수행하면 됩니다. 따라서 먼저 DB 클러스터에 대한 Kerberos 인증을 설정해야 합니다. 자세한 내용은 Kerberos 인증 설정을 참조하세요. 설정을 완료한 후에는 Kerberos를 사용하여 PostgreSQL 클라이언트에 연결할 수 있는지 확인합니다. 자세한 내용은 Kerberos 인증을 사용하여 연결을 참조하세요.

Babelfish의 로그인 및 사용자 프로비저닝

Tabular Data Stream(TDS) 포트에서 생성된 Windows 로그인은 TDS 포트 또는 PostgreSQL 포트와 함께 사용할 수 있습니다. 먼저 인증에 Kerberos를 사용할 수 있는 로그인이 T-SQL 사용자 및 애플리케이션이 Babelfish 데이터베이스에 연결하는 데 사용되기 전에 TDS 포트에서 프로비저닝되어야 합니다. Windows 로그인을 생성할 경우 관리자는 DNS 도메인 이름 또는 NetBIOS 도메인 이름을 사용하여 로그인을 제공할 수 있습니다. 일반적으로 NetBIOS 도메인은 DNS 도메인 이름의 하위 도메인입니다. 예를 들어 DNS 도메인 이름이 `CORP.EXAMPLE.COM`인 경우 NetBIOS 도메인은 `CORP`일 수 있습니다. 로그인을 위해 NetBIOS 도메인 이름 형식을 제공할 경우 DNS 도메인 이름에 대한 매핑이 있어야 합니다.

DNS 도메인 이름 매핑에 대한 NetBIOS 도메인 이름 관리

NetBIOS 도메인 이름과 DNS 도메인 이름 간의 매핑을 관리하기 위해 Babelfish는 매핑을 추가, 제거, 잘라내기할 수 있는 시스템 저장 프로시저를 제공합니다. `sysadmin` 역할이 있는 사용자만 이러한 프로시저를 실행할 수 있습니다.

NetBIOS와 DNS 도메인 이름 간에 매핑을 생성하려면 Babelfish에서 제공하는 시스템 저장 프로시저인 `babelfish_add_domain_mapping_entry`를 사용하세요. 두 인수 모두 유효한 값이 있어야 하고 NULL이 아니어야 합니다.

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

다음 예제에서는 NetBIOS 이름인 CORP와 DNS 도메인 이름인 CORP.EXAMPLE.COM 간에 매핑을 생성하는 방법을 보여 줍니다.

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

기존 매핑 항목을 삭제하려면 시스템 저장 프로시저 babelfish_remove_domain_mapping_entry를 사용하세요.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

다음 예제는 NetBIOS 이름인 CORP에 대한 매핑을 제거하는 방법을 보여 줍니다.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

기존 매핑 항목을 제거하려면 시스템 저장 프로시저 babelfish_truncate_domain_mapping_table을 사용하세요.

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

NetBIOS와 DNS 도메인 이름 간의 모든 매핑을 보려면 다음 쿼리를 사용합니다.

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

로그인 관리

로그인 생성

올바른 권한이 있는 로그인을 사용하여 TDS 엔드포인트를 통해 DB에 연결합니다. 로그인을 위해 생성된 데이터베이스 사용자가 없는 경우 로그인은 게스트 사용자에게 매핑됩니다. 게스트 사용자가 활성화되지 않은 경우 로그인 시도가 실패합니다.

다음 쿼리를 사용하여 Windows 로그인을 생성합니다. 이 `FROM WINDOWS` 옵션을 사용하면 Active Directory를 사용하여 인증할 수 있습니다.

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

다음 예시에서는 기본 데이터베이스인 db1을 사용하여 Active Directory 사용자 [corp\ test1]에 대한 로그인을 생성하는 방법을 보여줍니다.

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

이 예시에서는 NetBIOS 도메인인 CORP와 DNS 도메인 이름인 CORP.EXAMPLE.COM 간에 매핑이 있다고 가정합니다. 매핑이 없는 경우 DNS 도메인 이름 [CORP.EXAMPLE.COM\test1]을 제공해야 합니다.

> **Note**
> Active Directory 사용자를 기준으로 한 로그인은 21자 미만의 이름으로 제한됩니다.

로그인 삭제

로그인을 삭제하려면 다음 예시에 나온 것처럼 로그인에 사용된 것과 동일한 구문을 사용하세요.

```
DROP LOGIN [DNS domain name\login]
```

로그인 변경

로그인을 변경하려면 다음 예시에 나온 것처럼 로그인에 사용된 것과 동일한 구문을 사용하세요.

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

ALTER LOGIN 명령은 다음을 포함하여 Windows 로그인을 위한 제한된 옵션을 지원합니다.

- DISABLE - 로그인을 비활성화합니다. 비활성화된 로그인은 인증에 사용할 수 없습니다.
- ENABLE - 비활성화된 로그인을 활성화합니다.
- DEFAULT_DATABASE - 로그인의 기본 데이터베이스를 변경합니다.

> **Note**
> 모든 암호 관리는 AWS Directory Service를 통해 수행되므로, ALTER LOGIN 명령을 사용해도 데이터베이스 관리자는 Windows 로그인에 대한 암호를 변경하거나 설정할 수 없습니다.

Kerberos 인증을 사용하여 Babelfish for Aurora PostgreSQL에 연결

일반적으로 Kerberos를 사용하여 인증하는 데이터베이스 사용자는 클라이언트 시스템에서 인증을 수행합니다. 이러한 시스템은 Active Directory 도메인의 멤버입니다. 이들은 클라이언트 애플리케이션에서 Windows 인증을 사용하여 TDS 포트의 Babelfish for Aurora PostgreSQL 서버에 액세스합니다.

Kerberos 인증을 사용하여 PostgreSQL 포트에서 Babelfish for Aurora PostgreSQL에 연결

TDS 포트에서 생성된 로그인은 TDS 포트 또는 PostgreSQL 포트와 함께 사용할 수 있습니다. 그러나 PostgreSQL은 기본적으로 사용자 이름에 대소문자를 구분하는 비교를 사용합니다. Aurora PostgreSQL이 Kerberos 사용자 이름을 대소문자를 구분하지 않는 것으로 해석하려면 사용자 지정 Babelfish 클러스터 파라미터 그룹에서 `krb_caseins_users` 파라미터를 `true`로 설정해야 합니다. 이 파라미터는 기본적으로 `false`로 설정되어 있습니다. 자세한 내용은 대소문자를 구분하지 않는 사용자 이름 구성을 참조하세요. 또한 PostgreSQL 클라이언트 애플리케이션에서 <login@DNS 도메인 이름> 형식으로 로그인 사용자 이름을 지정해야 합니다. <DNS 도메인 이름\login> 형식은 사용할 수 없습니다.

자주 발생하는 오류

온프레미스 Microsoft Active Directory와 AWS Managed Microsoft AD 간에 포리스트 트러스트 관계를 설정할 수 있습니다. 자세한 내용은 신뢰 관계 생성을 참조하세요. 그런 다음, 호스트 엔드포인트에서 Amazon 도메인 `rds.amazonaws.com`을 사용하는 대신 특수 도메인별 엔드포인트를 사용하여 연결해야 합니다. 올바른 도메인별 엔드포인트를 사용하지 않으면 다음과 같은 오류가 발생할 수 있습니다.

```
Error: "Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

이 오류는 제공된 엔드포인트 URL에 대한 서비스 티켓을 TDS 클라이언트가 캐싱할 수 없을 때 발생합니다. 자세한 내용은 Kerbero를 사용하여 연결을 참조하세요.

Babelfish용 Active Directory 보안 그룹을 사용하여 Kerberos 인증 설정

Babelfish 버전 4.2.0부터 Active Directory 보안 그룹을 사용하여 Babelfish에 대한 Kerberos 인증을 설정할 수 있습니다. Active Directory를 사용하여 Kerberos 인증을 설정하려면 완료해야 하는 사전 조건은 다음과 같습니다.

- Babelfish를 사용하는 Kerberos 인증에 언급된 모든 단계를 따라야 합니다.
- DB 인스턴스가 Active Directory와 연결되어 있는지 확인합니다. 콘솔에서 또는 describe-db-instances AWS CLI 명령을 실행하여 도메인 구성원 자격 상태를 보고 확인할 수 있습니다.

  DB 인스턴스의 상태가 kerberos를 지원해야 합니다. 도메인 구성원 자격 이해에 대한 자세한 내용은 도메인 멤버십 이해 섹션을 참조하세요.
- 다음 쿼리를 사용하여 NetBIOS 도메인 이름과 DNS 도메인 이름 간의 매핑을 확인합니다.

  ```
  SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
  ```
- 계속 진행하기 전에 개별 로그인을 사용한 Kerberos 인증이 예상대로 작동하는지 확인합니다. Active Directory 사용자로서 Kerberos 인증을 사용한 연결이 성공해야 합니다. 문제가 발생하는 경우 자주 발생하는 오류 섹션을 참조하세요.

pg_ad_mapping 확장 설정

pg_ad_mapping 확장 설정에 언급된 모든 단계를 따라야 합니다. 확장 프로그램이 설치되었는지 확인하려면 TDS 엔드포인트에서 다음 쿼리를 실행합니다.

```
1> SELECT extname, extversion FROM pg_extension where extname like 'pg_ad_mapping';
2> GO
extname       extversion
------------- ----------
pg_ad_mapping 0.1

(1 rows affected)
```

그룹 로그인 관리

로그인 관리에 설명된 단계에 따라 그룹 로그인을 생성합니다. 필수는 아니지만, 유지 관리가 용이하도록 로그인 이름을 Active Directory(AD) 보안 그룹 이름과 동일하게 사용하는 것이 좋습니다. 예시:

```
CREATE LOGIN [corp\accounts-group] FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

T-SQL 그룹 로그인과 AD 보안 그룹 매핑

데이터베이스 서버에 대한 액세스가 필요한 각 AD 보안 그룹에 대해 T-SQL Windows 그룹 로그인을 명시적으로 프로비저닝해야 합니다. 하나 이상의 프로비저닝된 AD 보안 그룹에 속하는 AD 사용자는 데이터베이스 서버에 액세스할 수 있습니다.

> **Note**
>
> 이 T-SQL 로그인은 더 이상 암호 기반 인증을 사용하여 인증할 수 없습니다.

예를 들어, accounts-group은 AD의 보안 그룹이며 Babelfish에서 이 보안 그룹을 프로비저닝하려는 경우 [corp\accounts-group] 형식을 사용해야 합니다.

- AD 보안 그룹: accounts-group
- TSQL 로그인: [corp\accounts-group]
- 지정된 TSQL 로그인에 해당하는 PG 역할: accounts-group@CORP.EXAMPLE.COM

이제 관리자는 다음 psql 명령을 통해 PostgreSQL 엔드포인트에서 AD 보안 그룹과 T-SQL 로그인 간의 매핑을 생성할 수 있습니다. 함수 사용에 대한 자세한 내용은 `pg_ad_mapping` 확장의 함수 사용 섹션을 참조하세요.

> **Note**
>
> 매핑을 추가할 때 T-SQL 로그인을 login_name@FQDN 형식으로 지정해야 합니다. TDS 엔드포인트를 통해 연결할 때는 가중치가 무시됩니다. 가중치 사용에 대한 자세한 내용은 PostgreSQL 포트의 PostgreSQL 엔드포인트를 통해 Babelfish에 연결 섹션을 참조하세요.

```
postgres=>select pgadmap_set_mapping('accounts-group', 'accounts-
group@CORP.EXAMPLE.COM', <SID>, <Weight>);
```

AD 보안 그룹의 SID 검색에 대한 자세한 내용은 PowerShell에서 Active Directory 그룹 SID 검색 섹션을 참조하세요.

다음 표에는 AD 보안 그룹에서 T-SQL 로그인으로의 샘플 매핑이 나와 있습니다.

AD 보안 그룹	TSQL 로그인	지정된 TSQL 로그인에 해당하는 PG 역할	가중치
accounts-group	[corp\accounts-group]	accounts-group@CORP.EXAMPLE.COM	7
sales-group	[corp\sales-group]	sales-group@CORP.EXAMPLE.COM	10
dev-group	[corp\dev-group]	dev-group@CORP.EXAMPLE.COM	7

```
postgres=> select admap.ad_sid, admap.ad_grp, lgn.orig_loginname, lgn.rolname,
 admap.weight from pgadmap_read_mapping() as admap, sys.babelfish_authid_login_ext as
 lgn where admap.pg_role = lgn.rolname;
    ad_sid    |     ad_grp      |    orig_loginname    |              rolname
 | weight
--------------+----------------+---------------------+---------------------------------
+--------
 S-1-5-67-890 | accounts-group | corp\accounts-group  | accounts-group@CORP.EXAMPLE.COM
 |      7
 S-1-2-34-560 | sales-group    | corp\sales-group     | sales-group@CORP.EXAMPLE.COM
 |      10
 S-1-8-43-612 | dev-group      | corp\dev-group       | dev-group@CORP.EXAMPLE.COM
 |      7
 (7 rows)
```

TDS 엔드포인트를 통해 Babelfish에 연결

다음 예제에서 user1은 accounts-group 및 sales-group의 구성원이고, user2는 accounts-group 및 dev-group의 구성원입니다.

사용자 이름	AD 보안 그룹 구성원 자격
user1	accounts-group, sales-group
user2	accounts-group, dev-group

sqlcmd 유틸리티를 사용하여 Babelfish 데이터베이스 서버에 연결합니다. 아래 예제에 따라 사용자(이 예제에서는 user1)가 Kerberos를 사용하여 인증되었는지 확인할 수 있습니다.

```
1> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();
2>  GO
principal              gss_authenticated
---------------------- -----------------
user1@CORP.EXAMPLE.COM  1

((1 rows affected))
1> select suser_name();
2>  GO
suser_name
----------
corp\user1

(1 rows affected)
```

이 예제에서 user1은 accounts-group 및 sales-group의 권한을 상속합니다. `sys.login_token` 시스템 보기를 사용하여 그룹 구성원 자격을 확인할 수 있습니다.

```
1> SELECT name, type FROM sys.login_token;
2>  GO
name                type
------------------- ----
corp\accounts-group WINDOWS GROUP
corp\sales-group    WINDOWS GROUP
```

```
(2 rows affected)
```

감사 및 로깅

AD 보안 주체 ID를 확인하려면 아래 명령을 사용하세요.

```
1> select suser_name();
2> GO
suser_name
----------
corp\user1

(1 rows affected)
```

현재로서는 AD 사용자 ID가 로그에 표시되지 않습니다. `log_connections` 파라미터를 설정하여 DB 세션 설정을 로깅할 수 있습니다. 자세한 내용은 log_connections를 참조하세요. 다음 예제와 같이 출력에 AD 사용자 ID가 보안 주체로 포함됩니다. 이 출력과 관련된 백엔드 PID는 실제 AD 사용자에게 작업을 귀속시키는 데 도움이 됩니다.

```
bbf_group_ad_login@babelfish_db:[615]:LOG: connection authorized:
 user=bbf_group_ad_login database=babelfish_db application_name=sqlcmd GSS
 (authenticated=yes, encrypted=yes, principal=user1@CORP.EXAMPLE.COM)
```

AD 보안 그룹 구성원 자격 권한 활용

서버 수준 권한 상속

지정된 AD 보안 그룹의 구성원인 AD 사용자는 매핑된 Windows 그룹 로그인에 부여된 서버 수준 권한을 상속합니다. Babelfish의 `sysadmin` 서버 역할에 대한 구성원 자격이 부여된 `accounts-group` AD 보안 그룹을 예로 들어 보겠습니다. 다음 명령을 사용하여 서버 수준 권한을 상속할 수 있습니다.

```
1> ALTER SERVER ROLE sysadmin ADD MEMBER [corp\accounts-group];
```

따라서 `accounts-group` AD 보안 그룹의 구성원인 Active Directory 사용자는 `sysadmin` 역할과 관련된 서버 수준 권한을 상속받게 됩니다. 즉, `accounts-group`의 구성원인 `corp\user1`과 같은 사용자도 이제 Babelfish 내에서 서버 수준 작업을 수행할 수 있습니다.

> **Note**
>
> 서버 수준 DDL을 수행하려면 개별 AD 사용자에 대한 Windows 로그인이 있어야 합니다. 자세한 내용은 제한 사항 단원을 참조하십시오.

데이터베이스 수준 권한 상속

데이터베이스 수준 권한을 부여하려면 Windows 그룹 로그인으로 데이터베이스 사용자를 만들고 매핑해야 합니다. 지정된 AD 보안 그룹의 구성원인 AD 사용자는 해당 데이터베이스 사용자에게 부여된 데이터베이스 수준 권한을 상속합니다. 다음 예제에서는 Windows 그룹 [corp\accounts-group]에 대해 데이터베이스 수준 권한이 할당되는 방식을 확인할 수 있습니다.

```
1> CREATE DATABASE db1;
2> GO
1> USE db1;
2> GO
Changed database context to 'db1'.
1> CREATE TABLE dbo.t1(a int);
2> GO
```

Windows 그룹 로그인 [corp\accounts-group]을 위한 데이터베이스 사용자 [corp\sales-group]을 생성합니다. 이 단계를 수행하려면 sysadmin의 구성원인 로그인을 사용하여 TDS 엔드포인트를 통해 연결하세요.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group];
2> GO
```

이제 AD 사용자 user1으로 연결하여 표 t1의 액세스 권한을 확인합니다. 아직 데이터베이스 수준 권한을 부여하지 않았으므로, 권한 거부 오류가 발생합니다.

```
1> SELECT * FROM dbo.t1;
2> GO
Msg 33557097, Level 16, State 1, Server db-inst, Line 1
permission denied for table t1
```

데이터베이스 사용자 [corp\accounts-group]에게 표 t1의 SELECT 권한을 부여하세요. 이 단계를 수행하려면 sysadmin의 구성원인 로그인을 사용하여 TDS 엔드포인트를 통해 연결하세요.

```
1> GRANT SELECT ON dbo.t1 TO [corp\accounts-group];
2> GO
```

AD 사용자 user1으로 연결하여 액세스를 검증합니다.

```
1> SELECT * FROM dbo.t1;
2> GO
a
-----------

(0 rows affected)
```

기본 또는 명시적 스키마를 기반으로 DDL 문 동작 처리

AD 인증 세션을 사용하는 경우 현재 세션의 기본 스키마는 다음 조건에 따라 결정됩니다.

- 개별 데이터베이스 사용자가 있는 경우 사용자의 기본 스키마가 현재 세션의 기본 스키마로 간주됩니다.
- 그룹 데이터베이스 사용자의 기본 스키마가 있는 경우 그룹 데이터베이스 사용자의 기본 스키마는 보안 주체 ID가 최소인 현재 세션의 기본 스키마로 간주됩니다.

CREATE DDL 문 동작에 대한 이해

CREATE DDL 문에 명시적 스키마가 지정되지 않은 경우 현재 세션의 기본 스키마에서 객체 생성이 수행됩니다. 스키마가 기본값으로든 명시적으로든 확인할 수 없는 경우 DDL 문에서 다음 오류가 발생합니다.

```
"Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command."
```

Example : Windows 그룹 사용자의 기본 스키마가 존재하지 않습니다.

Windows 그룹 사용자 [corp\accounts-group]의 기본 스키마가 NULL이고 AD 사용자 user1이 스키마를 명시적으로 지정하지 않고 DDL을 수행하려고 합니다. user1에는 개별 Windows 로그인 및 사용자가 존재하지 않으므로, Windows 그룹 사용자 [corp\accounts-group]의 데이터베이스 수준 권한만 갖게 됩니다.

```
1> create TABLE t2(a int);
2> GO

Msg 33557097, Level 16, State 1, Server db-inst, Line 1
Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command.
```

> **Note**
>
> AD 사용자 user1에는 개별 Windows 로그인 및 사용자가 존재하지 않습니다.

Example : Windows 그룹 사용자의 기본 스키마가 있습니다.

sysadmin을 사용하여 기본 스키마를 통해 [corp\accounts-group] 로그인용 Windows 그룹 사용자를 생성합니다.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group] WITH
 DEFAULT_SCHEMA = sch_acc;
2> GO
1> CREATE SCHEMA sch_acc AUTHORIZATION [gad\accounts-group];
2> GO
1> SELECT name, principal_id, default_schema_name FROM sys.database_principals WHERE
 name = 'corp\accounts-group';
2> GO

name                  principal_id default_schema_name
```

```
------------------ ------------ -------------------
corp\accounts-group 24162        sch_acc

(1 rows affected)
```

AD 사용자 user1을 사용하여 스키마를 명시적으로 지정하지 않고 객체를 만들어 보세요. 표 t2는 [corp \accounts-group] Windows 그룹 사용자의 기본 스키마에서 생성됩니다. 이 객체의 소유자는 sch_acc 스키마의 소유자와 동일합니다.

```
1> CREATE TABLE t_group(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't_group';
2> GO

name    schema_name
------- -----------
t_group sch_acc

(1 rows affected)
```

Note

AD 사용자 user1에는 개별 Windows 로그인 및 사용자가 존재하지 않습니다.

Example : AD 사용자에 대해 개별 데이터베이스 사용자도 있습니다.

AD 사용자에 대해 개별 데이터베이스 사용자도 존재하는 경우 객체는 항상 개별 데이터베이스 사용자와 연결된 스키마에 생성됩니다. 데이터베이스 사용자에 대한 스키마가 없는 경우 dbo 스키마가 사용됩니다. AD 사용자 user1을 위한 개별 Windows 로그인 및 데이터베이스 사용자를 생성합니다. sysadmin 로그인을 사용하여 TDS 엔드포인트를 통해 연결

```
1> CREATE LOGIN [corp\user1] FROM WINDOWS;
2> GO
1> CREATE USER [corp\user1] FOR LOGIN [corp\user1] WITH DEFAULT_SCHEMA = sch1;
```

```
2> GO
1> CREATE SCHEMA sch1 AUTHORIZATION [corp\user1];
2> GO
1> SELECT name, default_schema_name FROM sys.database_principals WHERE name = 'corp
\user1';
2> GO

name      default_schema_name
--------- -------------------
corp\user1 sch1

(1 rows affected)
```

AD 사용자 user1을 사용하여 연결하고, 스키마를 명시적으로 지정하지 않고 객체를 만들어 보세요. 표 t2는 스키마 sch1에 생성됩니다. 또한 이 객체의 소유자는 스키마 sch1의 소유자와 동일하다는 점에 유의하세요.

```
1> CREATE TABLE t2(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't2';
2> GO

name schema_name
---- -----------
t2   sch1

(1 rows affected)
```

제한 사항

- Dump/Restore 유틸리티는 pg_ad_mapping 확장 매핑 덤프를 지원하지 않습니다. 복원 후에는 이러한 매핑을 다시 생성해야 합니다.
- pg_ad_mapping을 사용하는 Babelfish 및 Aurora PostgreSQL 인스턴스에는 블루-그린 배포가 지원되지 않습니다.
- 암시적 스키마 생성은 지원되지 않습니다. 암시적 스키마 생성이 필요한 DDL 문은 지원되지 않습니다.

- 서버 수준의 DDL인 ALTER AUTHORIZATION ALTER AUTHORIZATION ON DATABASE , CREATE DATABASE, CREATE LOGIN, ALTER LOGIN, ALTER SERVER ROLE, ALTER DATABASE는 개별 Windows 로그인이 없고 그룹 Windows 로그인만 있는 경우 그룹 AD 인증 세션에서 지원되지 않습니다. 이 제한을 해결하려면 암호로 인증된 세션에서 이러한 작업을 수행하거나 개별 Windows 로그인을 생성하는 것이 좋습니다.
- 암시적 사용자 생성은 지원되지 않습니다. 이상적인 T-SQL 동작 [Babelfish에서는 아직 지원되지 않음]: DDL 및 GRANT/REVOKE와 같은 액세스 제어문의 경우 AD 사용자의 이름이 명령에 지정되어 있지만, 데이터베이스에 존재하지 않는다면 AD 사용자로 명명된 데이터베이스 사용자가 암시적으로 생성됩니다.
- PSQL 엔드포인트에서 생성되고 그룹 AD 인증 세션의 TDS 엔드포인트에서 실행되는 PL/pgSQL 프로시저 또는 함수의 DDL인 경우:
  - ALTER/DROP 문이 지원됩니다.
  - 스키마가 명시적으로 제공되지 않고 기본 스키마가 현재 세션에 대해 null인 경우 CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE FUNCTION/PROC, CREATE TYPE, CREATE SEQUENCE, CREATE TRIGGER, SELECT INTO, CREATE FULLTEXT INDEX, CREATE UNIQUE INDEX에서 오류가 발생합니다.
  - CREATE DATABASE , CREATE EXTENSION 및 PG(T-SQL에 없음) 특정 객체용 기타 모든 CREATE 문(CREATE SUBSCRIPTION, CREATE TABLESPACE, CREATE POLICY, CREATE CONVERSION)은 지원되지 않습니다.
- PostgreSQL 엔드포인트의 DDL은 그룹 AD 인증 세션에서 지원되지 않습니다. 해결 방법으로 언제든지 마스터 사용자 또는 암호 기반 인증 메커니즘을 사용하는 다른 사용자로 연결할 수 있습니다.
- SUSER_SID(), IS_SRVROLEMEMBER(), IS_MEMBER(), sys.dm_exec_sessions와 같은 시스템 객체에는 다음과 같은 제한이 있습니다.
  - AD 사용자 또는 AD 보안 그룹이 제공된 경우 SUSER_SID()는 SID를 반환하지 않습니다.
  - 현재 AD 사용자가 Windows 그룹 로그인의 서버 역할 구성원 자격으로부터 서버 역할 구성원 자격을 상속받는 경우 IS_SRVROLEMEMBER()는 역할 구성원 자격을 고려하지 않습니다.
  - IS_MEMBER()는 모든 Windows 그룹 관련 쿼리에 대해 false를 반환합니다.
  - sys.dm_exec_sessions에는 예상 값 login_name, nt_user_name 열이 표시되지 않습니다.

PostgreSQL 포트의 PostgreSQL 엔드포인트를 통해 Babelfish에 연결

TDS 포트에서 생성된 그룹 로그인을 활용하여 PostgreSQL 포트를 통해 연결할 수도 있습니다. PostgreSQL 포트를 통해 연결하려면 PostgreSQL 클라이언트 애플리케이션의

<ad_username@FQDN> 형식으로 AD 사용자 이름을 지정해야 합니다. <DNS domain name \ad_username> 형식은 사용할 수 없습니다.

PostgreSQL은 기본적으로 사용자 이름에 대소문자를 구분하는 비교를 사용합니다. Aurora PostgreSQL이 Kerberos 사용자 이름을 대소문자를 구분하지 않는 것으로 해석하려면 사용자 지정 Babelfish 클러스터 파라미터 그룹에서 krb_caseins_users 파라미터를 true로 설정해야 합니다. 이 파라미터는 기본적으로 false로 설정되어 있습니다. 자세한 내용은 [대소문자를 구분하지 않는 사용자 이름을 위해 Aurora PostgreSQL DB 클러스터 구성](#) 단원을 참조하십시오.

AD 사용자가 여러 그룹에 속해 있는 경우 T-SQL과 PostgreSQL 엔드포인트 간의 동작 차이

AD 사용자 user1이 2개의 AD 보안 그룹 [corp\accounts-group] 및 [corp\sales-group]의 일부이고 DB 관리자가 다음과 같은 방식으로 사용자 매핑을 설정했다고 가정해 보겠습니다.

```
postgres=> select * from pgadmap_read_mapping();

ad_sid        | pg_role                          | weight | ad_grp
-------------+----------------------------------+--------+---------------
S-1-5-67-980 | accounts-group@CORP.EXAMPLE.COM | 7      | accounts-group
S-1-2-34-560 | sales-group@CORP.EXAMPLE.COM    | 10     | sales-group
(2 rows)
```

사용자가 T-SQL 엔드포인트에서 연결하는 경우 권한 부여 중에 연결된 모든 T-SQL 로그인의 권한이 상속됩니다. 이 예제에서 user1은 T-SQL 그룹 로그인 모두의 권한 통합을 상속하며 가중치는 무시됩니다. 이는 표준 T-SQL 동작과 일치합니다.

그러나 동일한 사용자가 PostgreSQL 엔드포인트에서 연결하는 경우 가중치가 가장 높은 하나의 관련 T-SQL 로그인에서만 권한을 상속할 수 있습니다. 두 T-SQL 그룹 로그인에 동일한 가중치가 할당되면 AD 사용자는 가장 최근에 추가된 매핑에 해당하는 T-SQL 로그인의 권한을 상속받습니다. PostgreSQL의 경우 모호성을 피하려면 개별 DB 역할의 상대적 허가/권한을 반영하는 가중치를 지정하는 것이 좋습니다. 아래 예제에서 user1은 PSQL 엔드포인트를 통해 연결했고 sales-groups 권한만 상속했습니다.

```
babelfish_db=> select session_user, current_user;

   session_user                  |   current_user
------------------------------+---------------------------
```

```
 sales-group@CORP.EXAMPLE.COM | sales-group@CORP.EXAMPLE.COM
(1 row)


babelfish_db=> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();

     principal          | gss_authenticated
------------------------+------------------
 user1@CORP.EXAMPLE.COM | t
(1 row)
```

Babelfish DB 클러스터에 연결

Babelfish에 연결하려면 Babelfish를 실행하는 Aurora PostgreSQL 클러스터의 엔드포인트에 연결합니다. 클라이언트는 TDS 버전 7.1부터 7.4까지와 호환되는 다음 클라이언트 드라이버 중 하나를 사용할 수 있습니다.

- ODBC(Open Database Connectivity)
- OLE DB 드라이버/MSOLEDBSQL
- 자바 데이터베이스 커넥티비티 (JDBC) 버전 8.2.2 (mssql-jdbc-8.2.2) 이상
- Microsoft SqlClient Data Provider for SQL Server
- .NET Data Provider for SQL Server
- SQL Server 기본 클라이언트 11.0(더 이상 사용되지 않음)
- OLE DB Provider/SQLOLEDB(더 이상 사용되지 않음)

Babelfish를 사용하여 다음을 실행합니다.

- TDS 포트(기본적으로 포트 1433)의 SQL Server 도구, 애플리케이션 및 구문입니다.
- PostgreSQL 포트(기본적으로 포트 5432)의 PostgreSQL 도구, 애플리케이션 및 구문입니다.

Aurora PostgreSQL에 연결하는 일반적인 방법에 대한 자세한 내용은 Amazon Aurora PostgreSQL DB 클러스터에 연결 섹션을 참조하세요.

> **Note**
>
> SQL Server OLEDB 제공업체를 사용하여 메타데이터에 액세스하는 타사 개발자 도구는 지원되지 않습니다. 이러한 도구에는 SQL Server JDBC, ODBC 또는 SQL Native 클라이언트 연결을 사용하는 것이 좋습니다.

Babelfish 버전 5.1.0부터 엔드투엔드 연결 암호화가 기본적으로 적용됩니다. 지속적인 연결을 보장하는 방법:

- 연결에 대한 SSL/TLS 암호화를 구성합니다. 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.
- 필요한 인증서를 클라이언트 컴퓨터에 가져옵니다. 자세한 내용은 Microsoft SQL Server DB 인스턴스와 함께 SSL 사용을 참조하세요.

이전 버전의 Babelfish(버전 5.1.0 전)에서 암호화 설정을 계속 사용하려면 DB 클러스터 파라미터 그룹에서 `rds.force_ssl` 파라미터를 `0`으로 설정하면 됩니다.

주제

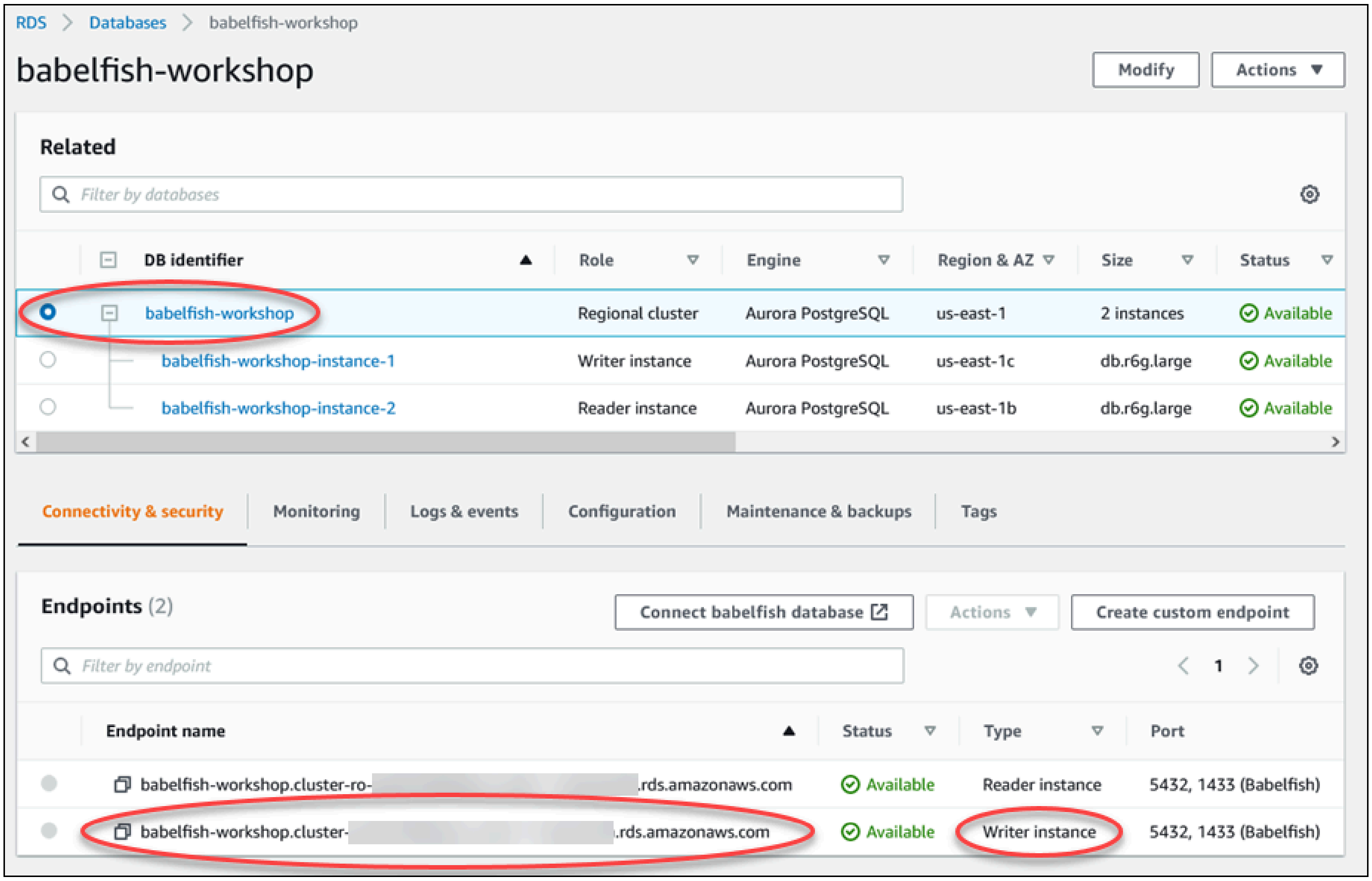
- 라이터 엔드포인트 및 포트 번호 찾기
- Babelfish에 대한 C# 또는 JDBC 클라이언트 연결 생성
- SQL Server 클라이언트 도구를 사용하여 DB 클러스터에 연결
- SQL Server 클라이언트를 사용하여 DB 클러스터에 연결

라이터 엔드포인트 및 포트 번호 찾기

Babelfish DB 클러스터에 연결하려면 DB 클러스터의 라이터(프라이머리) 인스턴스와 연결된 엔드포인트를 사용합니다. 인스턴스의 상태는 사용 가능(Available)이어야 합니다. Babelfish for Aurora PostgreSQL DB 클러스터를 생성한 후 인스턴스를 사용할 수 있기까지 최대 20분 정도 걸릴 수 있습니다.

데이터베이스 엔드포인트를 찾으려면

1. Babelfish의 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스(Databases)를 선택합니다.
3. 자세한 내용을 보려면 나열된 Babelfish for Aurora PostgreSQL DB 클러스터를 선택합니다.
4. 연결성 및 보안(Connectivity & security) 탭에서 사용 가능한 클러스터 엔드포인트(Endpoints) 값을 참고합니다. 데이터베이스 쓰기 또는 읽기 작업을 수행하는 모든 애플리케이션에 대해 연결 문자열에서 라이터 인스턴스의 클러스터 엔드포인트를 사용합니다.



Aurora DB 클러스터 세부 정보에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 섹션을 참조하세요.

Babelfish에 대한 C# 또는 JDBC 클라이언트 연결 생성

C# 및 JDBC 클래스를 사용하여 Babelfish for Aurora PostgreSQL에 연결하는 몇 가지 예는 다음과 같습니다.

Example : C# 코드를 사용하여 DB 클러스터 연결

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionString = connectionString + ";ServerCertificate=/path/to/certificate.pem";


SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example : 일반 JDBC API 클래스 및 인터페이스를 사용하여 DB 클러스터에 연결

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionUrl = connectionUrl + ";serverCertificate=/path/to/certificate.pem";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example : SQL Server별 JDBC 클래스 및 인터페이스를 사용하여 DB 클러스터에 연결

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
```

```
    "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

// [optional] To validate server certificate during TLS/SSL connection
ds.setServerCertificate("/path/to/certificate.pem");

Connection con = ds.getConnection();
```

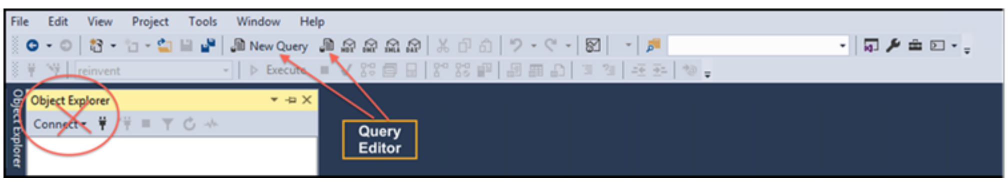
> **Important**
>
> 인증서가 AWS Management Console의 DB 클러스터 구성에 표시된 인증 기관과 일치하는지 확인합니다.

SQL Server 클라이언트 도구를 사용하여 DB 클러스터에 연결

SQL Server 클라이언트를 사용하여 TDS 포트의 Babelfish에 연결할 수 있습니다. Babelfish 2.1.0 이상 릴리스부터 SSMS 객체 탐색기 또는 SSMS 쿼리 편집기를 사용하여 Babelfish 클러스터에 연결할 수 있습니다.

제한 사항

- Babelfish 2.1.0 및 이전 버전에서는 `PARSE`를 사용하여 SQL 구문을 확인하는 작업이 제대로 작동하지 않습니다. `PARSE` 명령은 쿼리를 실행하지 않고 구문을 확인하는 대신 쿼리를 실행하지만, 결과를 표시하지 않습니다. SMSS <Ctrl><F5> 키 조합을 사용하여 구문을 확인하면 동일한 비정상적인 동작이 발생합니다. 즉, Babelfish는 출력을 제공하지 않고 예기치 않게 쿼리를 실행합니다.
- Babelfish는 Multiple Active Result Sets(MARS)를 지원하지 않습니다. Babelfish에 연결하는 데 사용하는 모든 클라이언트 애플리케이션이 MARS를 사용하도록 설정되어 있지 않은지 확인합니다.
- Babelfish 1.3.0 및 이전 버전의 경우 SSMS에 대해 쿼리 편집기만 지원됩니다. Babelfish에서 SSMS를 사용하려면 객체 탐색기가 아닌 SSMS에서 쿼리 편집기 연결 대화상자를 열어야 합니다. 객체 탐색기의 대화 상자가 열리면 대화 상자를 취소하고 쿼리 편집기를 다시 엽니다. 다음 이미지에서 Babelfish 1.3.0 또는 이전 버전에 연결할 때 선택할 메뉴 옵션을 찾을 수 있습니다.



SQL Server와 Babelfish 간의 상호 운용성 및 동작 차이에 대한 자세한 내용은 [Babelfish for Aurora PostgreSQL과 SQL Server의 차이점] 섹션을 참조하세요.

sqlcmd를 사용하여 DB 클러스터에 연결

버전 19.1 이전의 SQL Server `sqlcmd` 명령줄 클라이언트를 사용해야만 Babelfish를 지원하는 Aurora PostgreSQL DB 클러스터에 연결하고 상호 작용할 수 있습니다. SSMS 버전 19.2는 Babelfish 클러스터에 연결할 수 없습니다. 다음 명령을 사용하여 연결합니다.

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

옵션은 다음과 같습니다.

- `-S`는 DB 클러스터의 엔드포인트 및 TDS 포트(선택 사항)입니다.

- -U는 사용자의 로그인 이름입니다.
- -P는 사용자와 연결된 암호입니다.
- -d는 Babelfish 데이터베이스의 이름입니다.

연결 후에는 SQL Server에서 사용하는 것과 동일한 명령을 많이 사용할 수 있습니다. 몇 가지 예는 Babelfish 시스템 카탈로그에서 정보 얻기 섹션을 참조하세요.

SSMS를 사용하여 DB 클러스터에 연결

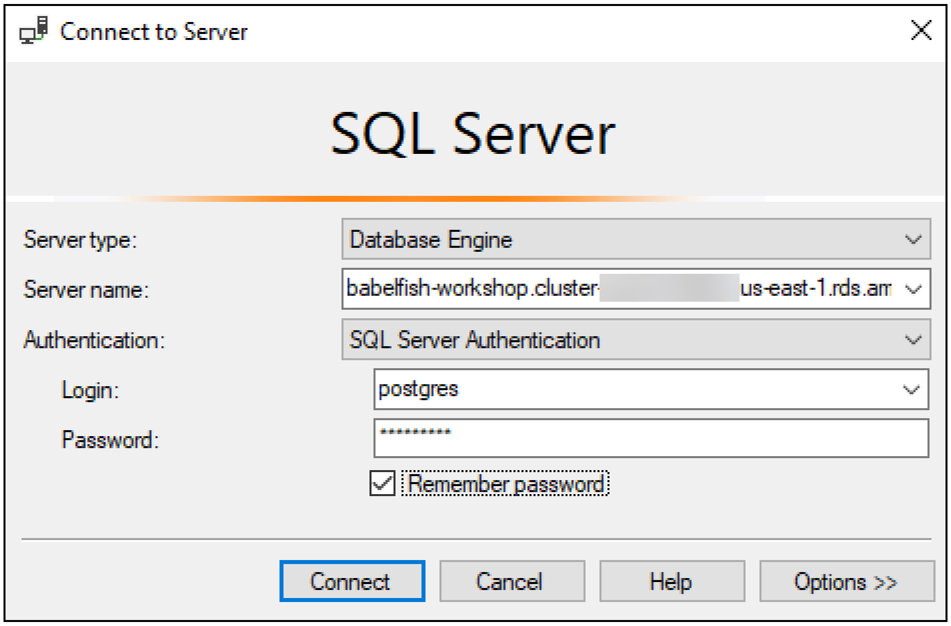
Microsoft SQL Server Management Studio(SSMS)를 사용하여 Babelfish를 실행하는 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다. SSMS에는 SQL Server 데이터베이스를 Babelfish for Aurora PostgreSQL로 마이그레이션에서 설명하는 SQL Server 가져오기 및 내보내기 마법사를 비롯한 여러 도구가 포함되어 있습니다. SSMS에 대한 자세한 내용은 Microsoft 설명서의 SQL Server Management Studio(SSMS) 다운로드를 참조하세요. SSL/TLS를 구성하려면 Microsoft SQL Server DB 인스턴스와 함께 SSL 사용을 참조하세요.

SSMS를 사용하여 Babelfish 데이터베이스에 연결하려면

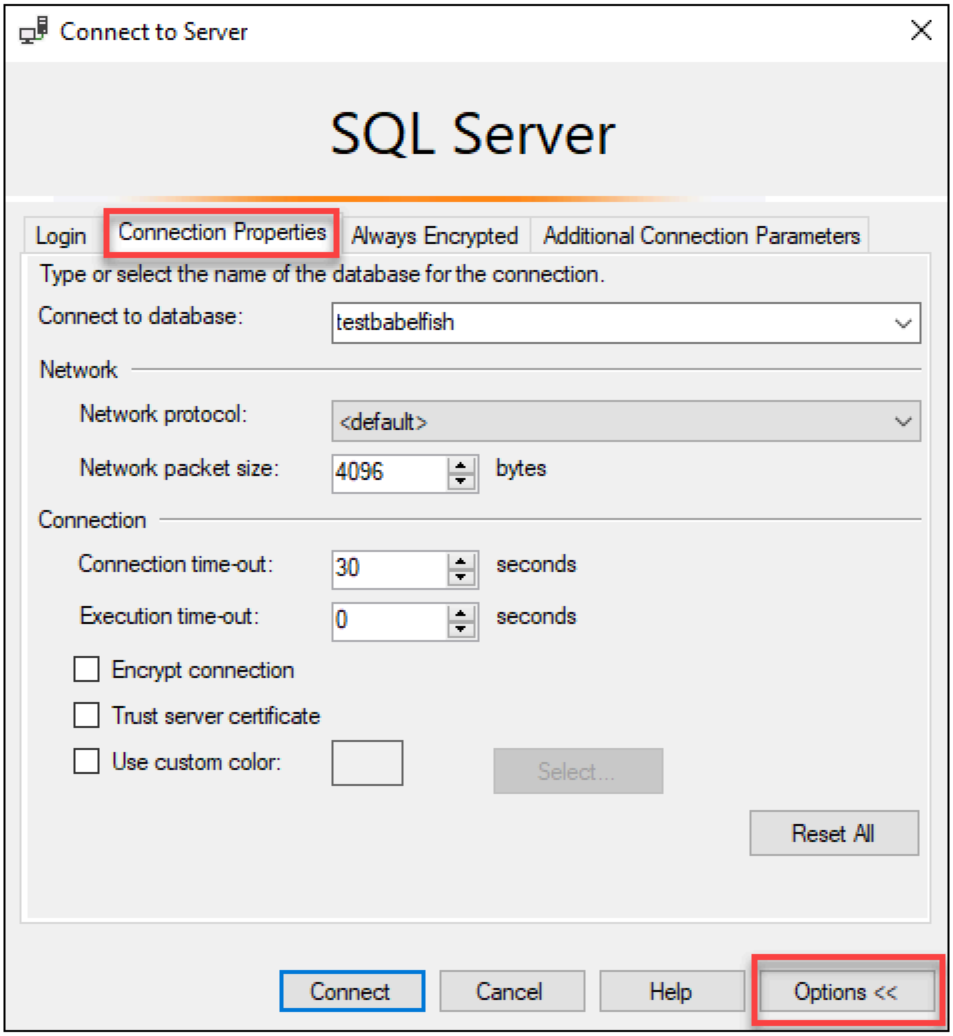
1. SSMS를 시작합니다.
2. Connect to Server 대화 상자가 열립니다. 연결을 계속하려면 다음 중 하나를 수행합니다.
   - 새 쿼리(New Query)를 선택합니다.
   - 쿼리 편집기가 열려 있는 경우 쿼리(Query), 연결(Connection), 연결(Connect)을 선택합니다.
3. 데이터베이스에 대한 다음 정보를 제공합니다.
   a. 서버 유형(Server type)에서 데이터베이스 엔진(Database Engine)을 선택합니다.
   b. 서버 이름(Server name)에 DNS 이름을 입력합니다. 예를 들어 서버 이름은 다음과 같은 형식이어야 합니다.

   ```
   cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
   ```

   c. 인증(Authentication)]의 경우 SQL Server 인증(SQL Server Authentication)을 선택합니다.
   d. 로그인(Login)에 데이터베이스를 생성할 때 선택한 사용자 이름을 입력합니다.
   e. 암호(Password)에 데이터베이스를 생성할 때 선택한 암호를 입력합니다.



4. (선택 사항) 옵션(Options)을 선택한 다음, 연결 속성(Connection Properties) 탭을 선택합니다.



5. (선택 사항) 데이터베이스에 연결(Connect to database)에서 연결할 마이그레이션된 SQL Server 데이터베이스의 이름을 지정하고 연결(Connect)을 선택합니다.

   SSMS가 연결 문자열을 적용할 수 없다는 메시지가 나타나면 확인(OK)을 선택합니다.

   Babelfish 연결에 문제가 있으면 연결 실패 단원을 참조하세요.

   SQL Server 연결 문제에 대한 자세한 내용은 Amazon RDS 사용 설명서의 SQL Server 인스턴스에 대한 연결 문제 해결을 참조하세요.

SQL Server 클라이언트를 사용하여 DB 클러스터에 연결

PostgreSQL 클라이언트를 사용하여 PostgreSQL 포트의 Babelfish에 연결할 수 있습니다. 버전 5.1.0부터 Babelfish 서버는 기본적으로 엔드투엔드 연결 암호화를 적용합니다. SSL/TLS 인증서를 사용하도록 애플리케이션을 업데이트하세요. SSL/TLS 인증서 구성에 대한 자세한 내용은 the section called “SSL/TLS를 이용한 Aurora PostgreSQL 데이터 보안” 섹션을 참조하세요.

psql을 사용하여 DB 클러스터에 연결

PostgreSQL 웹 사이트에서 PostgreSQL 클라이언트를 다운로드할 수 있습니다. psql을 설치하려면 운영 체제 버전별 지침을 따르세요.

`psql` 명령줄 클라이언트를 사용하여 Babelfish를 지원하는 Aurora PostgreSQL DB 클러스터를 쿼리할 수 있습니다. 연결할 때 PostgreSQL 포트를 사용합니다(기본적으로 포트 5432). 일반적으로 기본값에서 변경하지 않는 한 포트 번호를 지정할 필요가 없습니다. 다음 명령을 사용하여 `psql` 클라이언트에서 Babelfish에 연결합니다.

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

파라미터는 다음과 같습니다.

- `-h` - 액세스할 DB 클러스터(클러스터 엔드포인트)의 호스트 이름입니다.
- `-p` - DB 인스턴스에 연결하는 데 사용할 PostgreSQL 포트 번호입니다.
- `-d` – 연결하려는 데이터베이스입니다. 기본값은 `babelfish_db`입니다.
- `-U` - 액세스할 데이터베이스 사용자 계정입니다. (이 예에서는 기본 마스터 사용자 이름을 보여줍니다.)

psql 클라이언트에서 SQL 명령을 실행하면 세미콜론으로 명령을 종료합니다. 예를 들어 다음 SQL 명령은 pg_tables 시스템 뷰를 쿼리하여 데이터베이스의 각 테이블에 대한 정보를 반환합니다.

```
SELECT * FROM pg_tables;
```

또한 psql 클라이언트에는 기본 제공 메타 명령 집합이 있습니다. 메타 명령은 서식을 조정하거나 메타데이터를 사용하기 쉬운 형식으로 반환하는 바로 가기를 제공하는 바로 가기입니다. 예를 들어, 다음 메타 명령은 이전의 SQL 명령과 유사한 정보를 반환합니다.

```
\d
```

메타 명령은 세미콜론(;)으로 종료할 필요가 없습니다.

psql 클라이언트를 종료하려면 \q를 입력합니다.

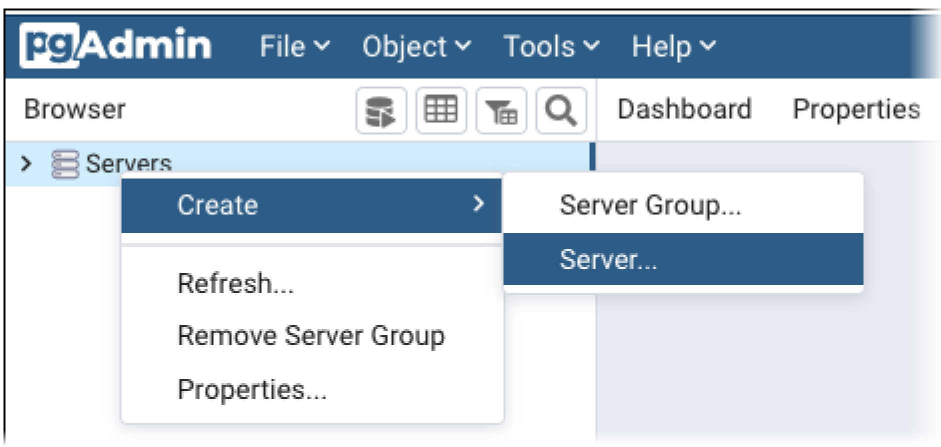
psql 클라이언트를 사용하여 Aurora PostgreSQL 클러스터를 쿼리하는 방법에 대한 자세한 내용은 PostgreSQL 설명서를 참조하세요.

pgAdmin을 사용하여 DB 클러스터에 연결

pgAdmin 클라이언트를 사용하여 기본 PostgreSQL 언어로 데이터에 액세스할 수 있습니다.

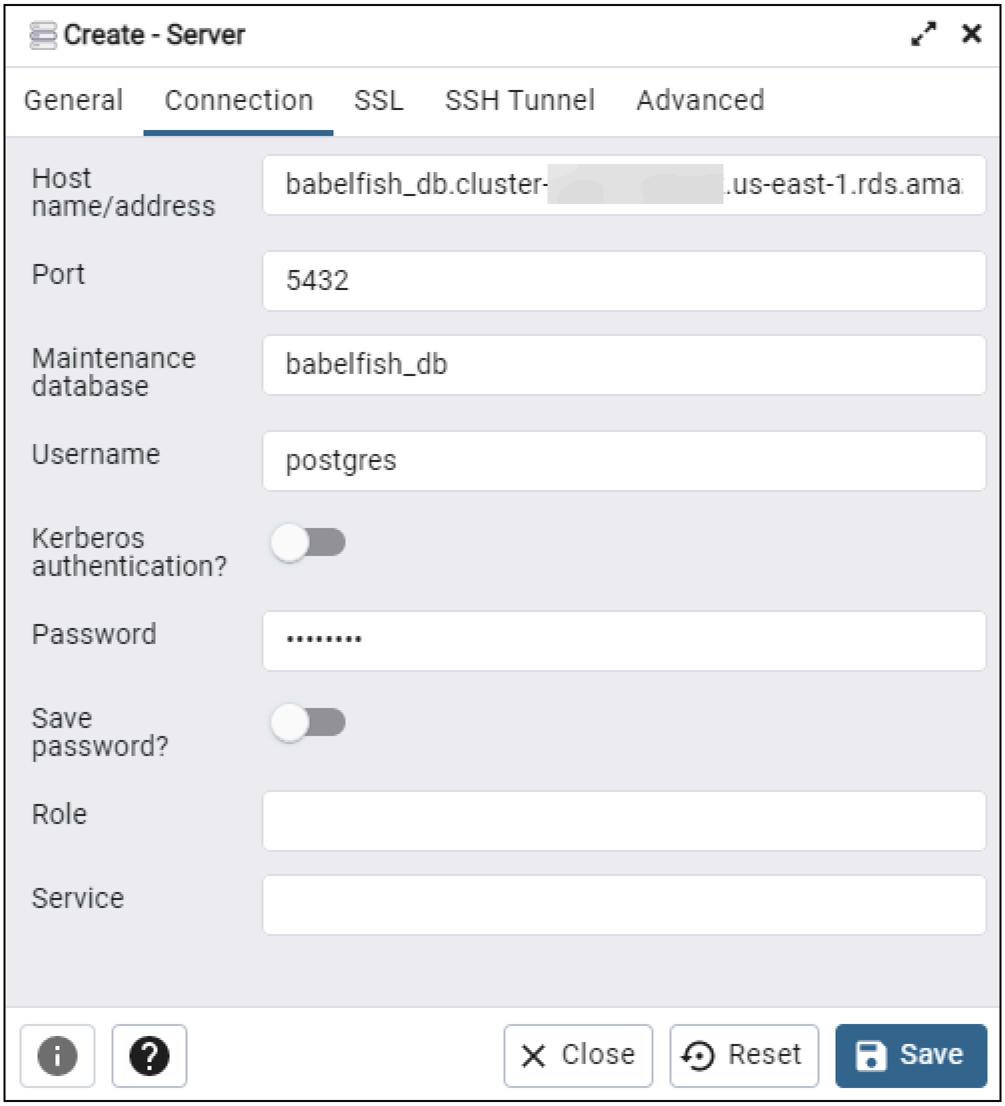
pgAdmin 클라이언트를 사용하여 클러스터에 연결하려면

1. pgAdmin 웹 사이트에서 pgAdmin 클라이언트를 다운로드하여 설치합니다.
2. 클라이언트를 열고 pgAdmin을 사용하여 인증합니다.
3. 서버(Servers)에 대한 컨텍스트 메뉴를 연 다음(마우스 오른쪽 버튼 클릭), 생성(Create), 서버(Servers)를 선택합니다.

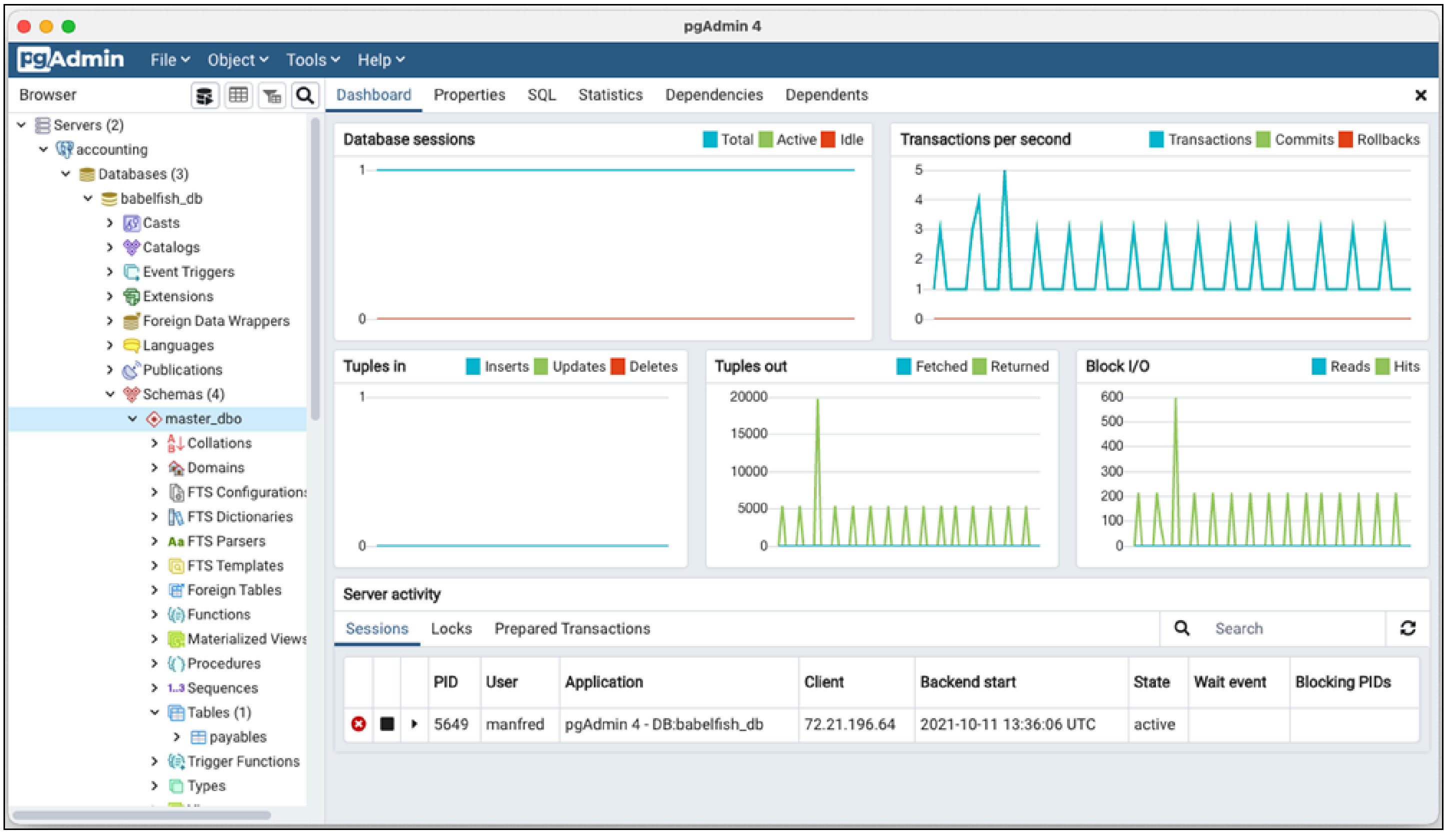


4. 생성 - 서버(Create - Server) 대화 상자에 정보를 입력합니다.

   연결(Connection) 탭에서 호스트(Host)에 대한 Aurora PostgreSQL 클러스터 주소 및 포트(Port)에 대한 PostgreSQL 포트 번호(기본값: 5432)를 추가합니다. 인증 세부 정보를 제공하고 저장(Save)을 선택합니다.



연결 후 pgAdmin 기능을 사용하여 PostgreSQL 포트에서 Aurora PostgreSQL 클러스터를 모니터링하고 관리할 수 있습니다.



자세한 내용은 pgAdmin 웹 페이지를 참조하세요.

Babelfish 작업

Babelfish와 SQL Server 작업 간의 차이점과 Babelfish와 PostgreSQL 데이터베이스 간의 차이점 등 Babelfish에 대한 사용 정보가 아래에 나와 있습니다.

주제

Babelfish 시스템 카탈로그에서 정보 얻기

SQL Server에서 사용되는 것과 동일한 여러 시스템 뷰를 쿼리하여 Babelfish 클러스터에 저장된 데이터베이스 객체에 대한 정보를 얻을 수 있습니다. Babelfish의 새로운 릴리스마다 더 많은 시스템 뷰를 지원합니다. 현재 사용할 수 있는 뷰 목록은 SQL Server system catalog views 테이블을 참조하세요.

이러한 시스템 뷰는 시스템 카탈로그(`sys.schemas`)의 정보를 제공합니다. Babelfish의 경우 이러한 뷰에는 SQL Server 및 PostgreSQL 시스템 스키마가 모두 포함되어 있습니다. Babelfish에서 시스템 카탈로그 정보를 쿼리하려면 다음 예와 같이 TDS 포트 또는 PostgreSQL 포트를 사용할 수 있습니다.

- **`sqlcmd`**를 사용하여 T-SQL 포트를 쿼리하거나 다른 SQL Server 클라이언트를 쿼리합니다.

  ```
  1> SELECT * FROM sys.schemas
  2> GO
  ```

  이 쿼리는 다음과 같이 SQL Server 및 Aurora PostgreSQL 시스템 스키마를 반환합니다.

  ```
  name
  -----------------------------------------------------------
  ```

```
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- **psql** 또는 **pgAdmin**을 사용하여 PostgreSQL 포트를 쿼리합니다. 이 예에서는 psql 목록 스키마 메타 명령(\dn)을 사용합니다.

```
babelfish_db=> \dn
```

쿼리는 sqlcmd에서 T-SQL 포트에 반환한 것과 동일한 결과 세트를 반환합니다.

```
        List of schemas
             Name
------------------------------

 demographic_dbo

 public
 sys
 master_dbo
 tempdb_dbo
...
```

Babelfish에서 사용할 수 있는 SQL Server 시스템 카탈로그

다음 테이블에서는 Babelfish에 현재 구현된 SQL Server 뷰를 확인할 수 있습니다. SQL Server의 시스템 카탈로그에 대한 자세한 내용은 Microsoft 설명서의 [시스템 카탈로그 뷰(Transact-SQL)](https://)를 참조하세요.

뷰 이름	설명 또는 Babelfish 제한(있는 경우)
`sys.all_columns`	모든 테이블 및 뷰의 모든 열
`sys.all_objects`	모든 스키마의 모든 객체
`sys.all_sql_modules`	`sys.sql_modules` 및 `sys.system_sql_modules` 연합

뷰 이름	설명 또는 Babelfish 제한(있는 경우)
sys.all_views	모든 스키마의 모든 뷰
sys.columns	사용자 정의 테이블 및 뷰의 모든 열
sys.configurations	Babelfish 지원은 단일 읽기 전용 구성으로 제한됩니다.
sys.data_spaces	각 데이터 공간에 대한 행이 포함됩니다. 파일 그룹, 파티션 구성표 또는 FILESTREAM 데이터 파일 그룹이 될 수 있습니다.
sys.database_files	데이터베이스 자체에 저장된 데이터베이스의 각 파일에 대해 하나의 행이 포함된 데이터베이스별 뷰입니다.
sys.database_mirroring	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.database_mirroring을 참조하세요.
sys.database_principals	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.database_principals를 참조하세요.
sys.database_role_members	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.database_role_members를 참조하세요.
sys.databases	모든 스키마의 모든 데이터베이스
sys.dm_exec_connections	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.dm_exec_connections를 참조하세요.
sys.dm_exec_sessions	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.dm_exec_sessions를 참조하세요.
sys.dm_hadr_database_replica_states	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.dm_hadr_database_replica_states를 참조하세요.
sys.dm_os_host_info	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.dm_os_host_info를 참조하세요.

뷰 이름	설명 또는 Babelfish 제한(있는 경우)
sys.endpoints	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.endpoints를 참조하세요.
sys.indexes	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.indexes를 참조하세요.
sys.languages	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.languages를 참조하세요.
sys.schemas	모든 스키마
sys.server_principals	모든 로그인 및 역할
sys.sql_modules	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.sql_modules를 참조하세요.
sys.sysconfigures	Babelfish 지원은 단일 읽기 전용 구성으로 제한됩니다.
sys.syscurconfigs	Babelfish 지원은 단일 읽기 전용 구성으로 제한됩니다.
sys.sysprocesses	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.sysprocesses를 참조하세요.
sys.system_sql_modules	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.system_sql_modules를 참조하세요.
sys.table_types	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.table_types를 참조하세요.
sys.tables	스키마의 모든 테이블
sys.xml_schema_collections	자세한 내용은 Microsoft Transact-SQL 설명서의 sys.xml_schema_collections를 참조하세요.

PostgreSQL은 SQL Server 객체 카탈로그 뷰와 유사한 시스템 카탈로그를 구현합니다. 전체 시스템 카탈로그 목록은 PostgreSQL 설명서의 [시스템 카탈로그]를 참조하세요.

Babelfish에서 지원되는 DDL 내보내기

Babelfish 2.4.0 및 3.1.0 버전의 경우 Babelfish는 다양한 도구에서 DDL 내보내기를 지원합니다. 예를 들어, SQL Server Management Studio(SSMS)에서 이 기능을 사용하여 Babelfish for Aurora PostgreSQL 데이터베이스의 다양한 객체에 대한 데이터 정의 스크립트를 생성할 수 있습니다. 그런 다음, 이 스크립트에서 생성된 DDL 명령을 사용하여 다른 Babelfish for Aurora PostgreSQL 또는 SQL Server 데이터베이스에서 동일한 객체를 생성할 수 있습니다.

Babelfish는 지정된 버전에서 다음과 같은 객체에 대한 DDL 내보내기를 지원합니다.

객체 목록	2.4.0	3.1.0
사용자 테이블	예	예
기본 키	예	예
외래 키	예	예
고유한 제약 조건	예	예
인덱스	예	예
제약 조건 확인	예	예
보기	예	예
저장 프로시저	예	예
사용자 정의 함수	예	예
테이블 값 함수	예	예
트리거	예	예
사용자 정의 데이터 형식	아니요	아니요
사용자 정의 테이블 유형	아니요	아니요
Users	아니요	아니요

객체 목록	2.4.0	3.1.0
로그인	아니요	아니요
시퀀스	아니요	아니요
Roles	아니요	아니요

내보낸 DDL의 제한 사항

- 내보낸 DDL을 사용하여 객체를 다시 생성하기 전에 이스케이프 해치 사용 - Babelfish는 내보낸 DDL 스크립트의 모든 명령을 지원하지는 않습니다. Babelfish의 DDL 명령에서 객체를 다시 생성할 때 발생하는 오류를 방지하려면 이스케이프 해치를 사용하세요. 이스케이프 해치에 대한 자세한 내용은 이스케이프 해치를 사용하여 Babelfish 오류 처리 관리 단원을 참조하세요.
- 명시적 COLLATE 절과 함께 CHECK 제약 조건을 포함하는 객체 - SQL Server 데이터베이스에서 생성된 이러한 객체가 포함된 스크립트는 Babelfish 데이터베이스와 같이 서로 형태는 다르지만 동일한 데이터 정렬을 사용합니다. 예를 들어 sql_latin1_general_cp1_cs_as, sql_latin1_general_cp1251_cs_as 및 latin1_general_cs_as 같은 몇 가지 데이터 정렬은 가장 가까운 Windows 데이터 정렬인 latin1_general_cs_as로 생성됩니다.

Babelfish for Aurora PostgreSQL에서 권한 및 액세스 제어 관리

Babelfish for Aurora PostgreSQL에서 데이터베이스, 스키마 및 객체에 대한 권한 및 액세스 제어를 관리할 수 있습니다. 다음 표에는 Babelfish에서 다양한 액세스 제어 시나리오를 달성할 수 있는 권한을 부여하는 특정 SQL 명령이 요약되어 있습니다. 현재 지원되지 않는 사례에 대한 해결 방법뿐만 아니라 구현할 수 있는 지원되는 사용 사례를 다룹니다. 이렇게 하면 Babelfish 데이터베이스로 작업할 때 보안 및 규정 준수 요구 사항에 맞게 적절한 권한을 구성할 수 있습니다.

지원되는 사용 사례

다음 표에서는 Babelfish에서 지원되는 사용 사례를 설명합니다. 각 사용 사례에 대해 표에는 이를 달성하는 데 필요한 작업과 샘플 SQL 명령이 나와 있습니다.

사용 사례	작업	SQL 명령	설명	Babelfish 버전 호환성
모든 데이터베이스에서 SELECTs/DMLs/DDLs를 수행하기 위한 로그인 허용	sysadmin 서버 역할에 로그인 추가	ALTER SERVER ROLE sysadmin ADD MEMBER `login`	없음	모든 버전
데이터베이스에서 SELECTs/DMLs/DDLs를 수행하기 위한 로그인 허용	로그인을 데이터베이스의 소유자로 설정	ALTER AUTHORIZATION ON DATABASE::`database` TO `login`	데이터베이스에는 소유자가 하나만 있을 수 있습니다.	버전 3.4 이상
데이터베이스 사용자가 스키마에서 SELECTs/DMLs를 수행하도록 허용	스키마에서 데이터베이스 사용자에게 권한 부여	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON SCHEMA::`schema` TO `user`	없음	버전 3.6 이상, 4.2 이상
데이터베이스 사용자가 스키마에서 SELECTs/DMLs를 수행하도록 허용	스키마 생성 시 스키마의 데이터베이스 사용자 소유자 생성	CREATE SCHEMA `schema` AUTHORIZATION `user`	만든 후 스키마 소유권 변경은 현재 지원되지 않습니다.	버전 1.2 이상
데이터베이스 사용자가 객체에서 SELECTs/DMLs를 수행하도록 허용	객체의 데이터베이스 사용자에게 권한 부여	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON	없음	모든 버전

사용 사례	작업	SQL 명령	설명	Babelfish 버전 호환성
		OBJECT::`object` TO `user`		
데이터베이스 사용자가 데이터베이스 삭제를 포함하여 데이터베이스에서 SELECTs/DMLs/DDLs를 수행하도록 허용	db_owner 고정 데이터베이스 역할에 사용자 추가	ALTER ROLE db_owner ADD MEMBER `user`	db_owner 고정 데이터베이스 역할에 사용자만 추가할 수 있습니다. db_owner 역할에 역할을 추가하는 것은 아직 지원되지 않습니다.	버전 4.5 이상, 5.1 이상
사용자 지정 데이터베이스 역할의 사용자 또는 구성원이 데이터베이스에서 SELECT만 수행하도록 허용	db_datareader 고정 데이터베이스 역할에 사용자 또는 역할 추가	ALTER ROLE db_datareader ADD MEMBER `user` / `role`	없음	버전 4.5 이상, 5.1 이상
사용자 지정 데이터베이스 역할의 사용자 또는 구성원이 데이터베이스에서 DML만 수행하도록 허용	db_datawriter 고정 데이터베이스 역할에 사용자 또는 역할 추가	ALTER ROLE db_datawriter ADD MEMBER `user` / `role`	없음	버전 4.5 이상, 5.1 이상
사용자 지정 데이터베이스 역할의 사용자 또는 구성원이 데이터베이스에서 DDL만 수행하도록 허용	db_accessadmin 고정 데이터베이스 역할에 사용자 또는 역할 추가	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	없음	버전 4.5 이상, 5.1 이상

사용 사례	작업	SQL 명령	설명	Babelfish 버전 호환성
사용자 지정 데이터베이스 역할의 사용자 또는 구성원에게 사용자 지정 역할 CREATE/ALTER/DROP, 데이터베이스의 객체에 대한 권한 GRANT/REVOKE 및/또는 데이터베이스의 CREATE SCHEMA만 허용	db_securityadmin 고정 데이터베이스 역할에 사용자 또는 역할 추가	ALTER ROLE db_securityadmin ADD MEMBER `user` / `role`	없음	버전 4.5 이상, 5.1 이상
사용자 지정 데이터베이스 역할의 사용자 또는 구성원에게 모든 사용자에 대한 CREATE/ALTER/DROP, 데이터베이스 액세스 권한 부여 또는 취소, 로그인에 사용자 계정 매핑 및/또는 데이터베이스에서의 CREATE SCHEMA만 허용	db_accessadmin 고정 데이터베이스 역할에 사용자 또는 역할 추가	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	없음	버전 4.5 이상, 5.1 이상

사용 사례	작업	SQL 명령	설명	Babelfish 버전 호환성
로그인에 모든 데이터베이스에 대한 CREATE/DROP/ALTER만 허용	dbcreator 고정 서버 역할에 로그인 추가	ALTER SERVER ROLE dbcreator ADD MEMBER `login`	dbcreator 로그인이 액세스할 수 있는 데이터베이스만 변경할 수 있습니다.	버전 4.5 이상, 5.1 이상
로그인에 모든 로그인에 대한 CREATE/ALTER/DROP만 허용	securityadmin 고정 서버 역할에 로그인 추가	ALTER SERVER ROLE securityadmin ADD MEMBER `login`	없음	버전 4.5 이상, 5.1 이상

지원되지 않는 사용 사례 및 해결 방법

다음 표에서는 Babelfish에서 지원되지 않지만 해결 방법을 사용하여 달성할 수 있는 사용 사례를 설명합니다.

사용 사례	작업	SQL 명령	설명	해결 방법을 위한 Babelfish 버전 호환성
다른 사용자에게 이러한 권한을 GRANT하는 옵션과 함께 사용자가 객체/스키마에 SELECT/DML을 수행하도록 허용	다른 모든 사용자에게 직접 권한 GRANT	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT/SCHEMA::`object/sc` TO `user`	GRANT ... WITH GRANT OPTION은 현재 지원되지 않습니다.	버전 3.6 이상, 4.2 이상
데이터베이스 사용자가 데이터베이스 삭제를 포함하여 데	db_owner 고정 데이터베이스 역할에 역할 구성원 추가	ALTER ROLE db_owner ADD MEMBER `user`	db_owner 역할에 역할 추가는 현재 지원되지 않습니다.	버전 4.5 이상, 5.1 이상

사용 사례	작업	SQL 명령	설명	해결 방법을 위한 Babelfish 버전 호환성
이터베이스에서 SELECTs/DMLs/DDLs를 수행하도록 허용				

지원되는 사용 사례

다음 표에서는 Babelfish에서 지원되지 않는 사용 사례를 설명합니다.

사용 사례	설명
사용자 지정 데이터베이스 역할의 사용자 또는 구성원이 데이터베이스에서 SELECT를 수행하지 못하도록 거부	db_denydatareader 고정 데이터베이스 역할은 아직 지원되지 않습니다.
사용자 지정 데이터베이스 역할의 사용자 또는 구성원이 데이터베이스에서 DML을 수행하지 못하도록 거부	db_denydatawriter 고정 데이터베이스 역할은 현재 지원되지 않습니다.
로그인에 모든 데이터베이스 연결에 대한 KILL만 허용	processadmin 고정 서버 역할은 현재 지원되지 않습니다.
로그인에 연결된 서버 추가 또는 제거만 허용	setupadmin 고정 서버 역할은 현재 지원되지 않습니다.

업그레이드 후 객체 소유권 차이 해결

Babelfish 버전 4.6 이상 및 5.2 이상에는 TDS 엔드포인트를 통한 객체 소유권 처리에 대한 변경 사항이 포함되어 있습니다. TDS 엔드포인트를 통해 새 객체를 생성하면 기존 사용자 대신 스키마 소유자가 이러한 객체를 소유합니다. 이 소유권 변경은 4.6 또는 5.2 전 버전에서 업그레이드할 때 기존 객체와 비교하여 새 객체의 권한 동작에 영향을 미칠 수 있습니다.

이러한 소유권 차이를 해결하기 위해 Babelfish는 `sys.generate_alter_ownership_statements()` 함수를 제공합니다. 이 함수는 객체 소유권을 스키마 소유권과 일치시키는 SQL 문을 생성합니다.

객체 소유권을 처리할 때 다음 제한 사항에 유의하세요.

- PostgreSQL 엔드포인트를 통해 CREATE 권한을 부여받은 사용자는 해당 스키마의 TDS 엔드포인트를 통해 객체를 생성할 수 없습니다.
- PostgreSQL 엔드포인트를 통해 T-SQL 객체에 대한 권한을 수정하는 것은 권장되지 않으며 잘못된 T-SQL 동작을 유발할 수 있습니다.
- 액세스 권한은 소유권 불일치로 인해 이전 객체와 새 객체 간에 다를 수 있습니다. 예를 들어 `sch_own`에서 소유한 스키마에 dbo에서 소유한 객체가 포함되어 있다고 생각해 봅시다. 이 경우 업그레이드 전에 생성된 dbo 소유 객체는 업그레이드 후 생성된 `sch_own` 소유 객체와 다른 액세스 권한을 가질 수 있습니다. 이는 SELECT 및 INSERT와 같은 작업에 영향을 미칠 수 있습니다.

DB 클러스터에 Babelfish 버전 4.6 또는 5.2 전에 생성된 객체가 포함된 경우 소유권을 조정하는 것이 좋습니다.

객체 소유권 차이를 해결하는 방법

1. PostgreSQL 엔드포인트를 사용하여 DB 클러스터의 `babelfish_db` 데이터베이스에 연결합니다.
2. 다음 명령 실행:

   ```
   SELECT * from sys.generate_alter_ownership_statements();
   ```

   이렇게 하면 객체 간의 소유권을 표준화하기 위한 SQL 문 목록이 생성됩니다.
3. 먼저 테스트 환경에서 생성된 문을 실행하여 효과를 검증한 후 프로덕션 환경에 적용합니다.

데이터베이스 전체에서 일관된 객체 소유권 모델을 달성하려면 이러한 문을 실행하는 것이 좋습니다.

Babelfish for Aurora PostgreSQL과 SQL Server의 차이점

Babelfish는 끊임없이 발전하는 Aurora PostgreSQL 기능으로, Aurora PostgreSQL 13.4의 초기 제공 이후 각 릴리스에 추가된 새 기능입니다. TDS 포트를 사용하는 T-SQL 언어를 통해 PostgreSQL 위에 T-SQL 의미 체계를 제공하도록 설계되었습니다. 버전별 Babelfish에서 지원되는 기능 테이블에 표시된 것처럼 Babelfish 버전이 나올 때마다 T-SQL 기능 및 동작에 잘 맞는 기능이 추가되고 있습니

다. Babelfish로 작업할 때 최상의 결과를 얻으려면 SQL Server에서 지원하는 T-SQL과 최신 버전의 Babelfish 간에 현재 존재하는 차이점을 이해하는 것이 좋습니다. 자세한 내용은 Babelfish에서의 T-SQL 차이점를 참조하세요.

Babelfish와 SQL Server에서 지원하는 T-SQL의 차이점 외에도 Aurora PostgreSQL DB 클러스터 컨텍스트에서 Babelfish와 PostgreSQL 간의 상호 운용성 문제를 고려해야 할 수도 있습니다. 앞서 언급했듯이 Babelfish는 TDS 포트를 사용하여 T-SQL 언어를 통해 PostgreSQL 외에 T-SQL 의미 체계를 지원합니다. 동시에 PostgreSQL SQL 문을 사용하여 표준 PostgreSQL 포트를 통해 Babelfish 데이터베이스에 액세스할 수도 있습니다. 프로덕션 배포에서 PostgreSQL과 Babelfish 기능을 모두 사용하려는 경우 스키마 이름, 식별자, 권한, 트랜잭션 의미 체계, 다중 결과 집합, 기본 데이터 정렬 등 간의 잠재적인 상호 운용성 문제를 알고 있어야 합니다. 간단히 말해서 Babelfish 컨텍스트에서 PostgreSQL 문 또는 PostgreSQL 액세스가 발생하면 PostgreSQL과 Babelfish 사이에 간섭이 발생할 수 있으며 새 버전의 Babelfish가 출시될 때 잠재적으로 구문, 의미 및 호환성에 영향을 미칠 수 있습니다. 모든 고려 사항에 대한 전체 정보 및 지침은 PostgreSQL용 Babelfish 설명서에서 Babelfish 상호 운용성에 대한 지침을 참조하세요.

> **Note**
>
> 동일한 애플리케이션 컨텍스트에서 PostgreSQL 기본 기능과 Babelfish 기능을 모두 사용하기 전에 Babelfish for PostgreSQL 설명서의 Babelfish 상호 운용성에 대한 지침에서 논의된 문제를 고려하는 것이 좋습니다. 이러한 상호 운용성 문제(Aurora PostgreSQL 및 Babelfish)는 Babelfish와 동일한 애플리케이션 컨텍스트에서 PostgreSQL 데이터베이스 인스턴스를 사용하려는 경우에만 관련이 있습니다.

주제

Babelfish 덤프 및 복원

버전 4.0.0과 3.4.0부터 Babelfish 사용자는 이제 덤프 및 복원 유틸리티를 활용하여 데이터베이스를 백업하고 복원할 수 있습니다. 자세한 내용은 Babelfish 덤프 및 복원을 참조하세요. 이 기능은 PostgreSQL 덤프 및 복원 유틸리티를 기반으로 구축됩니다. 자세한 내용은 pg_dump 및 pg_restore를 참조하세요. Babelfish에서 이 기능을 효과적으로 사용하려면 Babelfish에 맞게 특별히 조정된 PostgreSQL 기반 도구를 사용해야 합니다. Babelfish의 백업 및 복원 기능은 SQL Server의 백업 및 복

원 기능과 크게 다릅니다. 이러한 차이점에 대한 자세한 내용은 덤프 및 복원 기능의 차이점: Babelfish 와 SQL Server를 참조하세요. Babelfish for Aurora PostgreSQL은 Amazon Aurora PostgreSQL DB 클러스터 백업 및 복원을 위한 추가 기능을 제공합니다. 자세한 내용은 Amazon Aurora DB 클러스터 백업 및 복구 섹션을 참조하세요.

Babelfish에서의 T-SQL 차이점

현재 Babelfish 릴리스에서 지원되는 T-SQL 기능 표와 SQL Server와의 동작 차이점에 대한 몇 가지 참고 사항을 아래에서 확인할 수 있습니다.

다양한 버전 지원에 대한 자세한 내용은 버전별 Babelfish에서 지원되는 기능 단원을 참조하세요. 현재 지원되지 않는 기능에 대한 자세한 내용은 Babelfish에서 지원되지 않는 기능 단원을 참조하세요.

Babelfish는 Aurora PostgreSQL-Compatible Edition과 함께 사용할 수 있습니다. Babelfish 릴리스에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

기능 또는 구문	동작 또는 차이점에 대한 설명
\(줄 연속 문자)	문자 및 16진수 문자열에 대한 줄 연속 문자(줄 바꿈 앞 백슬래시)는 현재 지원되지 않습니다. 문자열의 경우 백슬래시-줄 바꿈은 문자열의 문자로 해석됩니다. 16진수 문자열의 경우 백슬래시-줄 바꿈으로 인해 구문 오류가 발생합니다.
@@version	`@@version` 에서 반환하는 값의 형식은 SQL Server에서 반환하는 값과 약간 다릅니다. `@@version` 형식에 따라 코드가 올바르게 작동하지 않을 수 있습니다.
집계 함수	집계 함수는 부분적으로 지원됩니다(AVG, COUNT, COUNT_BIG , GROUPING, MAX, MIN, STRING_AGG 및 SUM이 지원됨). 지원되지 않는 집계 함수 목록은 지원되지 않는 함수 단원을 참조하세요.
ALTER TABLE	단일 열 또는 제약 조건의 추가 또는 삭제만 지원합니다.
ALTER TABLE..ALTER COLUMN	NULL 및 NOT NULL은 현재 지정할 수 없습니다. 열의 null 허용 여부를 변경하려면 postgreSQL 문 ALTER TABLE..{SET|DROP} NOT NULL을 사용합니다.

기능 또는 구문	동작 또는 차이점에 대한 설명

AT TIME ZONE	일광 절약 시간(DST)에서 표준 시간으로 전환되는 동안 표준 시간 오프셋을 사용하여 중첩되는 기간이 표시됩니다. 명확히 이해하기 위해 다음 예제를 검토하세요.
	`SELECT CONVERT(DATETIME2(0), '2022-10-30T02:00:00', 126) AT TIME ZONE 'Central European Standard Time';`
	`GO;`
	`Result: 2022-10-30 02:00:00 +01:00`

열 별칭이 없는 빈 열 이름	`sqlcmd` 및 `psql` 유틸리티는 빈 이름이 있는 열을 다르게 처리합니다. • SQL Server `sqlcmd`는 빈 열 이름을 반환합니다. • PostgreSQL `psql`은 생성된 열 이름을 반환합니다.
CHECKSUM 함수	Babelfish와 SQL Server는 CHECKSUM 함수에 대해 서로 다른 해싱 알고리즘을 사용합니다. 따라서 Babelfish의 CHECKSUM 함수에서 생성된 해시 값은 SQL Server의 CHECKSUM 함수에서 생성된 해시 값과 다를 수 있습니다.
열 기본값	열 기본값을 생성할 때 제약 조건 이름은 무시됩니다. 열 기본값을 삭제하려면 `ALTER TABLE...ALTER COLUMN..DROP DEFAULT...` 구문을 사용합니다.
Constraint_name	SQL Server에서 제약 조건 이름은 표가 속한 스키마 내에서 고유해야 합니다. 그러나 Babelfish에서는 PRIMARY KEY 및 UNIQUE 제약 조건에만 적용됩니다. 다른 유형의 제약 조건에는 이 제한이 적용되지 않습니다.
제약 조건	PostgreSQL 개별 제약 조건 켜기 및 끄기를 지원하지 않습니다. 명령문이 무시되면서 경고가 발행됩니다.
IGNORE_DUP_KEY를 사용한 제약 조건	제약 조건은 이 속성 없이도 생성됩니다.

기능 또는 구문	동작 또는 차이점에 대한 설명
CREATE, ALTER, DROP SERVER ROLE	ALTER SERVER ROLE은 `sysadmin`에 대해서만 지원됩니다. 다른 모든 구문은 지원되지 않습니다. Babelfish의 T-SQL 사용자는 로그인(서버 주체), 데이터베이스 및 데이터베이스 사용자(데이터베이스 주체)의 개념에 대해 SQL Server와 유사한 경험을 합니다.
CREATE, ALTER LOGIN 절은 제한된 구문으로 지원	CREATE LOGIN... PASSWORD 절, ...DEFAULT_DATABASE 절 및 ...DEFAULT_LANGUAGE 절은 지원됩니다. ALTER LOGIN... PASSWORD 절은 지원되지만 ALTER LOGIN... OLD_PASSWORD 절은 지원되지 않습니다. sysadmin 구성원인 로그인만 암호를 수정할 수 있습니다.
CREATE DATABASE 대/소문자 구분하는 데이터 정렬	대소문자 구분하는 데이터 정렬은 CREATE DATABASE 문에서 지원되지 않습니다.
CREATE DATABASE 키워드 및 절	COLLATE 및 CONTAINMENT=NONE을 제외한 옵션은 지원되지 않습니다. COLLATE 절은 허용되며 항상 `babelfish pg_tsql.server_collation_name` 값으로 설정됩니다.
CREATE SCHEMA... 지원 절	CREATE SCHEMA 명령을 사용하여 빈 스키마를 만들 수 있습니다. 추가 명령을 사용하여 스키마 객체를 생성합니다.
Babelfish에서는 데이터베이스 ID 값이 다릅니다.	마스터 및 tempdb 데이터베이스는 데이터베이스 ID 1과 2가 아닙니다.
FORMAT 날짜 유형 함수는 다음과 같은 제한 사항으로 지원됩니다.	단일 문자 오전/오후는 지원되지 않습니다. SQL 서버의 'yyy' 형식은 1000 이상의 연도에 대해 4자리를 반환하지만 다른 연도에 대해서는 3자리만 반환합니다. 'g' 및 'R' 형식은 지원되지 않습니다. 'vi-VN' 로케일은 번역이 약간 다릅니다.

기능 또는 구문	동작 또는 차이점에 대한 설명
63자를 초과하는 식별자	PostgreSQL은 식별자에 대해 최대 63자까지 지원합니다. Babelfish는 63자보다 긴 식별자를 원래 이름의 해시를 포함하는 이름으로 변환합니다. 예를 들어, 'AB(ABC12345678901234567890123456789012345678901234567890123456 78901234567890'으로 생성된 테이블이 'ABC12345678901234 5678901234567890123456789012345678901234567890'으로 변환될 수 있습니다.
IDENTITY 열 지원	`IDENTITY` 열은 `tinyint`, `smallint`, `int`, `bigint`. `numeric` 및 `decimal`과 같은 데이터 유형에 대해 지원됩니다. SQL Server는 IDENTITY 열의 `numeric` 및 `decimal` 데이터 유형에 대해 38자리까지의 정밀도를 지원합니다. PostgreSQL은 IDENTITY 열의 `numeric` 및 `decimal` 데이터 유형에 대해 19자리까지의 정밀도를 지원합니다.
IGNORE_DUP_KEY가 포함된 인덱스	IGNORE_DUP_KEY가 포함된 인덱스를 생성하는 구문은 이 속성이 생략된 것처럼 인덱스를 만듭니다.
32개 이상의 열이 포함된 인덱스	인덱스에는 32개 이상의 열이 포함될 수 없습니다. 포함된 인덱스 열은 PostgreSQL의 최대값으로 계산되지만 SQL Server에서는 계산되지 않습니다.
인덱스(클러스터형)	클러스터형 인덱스는 NONCLUSTERED가 지정된 것처럼 만들어집니다.
인덱스 절	FILLFACTOR, ALLOW_PAGE_LOCKS, ALLOW_ROW _LOCKS, PAD_INDEX, STATISTICS_NORECOMPUTE, OPTIMIZE_FOR_SEQUENTIAL_KEY, SORT_IN_TEMPDB, DROP_EXISTING, ONLINE, COMPRESSION_DELAY, MAXDOP, DATA_COMPRESSION과 같은 절은 무시됩니다.
JSON 지원	이름-값 쌍의 순서는 보장되지 않습니다. 그러나 배열 유형은 영향을 받지 않습니다.

기능 또는 구문	동작 또는 차이점에 대한 설명
LOGIN 객체	PASSWORD, DEFAULT_DATABASE, DEFAULT_LANGUAGE, ENABLE, DISABLE을 제외하고 LOGIN 객체의 모든 옵션이 지원되지 않습니다.
NEWERENTIALID 함수	NEWID로 구현되고 순차적 동작은 보장되지 않습니다. `NEWSEQUENTIALID` 를 호출하는 경우 PostgreSQL은 새 GUID 값을 생성합니다.
OUTPUT 절은 다음 제한 사항을 통해 지원됩니다.	OUTPUT 및 OUTPUT INTO는 동일한 DML 쿼리에서 지원되지 않습니다. OUTPUT 절에서 UPDATE 또는 DELETE 작업의 대상이 아닌 테이블에 대한 참조는 지원되지 않습니다. OUTPUT... DELETED*, INSERTED*는 동일한 쿼리에서 지원되지 않습니다.
프로시저 또는 함수 파라미터 제한	Babelfish는 프로시저 또는 함수에 대해 최대 100개의 파라미터를 지원합니다.
ROWGUIDCOL	이 절은 현재 무시됩니다. `$GUIDGOL`을 참조하는 쿼리는 구문 오류를 일으킵니다.
SEQUENCE 객체 지원	SEQUENCE 객체는 tinyint, smallint, int, bigint, numeric 및 decimal과 같은 데이터 형식에 대해 지원됩니다. Aurora PostgreSQL은 SEQUENCE에서 numeric 및 decimal과 같은 데이터 유형에 대해 19자리까지의 정밀도를 지원합니다.
서버 수준 역할	`sysadmin` 서버 수준 역할이 지원됩니다. 다른 서버 수준 역할(`sysadmin` 이외)은 지원되지 않습니다.
`db_owner`를 제외한 데이터베이스 수준 역할	`db_owner` 데이터베이스 수준 역할과 사용자 정의 데이터베이스 수준 역할이 지원됩니다. 다른 데이터베이스 수준 역할(db_owner 이외)은 지원되지 않습니다.
SQL 키워드 `SPARSE`	키워드 SPARSE는 승인되고 무시됩니다.
SQL 키워드 절 `ON filegroup`	이 절은 현재 무시됩니다.

기능 또는 구문	동작 또는 차이점에 대한 설명
인덱스 및 제약 조건에 대한 SQL 키워드 `CLUSTERED` 및 `NONCLUSTERED`	Babelfish는 `CLUSTERED` 및 `NONCLUSTERED` 키워드를 수락하고 무시합니다.
`sysdatabases.cmptlevel`	`sysdatabases.cmptlevel` 은 항상 120으로 설정됩니다.
tempdb는 다시 시작 시 다시 초기화되지 않음	tempdb에서 생성된 영구 객체(예: 테이블 및 프로시저)는 데이터베이스를 다시 시작할 때 제거되지 않습니다.
TEXTIMAGE_ON 파일 그룹	Babelfish는 `TEXTIMAGE_ON` *`filegroup`* 절을 무시합니다.
시간 정밀도	Babelfish는 소수 초에 대해 6자리 정밀도를 지원합니다. 이 동작에는 부작용이 예상되지 않습니다.
트랜잭션 격리 수준	READUNCOMMITTED는 READCOMMITTED와 동일하게 취급됩니다.
가상 계산 열(비영구적)	가상 계산 열은 영구적으로 생성됩니다.
SCHEMABINDING 절 제외	이 절은 함수, 프로시저, 트리거 또는 뷰에는 지원되지 않습니다. WITH SCHEMABINDING이 지정된 것처럼 객체가 생성됩니다.

Babelfish의 트랜잭션 격리 수준

Babelfish는 트랜잭션 격리 수준 `READ UNCOMMITTED`, `READ COMMITTED`, `SNAPSHOT`을 지원합니다. Babelfish 3.4 버전부터 추가 격리 수준 `REPEATABLE READ` 및 `SERIALIZABLE`이 지원됩니다. Babelfish의 모든 격리 수준은 PostgreSQL의 해당 격리 수준 동작으로 지원됩니다. SQL Server와 Babelfish는 트랜잭션 격리 수준(동시 액세스 차단, 트랜잭션에 의한 잠금, 오류 처리 등)을 구현하기 위해 서로 다른 기본 메커니즘을 사용합니다. 또한 워크로드에 따라 동시 액세스가 작동하는 방식에도 미묘한 차이가 있습니다. 이 PostgreSQL 동작에 대한 자세한 내용은 [트랜잭션 격리](#)를 참조하세요.

주제

- [트랜잭션 격리 수준 개요](#)
- [트랜잭션 격리 수준 설정](#)
- [트랜잭션 격리 수준 활성화 또는 비활성화](#)
- [Babelfish 및 SQL Server 격리 수준 비교](#)

트랜잭션 격리 수준 개요

원래의 SQL Server 트랜잭션 격리 수준은 데이터 사본이 하나만 존재하고 쿼리가 행과 같은 리소스를 잠가야 액세스할 수 있는 수동적 잠금을 기반으로 합니다. 이후에 `READ COMMITTED` 격리 수준의 변형이 도입되었습니다. 이를 통해 행 버전을 사용하여 비차단 액세스를 사용하는 리더와 라이터 간의 동시성이 높아졌습니다. 이 외에도 `SNAPSHOT`이라는 새로운 격리 수준을 사용할 수 있습니다. 또한 행 버전을 사용하여 트랜잭션이 끝날 때까지 유지되는 읽기 데이터에 대한 공유 잠금을 피함으로써 `REPEATABLE READ` 격리 수준보다 더 나은 동시성을 제공합니다.

SQL Server와 달리 Babelfish의 모든 트랜잭션 격리 수준은 낙관적 잠금(MVCC)을 기반으로 합니다. 각 트랜잭션은 기본 데이터의 현재 상태에 관계없이 문의 시작 부분(`READ COMMITTED`) 또는 트랜잭션 시작 부분(`REPEATABLE READ`, `SERIALIZABLE`)에서 데이터 스냅샷을 확인할 수 있습니다. 따라서 Babelfish의 동시 트랜잭션 실행 동작은 SQL Server와 다를 수 있습니다.

SQL Server에서 처음에는 차단되었다가 나중에 성공하는 격리 수준 `SERIALIZABLE`의 트랜잭션을 예로 들어 보겠습니다. Babelfish에서는 동일한 행을 읽거나 업데이트하는 동시 트랜잭션과의 직렬화 충돌로 인해 실패로 끝날 수 있습니다. 동시 트랜잭션을 여러 개 실행하면 SQL Server와 비교하여 Babelfish의 최종 결과가 달라지는 경우도 생길 수 있습니다. 격리 수준을 사용하는 애플리케이션은 동시성 시나리오를 철저하게 테스트해야 합니다.

SQL Server 격리 수준	Babelfish 격리 수준	PostgreSQL 격리 수준	설명
READ UNCOMMITTED	READ UNCOMMITTED	READ UNCOMMITTED	READ UNCOMMITTED 는 Babelfish 또는 PostgreSQL의 READ COMMITTED 와 동일합니다.
READ COMMITTED	READ COMMITTED	READ COMMITTED	SQL Server READ COMMITTED 는 수동적 잠금 기반이고, Babelfish READ COMMITTED 는 스냅샷(MVCC) 기반입니다.
READ COMMITTED SNAPSHOT	READ COMMITTED	READ COMMITTED	둘 다 스냅샷(MVCC) 기반이지만 완전히 동일하지는 않습니다.
SNAPSHOT	SNAPSHOT	REPEATABLE READ	완전히 동일합니다.
REPEATABLE READ	REPEATABLE READ	REPEATABLE READ	SQL Server REPEATABLE READ는 수동적 잠금 기반이고, Babelfish REPEATABLE READ는 스냅샷(MVCC) 기반입니다.
SERIALIZABLE	SERIALIZABLE	SERIALIZABLE	SQL Server SERIALIZABLE 은 수동적 격리이고, Babelfish SERIALIZABLE 은 스냅샷(MVCC) 기반입니다.

> **Note**
>
> 테이블 힌트는 현재 지원되지 않으며 해당 동작은 Babelfish의 사전 정의된 이스케이프 해치 `escape_hatch_table_hints`를 사용하여 제어됩니다.

트랜잭션 격리 수준 설정

다음 명령을 사용하여 트랜잭션 격리 수준을 설정합니다.

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

트랜잭션 격리 수준 활성화 또는 비활성화

Babelfish에서는 트랜잭션 격리 수준 `REPEATABLE READ` 및 `SERIALIZABLE`이 기본적으로 비활성화되어 있으며, `sp_babelfish_configure`를 사용해 `babelfishpg_tsql.isolation_level_serializable` 또는 `babelfishpg_tsql.isolation_level_repeatable_read` 이스케이프 해치를 `pg_isolation`으로 설정하여 명시적으로 활성화해야 합니다. 자세한 내용은 [이스케이프 해치를 사용하여 Babelfish 오류 처리 관리](#) 단원을 참조하십시오.

다음은 각각의 이스케이프 해치를 설정하여 현재 세션에서 `REPEATABLE READ` 및 `SERIALIZABLE` 사용을 활성화 또는 비활성화하는 예제입니다. 필요에 따라 현재 세션뿐만 아니라 이후의 모든 새 세션에 대한 이스케이프 해치를 설정하는 `server` 파라미터를 포함하세요.

현재 세션에서만 `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`를 사용하도록 설정합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`를 현재 세션과 이후의 모든 새 세션에서 사용하도록 설정합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ를 현재 세션과 이후의 새 세션에서 사용할 수 없도록 합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

현재 세션에서만 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE를 사용하도록 설정합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE를 현재 세션과 이후의 모든 새 세션에서 사용하도록 설정합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE를 현재 세션과 이후의 새 세션에서 사용할 수 없도록 합니다.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

Babelfish 및 SQL Server 격리 수준 비교

다음은 SQL Server와 Babelfish가 ANSI 격리 수준을 구현하는 방식의 미묘한 차이에 대한 몇 가지 예입니다.

> **Note**
>
> - `REPEATABLE READ` 및 `SNAPSHOT` 격리 수준은 Babelfish에서 동일합니다.
> - `READ UNCOMMITTED` 및 `READ COMMITTED` 격리 수준은 Babelfish에서 동일합니다.

다음 예제에서는 아래에 언급된 모든 예제에 대한 기본 테이블을 생성하는 방법을 보여줍니다.

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

주제

SQL Server **`READ UNCOMMITTED`** 격리 수준과 비교한 Babelfish **`READ UNCOMMITTED`**

다음 표는 동시 트랜잭션이 실행될 때 더티 읽기에 대한 세부 정보를 제공합니다. 이는 Babelfish 구현과 비교하여 SQL Server에서 `READ UNCOMMITTED` 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **READ UNCOMMITTED**	Babelfish **READ UNCOMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	없음	없음
트랜잭션의 유휴 상태	`UPDATE employee SET age=0;`	업데이트 성공	업데이트 성공
트랜잭션의 유휴 상태	`INSERT INTO employee VALUES (4, 'D', 40);`	삽입 성공	삽입 성공
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	트랜잭션 1은 트랜잭션 2에서 커밋되지 않은 변경 사항을 확인할 수 있습니다.	Babelfish의 READ `COMMITTED` 와 동일합니다. 트랜잭션 2의 커밋되지 않은 변경 사항은 트랜잭션 1에 표시되지 않습니다.
트랜잭션의 유휴 상태	`COMMIT`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	트랜잭션 2에서 커밋된 변경 사항을 확인합니다.	트랜잭션 2에서 커밋된 변경 사항을 확인합니다.

SQL Server **READ COMMITTED** 격리 수준과 비교한 Babelfish **READ COMMITTED**

다음 표에서는 동시 트랜잭션이 실행될 때 읽기-쓰기 차단 동작에 대한 세부 정보를 제공합니다. 이는 Babelfish 구현과 비교하여 SQL Server에서 `READ COMMITTED` 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **READ COMMITTED**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`UPDATE employee SET age=100 WHERE id = 1;`	업데이트 성공	업데이트 성공
`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	트랜잭션의 유휴 상태	트랜잭션 2가 커밋될 때까지 단계가 차단됩니다.	트랜잭션 2 변경 사항은 아직 표시되지 않습니다. id = 3인 행을 업데이트합니다.
트랜잭션의 유휴 상태	`COMMIT`	트랜잭션 2가 성공적으로 커밋됩니다. 이제 트랜잭션 1의 차단이 해제되고 트랜잭션 2의 업데이트를 확인할 수 있습니다.	트랜잭션 2가 성공적으로 커밋됩니다.
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	트랜잭션 1이 id = 1인 행을 업데이트합니다.	트랜잭션 1이 id = 3인 행을 업데이트합니다.

SQL Server **READ COMMITTED SNAPSHOT** 격리 수준과 비교한 Babelfish **READ COMMITTED**

다음 표에서는 동시 트랜잭션이 실행될 때 새로 삽입된 행의 차단 동작에 대한 세부 정보를 제공합니다. 이는 `READ COMMITTED` Babelfish 구현과 비교하여 SQL Server에서 `READ COMMITTED SNAPSHOT` 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **READ COMMITTED SNAPSHOT**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	없음	없음
`INSERT INTO employee VALUES (4, 'D', 40);`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`UPDATE employee SET age = 99;`	트랜잭션 1가 커밋될 때까지 단계가 차단됩니다. 삽입된 행은 트랜잭션 1에 의해 잠깁니다.	3개 행이 업데이트됩니다. 새로 삽입된 행은 아직 표시되지 않습니다.
`COMMIT`	트랜잭션의 유휴 상태	커밋 성공. 이제 트랜잭션 2가 차단 해제됩니다.	커밋 성공.
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	4개 행 모두 age=99가 됩니다.	id = 4인 행은 업데이트 쿼리 중에 트랜잭션 2에 표시되지 않았으므로 age 값이 40입니다. 다른 행은 age=99로 업데이트됩니다.

SQL Server **REPEATABLE READ** 격리 수준과 비교한 Babelfish **REPEATABLE READ**

다음 표에서는 동시 트랜잭션이 실행될 때 읽기-쓰기 차단 동작에 대한 세부 정보를 제공합니다. 이는 REPEATABLE READ Babelfish 구현과 비교하여 SQL Server에서 REPEATABLE READ 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	없음	없음
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee WHERE id != 1;`	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	트랜잭션 2는 트랜잭션 1이 커밋될 때까지 차단됩니다.	트랜잭션 2는 정상적으로 진행됩니다.
`COMMIT`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	트랜잭션 1의 업데이트가 표시됩니다.	트랜잭션 1의 업데이트가 표시되지 않습니다.
`COMMIT`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	트랜잭션 1의 업데이트를 확인합니다.	트랜잭션 1의 업데이트를 확인합니다.

다음 표에서는 동시 트랜잭션이 실행될 때 쓰기-쓰기 차단 동작에 대한 세부 정보를 제공합니다. 이는 REPEATABLE READ Babelfish 구현과 비교하여 SQL Server에서 REPEATABLE READ 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	없음	없음
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`UPDATE employee SET name='A_TXN2' WHERE id=1;`	트랜잭션 2가 차단됩니다.	트랜잭션 2가 차단됩니다.
`COMMIT`	트랜잭션의 유휴 상태	커밋이 완료되고 트랜잭션 2가 차단 해제되었습니다.	동시 업데이트로 인해 액세스를 직렬화할 수 없다는 오류와 함께 커밋이 완료되고 트랜잭션 2가 실패합니다.
트랜잭션의 유휴 상태	`COMMIT`	커밋 성공.	트랜잭션 2는 이미 중단된 상태입니다.
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	id=1인 행에서 name='A_TX2'입니다.	id=1인 행에서 name='A_TX1'입니다.

다음 표에서는 동시 트랜잭션이 실행될 때 가상 읽기 동작에 대한 세부 정보를 제공합니다. 이는 REPEATABLE READ Babelfish 구현과 비교하여 SQL Server에서 REPEATABLE READ 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`INSERT INTO employee VALUES (4, 'NewRowName', 20);`	트랜잭션 2는 차단 없이 진행됩니다.	트랜잭션 2는 차단 없이 진행됩니다.
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	새로 삽입한 행이 표시됩니다.	새로 삽입한 행이 표시됩니다.
트랜잭션의 유휴 상태	`COMMIT`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	트랜잭션 2에서 삽입한 새 행이 표시됩니다.	트랜잭션 2에서 삽입한 새 행이 표시되지 않습니다.
`COMMIT`	트랜잭션의 유휴 상태	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	새로 삽입한 행이 표시됩니다.	새로 삽입한 행이 표시됩니다.

다음 표는 동시 트랜잭션이 실행되는 시기 및 `REPEATABLE READ` Babelfish 구현과 비교하여 SQL Server에서 `REPEATABLE READ` 격리 수준을 사용할 때의 다양한 최종 결과를 제공합니다.

트랜잭션 1	트랜잭션 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	없음	없음
`UPDATE employee SET age = 100 WHERE age IN (SELECT MIN(age) FROM employee);`	트랜잭션의 유휴 상태	트랜잭션 1이 id = 1인 행을 업데이트합니다.	트랜잭션 1이 id = 1인 행을 업데이트합니다.
트랜잭션의 유휴 상태	`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	SELECT 문이 트랜잭션 1의 UPDATE 쿼리로 잠긴 행을 읽으려고 하기 때문에 트랜잭션 2가 차단됩니다.	트랜잭션 2는 읽기가 차단되지 않으므로 차단 없이 진행됩니다. SELECT 문이 실행되고 마지막으로 트랜잭션 1의 변경 내용이 아직 표시되지 않으므로 id = 3인 행이 업데이트됩니다.
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	이 단계는 트랜잭션 1이 커밋된 후에 실행됩니다. id = 1인 행은 이전 단계의 트랜잭션 2에 의해 업데이트되며 여기에 표시됩니다.	트랜잭션 2가 id = 3인 행을 업데이트합니다.

트랜잭션 1	트랜잭션 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
COMMIT	트랜잭션의 유휴 상태	이제 트랜잭션 2가 차단 해제됩니다.	커밋 성공.
트랜잭션의 유휴 상태	COMMIT	없음	없음
SELECT * FROM employee;	트랜잭션의 유휴 상태	두 트랜잭션 모두 id = 1인 행에서 업데이트를 실행합니다.	트랜잭션 1과 2가 서로 다른 행을 업데이트합니다.

SQL Server **SERIALIZABLE** 격리 수준과 비교한 Babelfish **SERIALIZABLE**

다음 표에서는 동시 트랜잭션이 실행될 때 범위 잠금에 대한 세부 정보를 제공합니다. 이는 SERIALIZABLE Babelfish 구현과 비교하여 SQL Server에서 SERIALIZABLE 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
BEGIN TRANSACTION	BEGIN TRANSACTION	없음	없음
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	없음	없음
SELECT * FROM employee;	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	INSERT INTO employee VALUES (4, 'D', 35);	트랜잭션 2는 트랜잭션 1이 커밋될 때까지 차단됩니다.	트랜잭션 2는 차단 없이 진행됩니다.
트랜잭션의 유휴 상태	SELECT * FROM employee;	없음	없음

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`COMMIT`	트랜잭션의 유휴 상태	트랜잭션 1이 성공적으로 커밋됩니다. 이제 트랜잭션 2가 차단 해제됩니다.	트랜잭션 1이 성공적으로 커밋됩니다.
트랜잭션의 유휴 상태	`COMMIT`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	새로 삽입한 행이 표시됩니다.	새로 삽입한 행이 표시됩니다.

다음 표는 동시 트랜잭션이 실행되는 시기 및 `SERIALIZABLE` Babelfish 구현과 비교하여 SQL Server에서 `SERIALIZABLE` 격리 수준을 사용할 때의 다양한 최종 결과를 제공합니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	없음	없음
트랜잭션의 유휴 상태	`INSERT INTO employee VALUES (4, 'D', 40);`	없음	없음
`UPDATE employee SET age =99 WHERE id = 4;`	트랜잭션의 유휴 상태	트랜잭션 1은 트랜잭션 2가 커밋될 때까지 차단됩니다.	트랜잭션 1은 차단 없이 진행됩니다.
트랜잭션의 유휴 상태	`COMMIT`	트랜잭션 2가 성공적으로 커밋됩니다. 이제	트랜잭션 2가 성공적으로 커밋됩니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
		트랜잭션 1이 차단 해제됩니다.	
COMMIT	트랜잭션의 유휴 상태	없음	없음
SELECT * FROM employee;	트랜잭션의 유휴 상태	새로 삽입된 행의 age 값이 99로 표시됩니다.	새로 삽입된 행의 age 값이 40으로 표시됩니다.

다음 표는 고유한 제약 조건이 있는 테이블에 INSERT하는 경우의 세부 정보를 제공합니다. 이는 SERIALIZABLE Babelfish 구현과 비교하여 SQL Server에서 SERIALIZABLE 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
BEGIN TRANSACTION	BEGIN TRANSACTION	없음	없음
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	없음	없음
트랜잭션의 유휴 상태	INSERT INTO employee VALUES (4, 'D', 40);	없음	없음
INSERT INTO employee VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);	트랜잭션의 유휴 상태	트랜잭션 1은 트랜잭션 2가 커밋될 때까지 차단됩니다.	트랜잭션 1은 트랜잭션 2가 커밋될 때까지 차단됩니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
트랜잭션의 유휴 상태	`COMMIT`	트랜잭션 2가 성공적으로 커밋됩니다. 이제 트랜잭션 1이 차단 해제됩니다.	트랜잭션 2가 성공적으로 커밋됩니다. 키 값 중복 오류로 인해 트랜잭션 1이 중단되었으며 고유한 제약 조건을 위반했습니다.
`COMMIT`	트랜잭션의 유휴 상태	트랜잭션 1이 성공적으로 커밋됩니다.	트랜잭션 간의 읽기 또는 쓰기 종속성으로 인해 액세스를 직렬화할 수 없어 트랜잭션 1 커밋이 실패합니다.
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	행 (5, 'E', 50)이 삽입됩니다.	4개의 행만 존재합니다.

Babelfish에서 직렬화 가능(serializable) 격리 수준으로 실행되는 동시 트랜잭션은 이러한 트랜잭션의 실행이 해당 트랜잭션의 가능한 모든 직렬(한 번에 하나씩) 실행과 일치하지 않으면 직렬화 이상 오류가 발생하여 실패합니다.

다음 표에서는 동시 트랜잭션이 실행될 때 직렬화 이상에 대한 세부 정보를 제공합니다. 이는 `SERIALIZABLE` Babelfish 구현과 비교하여 SQL Server에서 `SERIALIZABLE` 격리 수준을 사용할 때 관찰된 결과를 보여줍니다.

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	없음	없음

트랜잭션 1	트랜잭션 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	없음	없음
`UPDATE employee SET age=5 WHERE age=10;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	트랜잭션 2는 트랜잭션 1이 커밋될 때까지 차단됩니다.	트랜잭션 2는 차단 없이 진행됩니다.
트랜잭션의 유휴 상태	`UPDATE employee SET age=35 WHERE age=30;`	없음	없음
`COMMIT`	트랜잭션의 유휴 상태	트랜잭션 1이 성공적으로 커밋됩니다.	트랜잭션 1이 먼저 커밋되고 커밋을 완료할 수 있습니다.
트랜잭션의 유휴 상태	`COMMIT`	트랜잭션 2가 성공적으로 커밋됩니다.	직렬화 오류로 인해 트랜잭션 2 커밋이 실패하고 전체 트랜잭션이 롤백되었습니다. 트랜잭션 2를 재시도합니다.
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	두 트랜잭션의 변경 사항이 모두 표시됩니다.	트랜잭션 2가 롤백되었습니다. 트랜잭션 1의 변경 사항만 표시됩니다.

Babelfish에서는 모든 동시 트랜잭션이 `SERIALIZABLE` 격리 수준에서 실행되는 경우에만 직렬화 이상이 발생할 수 있습니다. 다음 표에서는 위 예제를 사용하되 트랜잭션 2를 `REPEATABLE READ` 격리 수준으로 설정해 보겠습니다.

트랜잭션 1	트랜잭션 2	SQL Server 격리 수준	Babelfish 격리 수준
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	없음	없음
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	없음	없음
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	없음	없음
`UPDATE employee SET age=5 WHERE age=10;`	트랜잭션의 유휴 상태	없음	없음
트랜잭션의 유휴 상태	`SELECT * FROM employee;`	트랜잭션 2는 트랜잭션 1이 커밋될 때까지 차단됩니다.	트랜잭션 2는 차단 없이 진행됩니다.
트랜잭션의 유휴 상태	`UPDATE employee SET age=35 WHERE age=30;`	없음	없음
`COMMIT`	트랜잭션의 유휴 상태	트랜잭션 1이 성공적으로 커밋됩니다.	트랜잭션 1이 성공적으로 커밋됩니다.
트랜잭션의 유휴 상태	`COMMIT`	트랜잭션 2가 성공적으로 커밋됩니다.	트랜잭션 2가 성공적으로 커밋됩니다.
`SELECT * FROM employee;`	트랜잭션의 유휴 상태	두 트랜잭션의 변경 사항이 모두 표시됩니다.	두 트랜잭션의 변경 사항이 모두 표시됩니다.

구현이 제한된 Babelfish 기능 사용

Babelfish의 새로운 버전이 나올 때마다 T-SQL 기능 및 동작에 잘 맞는 여러 기능이 추가로 지원되고 있습니다. 하지만 현재 구현에는 지원되지 않는 기능 및 차이점이 몇 가지 있습니다. Babelfish와 T-

SQL의 기능적 차이에 대한 정보와 일부 해결 방법 또는 사용법 관련 참고 사항을 아래에서 확인할 수 있습니다.

Babelfish 버전 1.2.0의 경우 다음 기능이 현재 제한적으로 구현되어 있습니다.

- SQL Server 카탈로그(시스템 뷰) - 카탈로그 `sys.sysconfigures`,`sys.syscurconfigs` 및 `sys.configurations`에서 단일 읽기 전용 구성만 지원합니다. `sp_configure`는 현재 지원되지 않습니다. Babelfish에서 구현되는 다른 SQL Server 뷰에 대한 자세한 내용은 [Babelfish 시스템 카탈로그에서 정보 얻기](#) 섹션을 참조하세요.
- GRANT 권한 - GRANT...TO PUBLIC은 지원되지만 GRANT..TO PUBLIC WITH GRANT OPTION은 현재 지원되지 않습니다.
- SQL Server 소유권 체인 및 권한 메커니즘 제한 - Babelfish에서 SQL Server 소유권 체인은 뷰에 대해 작동하지만 저장 프로시저에 대해서는 작동하지 않습니다. 즉, 프로시저에는 호출 프로시저와 동일한 소유자가 소유한 다른 객체에 대한 명시적 액세스 권한이 부여되어야 합니다. SQL Server에서 호출자에게 프로시저에 대한 EXECUTE 권한을 부여하면 동일한 소유자가 소유한 다른 객체를 호출할 수 있습니다. Babelfish에서는 호출자에게 프로시저에서 액세스하는 객체에 대한 권한을 부여해야 합니다.
- 정규화되지 않은(스키마 이름 없음) 객체 참조 확인 - SQL 객체(프로시저, 뷰, 함수 또는 트리거)가 스키마 이름으로 한정하지 않고 객체를 참조하는 경우 SQL Server는 참조가 발생하는 SQL 객체의 스키마 이름을 사용하여 객체의 스키마 이름을 확인합니다. 현재 Babelfish는 프로시저를 실행하는 데이터베이스 사용자의 기본 스키마를 사용하여 이를 다르게 해결합니다.
- 기본 스키마 변경, 세션 및 연결 - 사용자가 `ALTER USER...WITH DEFAULT SCHEMA`를 사용하여 기본 스키마를 변경하면 변경 사항이 해당 세션에서 즉시 적용됩니다. 그러나 동일한 사용자에게 속한 현재 연결된 다른 세션의 경우 타이밍이 다음과 같이 다릅니다.
  - SQL Server의 경우: 변경 사항이 이 사용자의 다른 모든 연결에 즉시 적용됩니다.
  - Babelfish의 경우: 변경 사항이 이 사용자의 새 연결에만 적용됩니다.
- ROWVERSION 및 TIMESTAMP 데이터 유형 구현 및 이스케이프 해치 설정 - 이제 Babelfish에서 ROWVERSION 및 TIMESTAMP 데이터 유형이 지원됩니다. Babelfish에서 ROWVERSION 또는 TIMESTAMP를 사용하려면 이스케이프 해치 `babelfishpg_tsql.escape_hatch_rowversion`의 설정을 기본값(strict)에서 `ignore`로 변경해야 합니다. ROWVERSION 및 TIMESTAMP 데이터 유형의 Babelfish 구현은 대부분 SQL Server와 의미상 동일하지만 다음과 같은 예외가 있습니다.
  - 기본 제공 @@DBTS 함수는 SQL Server와 유사하게 동작하지만, 약간의 차이가 있습니다. Babelfish는 마지막으로 사용한 `SELECT @@DBTS` 값을 반환하는 대신 기본 PostgreSQL 데이터베이스 엔진과 다중 버전 동시성 제어(MVCC) 구현으로 인해 새로운 타임스탬프를 생성합니다.

  - SQL Server에서는 삽입되거나 업데이트된 모든 행이 고유한 ROWVERSION/TIMESTAMP 값을 갖습니다. Babelfish에서는 동일한 명령문에 의해 업데이트된 모든 삽입된 행에 동일한 ROWVERSION/TIMESTAMP 값이 할당됩니다.

    예를 들어 UPDATE 문 또는 INSERT-SELECT 문이 여러 행에 영향을 미치는 경우 SQL Server에서는 영향을 받는 행이 모두 ROWVERSION/TIMESTAMP 열에 서로 다른 값을 갖습니다. Babelfish(PostgreSQL)에서는 그러한 행이 동일한 값을 갖습니다.
  - SQL Server에서는 SELECT-INTO를 사용하여 새 테이블을 만들 때 생성되는 ROWVERSION/TIMESTAMP 열에 명시적 값(예: NULL)을 캐스팅할 수 있습니다. Babelfish에서는 동일한 작업을 수행하면 Babelfish에 의해 새 테이블의 각 행에 실제 ROWVERSION/TIMESTAMP 값이 할당됩니다.

  ROWVERSION/TIMESTAMP 데이터 유형의 이러한 사소한 차이점은 Babelfish에서 실행되는 애플리케이션에 부정적인 영향을 미치지 않습니다.
- 스키마 생성, 소유권 및 권한 - 비DBO 사용자가 소유하는 스키마에서 객체를 생성 및 액세스할 수 있는 권한(`CREATE SCHEMA` *`schema name`* `AUTHORIZATION` *`user name`* 사용)은 다음 표와 같이 SQL Server와 Babelfish 비DBO 사용자 간에 다릅니다.

스키마를 소유하는 데이터베이스 사용자(비DBO)가 수행할 수 있는 작업:	SQL Server	Babelfish
DBO의 추가 권한 부여 없이 스키마에 객체 생성	아니요	예
추가 권한 부여 없이 DBO가 스키마에서 생성한 객체에 액세스할 수 있나요?	예	아니요

- CREATE OR ALTER VIEW / ALTER VIEW 구문 - Babelfish에서 이러한 구문에 대한 지원에는 다음과 같은 제한 사항이 있습니다.
  - 이러한 문은 INSTEAD-OF 트리거가 연결된 뷰에서 사용할 수 없습니다.
  - 이러한 문은 이 뷰를 기반으로 다른 뷰가 있는 뷰에는 사용할 수 없습니다.

Babelfish 쿼리 성능 개선

쿼리 힌트와 PostgreSQL 최적화 프로그램을 사용하여 Babelfish에서 빠르게 쿼리를 처리할 수 있습니다.

주제

- 설명 계획을 사용하여 Babelfish 쿼리 성능 향상
- T-SQL 쿼리 힌트를 사용하여 Babelfish 쿼리 성능 향상

`sp_babelfish_volatility` 프로시저를 사용하여 쿼리 성능을 개선할 수도 있습니다. 자세한 내용은 sp_babelfish_volatility 섹션을 참조하세요.

하위 쿼리 변환 및 하위 쿼리 캐시를 사용하여 쿼리 성능을 개선할 수도 있습니다. 자세한 내용은 Aurora PostgreSQL에서 상관관계가 있는 하위 쿼리 최적화 섹션을 참조하세요.

설명 계획을 사용하여 Babelfish 쿼리 성능 향상

버전 2.1.0부터 Babelfish에는 PostgreSQL 옵티마이저를 투명하게 사용하여 TDS 포트에서 T-SQL 쿼리에 대한 예상 및 실제 쿼리 계획을 생성하는 두 함수가 포함됩니다. 이러한 함수는 SQL Server 데이터베이스에서 SET STATISTICS PROFILE 또는 SET SHOWPLAN_ALL을 사용하여 실행 속도가 느린 쿼리를 식별하고 개선하는 것과 유사합니다.

> **Note**
> 함수, 제어 흐름 및 커서에서 쿼리 계획을 가져오는 작업은 현재 지원되지 않습니다.

이 테이블에서는 SQL Server, Babelfish, PostgreSQL의 쿼리 계획 설명 함수 비교 정보를 확인할 수 있습니다.

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	EXPLAIN ANALYZE

SQL Server	Babelfish	PostgreSQL
SQL Server 옵티마이저 사용	PostgreSQL 옵티마이저 사용	PostgreSQL 옵티마이저 사용
SQL Server 입력 및 출력 형식	SQL Server 입력 및 PostgreSQL 출력 형식	PostgreSQL 입력 및 출력 형식
세션에 대해 설정	세션에 대해 설정	특정 문에 적용
다음을 지원합니다. • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE • 제어 흐름을 포함한 함수 및 EXEC(CASE, WHILE-BREAK-CONTINUE, WAITFOR, BEGIN-END, IF-ELSE 등)	다음을 지원합니다. • SELECT • INSERT • UPDATE • DELETE • CREATE • EXECUTE • EXEC • RAISEERROR • THROW • PRINT • USE	다음을 지원합니다. • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE

다음과 같이 Babelfish 함수를 사용합니다.

- SET BABELFISH_SHOWPLAN_ALL[ON|OFF] – 예상 쿼리 실행 계획을 생성하려면 ON으로 설정합니다. 이 함수는 PostgreSQL `EXPLAIN` 명령의 동작을 구현합니다. 이 명령을 사용하여 지정된 쿼리에 대한 설명 계획을 가져옵니다.
- SET BABELFISH_STATISTICS PROFILE[ON|OFF] – 실제 쿼리 실행 계획에 대해 ON으로 설정합니다. 이 함수는 PostgreSQL `EXPLAIN ANALYZE` 명령의 동작을 구현합니다.

PostgreSQL `EXPLAIN` 및 `EXPLAIN ANALYZE`에 대한 자세한 내용은 PostgreSQL 설명서의 EXPLAIN을 참조하세요.

Note

버전 2.2.0부터 SHOWPLAN_ALL 및 STATISTICS PROFILE SET 명령에 대한 SQL Server 구문에서 BABELFISH_ 접두사를 사용하지 않도록 escape_hatch_showplan_all 파라미터를 무시하도록 설정할 수 있습니다.

예를 들어, 다음 명령 시퀀스는 쿼리 계획을 설정한 다음 쿼리를 실행하지 않고 SELECT 문에 대한 예상 쿼리 실행 계획을 반환합니다. 이 예에서는 sqlcmd 명령줄 도구를 통해 SQL Server 샘플 northwind 데이터베이스를 사용하여 TDS 포트를 쿼리합니다.

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


-----------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

쿼리 검토 및 조정을 마치면 다음과 같이 함수를 해제합니다.

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

BABELFISH_STATISTICS PROFILE을 ON으로 설정하면 실행된 각 쿼리는 일반 결과 세트를 반환한 후 실제 쿼리 실행 계획을 보여주는 추가 결과 세트를 반환합니다. Babelfish는 SELECT 문을 호출할 때 가장 빠른 결과 세트를 제공하는 쿼리 계획을 생성합니다.

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

결과 세트와 쿼리 계획이 반환되며, 이 예에는 쿼리 계획만 표시되어 있습니다.

```
QUERY PLAN



---------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

PostgreSQL 옵티마이저에서 반환한 쿼리 및 결과를 분석하는 방법에 대한 자세한 내용은 explain.depesz.com을 참조하세요. PostgreSQL EXPLAIN 및 EXPLAIN ANALYZE에 대한 자세한 내용은 PostgreSQL 설명서의 EXPLAIN을 참조하세요.

Babelfish 설명 옵션을 제어하는 파라미터

다음 테이블에 표시된 파라미터를 사용하여 쿼리 계획에 표시되는 정보 유형을 제어할 수 있습니다.

파라미터	설명
babelfishpg_tsql.explain_buffers	옵티마이저에 대한 버퍼 사용 정보를 표시하거나 숨기는 부울입니다. (기본값: off)(허용: off, on)
babelfishpg_tsql.explain_costs	옵티마이저에 대한 예상 시작 및 총 비용 정보를 표시하거나 숨기는 부울입니다. (기본값: on)(허용: off, on)
babelfishpg_tsql.explain_format	`EXPLAIN` 계획에 대한 출력 형식을 지정합니다. (기본값: text)(허용: text, xml, json, yaml)
babelfishpg_tsql.explain_settings	EXPLAIN 계획 출력에 구성 파라미터에 대한 정보가 포함되도록 설정하거나 해제하는 부울입니다. (기본값: off)(허용: off, on)
babelfishpg_tsql.explain_summary	쿼리 계획 이후의 총 시간과 같은 요약 정보를 표시하거나 숨기는 부울입니다. (기본값: on)(허용: off, on)
babelfishpg_tsql.explain_timing	실제 시작 시간 및 출력의 각 노드에 소요된 시간을 표시하거나 숨기는 부울입니다. (기본값: on)(허용: off, on)
babelfishpg_tsql.explain_verbose	설명 계획의 가장 상세한 버전을 표시하거나 숨기는 부울입니다. (기본값: off)(허용: off, on)
babelfishpg_tsql.explain_wal	설명 계획의 일부로 WAL 기록 정보의 생성을 설정하거나 해제하는 부울입니다. (기본값: off)(허용: off, on)

PostgreSQL 클라이언트 또는 SQL Server 클라이언트를 사용하여 시스템에서 Babelfish 관련 파라미터 값을 확인할 수 있습니다. 다음 명령을 실행하여 현재 파라미터 값을 가져옵니다.

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

다음 출력에서 이 특정 Babelfish DB 클러스터의 모든 설정이 기본값임을 알 수 있습니다. 이 예에서는 일부 출력이 표시되지 않습니다.

```
             name                      setting                    short_desc
---------------------------------- --------
 --------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

다음 예제와 같이 `sp_babelfish_configure`를 사용하여 이러한 파라미터의 설정을 변경할 수 있습니다.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

클러스터 전체 수준에서 설정을 영구적으로 만들려면 다음 예제와 같이 키워드 server를 포함합니다.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

T-SQL 쿼리 힌트를 사용하여 Babelfish 쿼리 성능 향상

버전 2.3.0부터 Babelfish는 `pg_hint_plan`을 사용하여 쿼리 힌트 사용을 지원합니다. Aurora PostgreSQL에서는 `pg_hint_plan`이 기본적으로 설치됩니다. PostgreSQL 확장에 `pg_hint_plan`에 대한 자세한 내용은 https://github.com/ossc-db/pg_hint_plan 섹션을 참조하세요.

Aurora PostgreSQL에서 지원하는 이 확장 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보에서 Amazon Aurora PostgreSQL의 확장 버전을 참조하세요.

쿼리 최적화 프로그램은 SQL 문에 대한 최적의 실행 계획을 찾도록 설계되었습니다. 계획을 선택할 때 쿼리 최적화 프로그램은 엔진의 비용 모델과 열 및 테이블 통계를 모두 고려합니다. 하지만 제안된 계획이 데이터 세트의 요구 사항을 충족하지 못할 수 있습니다. 따라서 쿼리 힌트는 성능 문제를 해결하여 실행 계획을 개선합니다. `query hint`는 쿼리 실행 방법에 대해 데이터베이스 엔진에 지시하는 SQL 표준에 추가된 구문입니다. 예를 들어 힌트는 엔진이 순차 스캔을 따르고 쿼리 최적화 프로그램이 선택한 계획을 재정의하도록 지시할 수 있습니다.

Babelfish에서 T-SQL 쿼리 힌트 켜기

현재 Babelfish는 기본적으로 모든 T-SQL 힌트를 무시합니다. T-SQL 힌트를 적용하려면 enable_pg_hint 값을 ON으로 설정하여 `sp_babelfish_configure` 명령을 실행합니다.

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

server 키워드를 포함하여 클러스터 전체 수준에서 설정을 영구적으로 만들 수 있습니다. 현재 세션에 대해서만 설정을 구성하려면 서버를 사용하지 마세요.

`enable_pg_hint`가 ON으로 설정되면 Babelfish는 다음 T-SQL 힌트를 적용합니다.

- INDEX 힌트
- JOIN 힌트
- FORCE ORDER 힌트
- MAXDOP 힌트

예를 들어, 다음 명령 시퀀스가 `pg_hint_plan`을 켭니다.

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

SELECT 문에는 힌트가 적용되지 않습니다. 힌트가 없는 쿼리 계획이 반환됩니다.

```
QUERY PLAN



---------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

쿼리 힌트는 SELECT 문에 적용됩니다. 다음 출력은 병합 조인이 포함된 쿼리 계획이 반환되었음을 보여줍니다.

```
QUERY PLAN



---------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

제한 사항

쿼리 힌트를 사용하는 동안 다음 제한 사항을 고려하세요.

- enable_pg_hint가 설정되기 전에 쿼리 계획이 캐시되면 동일한 세션에서 힌트가 적용되지 않습니다. 새 세션에 적용됩니다.
- 스키마 이름이 명시적으로 입력되면 힌트를 적용할 수 없습니다. 해결 방법으로 테이블 별칭을 사용할 수 있습니다.
- 쿼리 힌트는 뷰 및 하위 쿼리에 적용할 수 없습니다.
- JOIN이 포함된 UPDATE/DELETE 문에는 힌트가 적용되지 않습니다.
- 존재하지 않는 인덱스 또는 테이블에 대한 인덱스 힌트는 무시됩니다.
- FORCE ORDER 힌트는 HASH JOIN 및 비 ANSI JOIN에 적용되지 않습니다.

Babelfish와 함께 Aurora PostgreSQL 확장 사용

Aurora PostgreSQL은 다른 AWS 서비스와의 작업을 위한 확장을 제공합니다. 이는 데이터 가져오기 또는 내보내기를 위해 DB 클러스터와 함께 Amazon S3를 사용하는 것과 같은 다양한 사용 사례를 지원하는 선택적 확장입니다.

- Amazon S3 버킷에서 Babelfish DB 클러스터로 데이터를 가져오려면 aws_s3 Aurora PostgreSQL 확장을 설정합니다. 이 확장을 사용하면 Aurora PostgreSQL DB 클러스터에서 Amazon S3 버킷으로 데이터를 내보낼 수도 있습니다.
- AWS Lambda은(는) 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있게 해주는 컴퓨팅 서비스입니다. Lambda 함수를 사용하여 DB 인스턴스의 이벤트 알림 처리와 같은 작업을 수행할 수 있습니다. Lambda에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda란 무엇입니까?를 참조하세요. Babelfish DB 클러스터에서 Lambda 함수를 호출하려면 aws_lambda Aurora PostgreSQL 확장을 설정합니다.

Babelfish 클러스터에 대해 이러한 확장을 설정하려면 먼저 내부 Babelfish 사용자에게 확장을 로드할 수 있는 권한을 부여해야 합니다. 권한을 부여한 후 Aurora PostgreSQL 확장을 로드할 수 있습니다.

Babelfish DB 클러스터에서 Aurora PostgreSQL 확장 사용 설정

aws_s3 또는 aws_lambda 확장을 로드하려면 Babelfish DB 클러스터에 필요한 권한을 부여해야 합니다.

다음 절차에서는 psql PostgreSQL 명령줄 도구를 사용하여 DB 클러스터에 연결합니다. 자세한 내용은 psql을 사용하여 DB 클러스터에 연결 섹션을 참조하세요. pgAdmin을 사용할 수도 있습니다. 자세한 내용은 pgAdmin을 사용하여 DB 클러스터에 연결을 참조하세요.

이 절차는 aws_s3과 aws_lambda를 차례로 로드합니다. 이러한 확장 중 하나만 사용하려면 둘 다 로드할 필요가 없습니다. aws_commons 확장은 각각에 필요하며 출력에 표시된 대로 기본적으로 로드됩니다.

Aurora PostgreSQL 확장에 대한 권한으로 Babelfish DB 클러스터 설정

1. Babelfish DB 클러스터에 연결합니다. Babelfish DB 클러스터를 생성할 때 지정한 "마스터" 사용자(-U)의 이름을 사용합니다. 기본값(postgres)은 예제에 표시되어 있습니다.

   대상 LinuxmacOS, 또는Unix:

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
-U postgres \
-d babelfish_db \
-p 5432
```

   Windows의 경우:

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
-U postgres ^
-d babelfish_db ^
-p 5432
```

   명령은 사용자 이름(-U)의 암호를 입력하라는 프롬프트로 응답합니다.

```
Password:
```

   DB 클러스터의 사용자 이름(-U)에 대한 암호를 입력합니다. 성공적으로 연결하면 다음과 비슷한 출력이 표시됩니다.

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

2. 내부 Babelfish 사용자에게 확장을 생성하고 로드할 수 있는 권한을 부여합니다.

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
```

```
GRANT ROLE
```

3. aws_s3 확장 생성 및 로드 aws_commons 확장이 필요하며 aws_s3이 설치되면 자동으로 설치됩니다.

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. aws_lambda 확장 생성 및 로드

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

Amazon S3에서 Babelfish 사용

Babelfish DB 클러스터에 사용할 Amazon S3 버킷이 아직 없는 경우 하나 생성할 수 있습니다. 사용하려는 Amazon S3 버킷에 대해 액세스 권한을 제공합니다.

Amazon S3 버킷을 사용하여 데이터를 가져오거나 내보내기 전에 다음 일회성 단계를 완료하세요.

Amazon S3 버킷에 대한 Babelfish DB 인스턴스의 액세스 설정

1. 필요한 경우 Babelfish 인스턴스용 Amazon S3 버킷을 생성합니다. 이렇게 하려면 Amazon Simple Storage Service 사용 설명서의 Create a bucket 지침을 따르세요.
2. Amazon S3 버킷에 파일을 업로드합니다. 이렇게 하려면 Amazon Simple Storage Service 사용 설명서의 Add an object to a bucket 단계를 따르세요.
3. 필요에 따라 권한을 설정합니다.
   - Amazon S3에서 데이터를 가져오려면 Babelfish DB 클러스터에 버킷 액세스 권한이 필요합니다. AWS Identity and Access Management(IAM) 역할을 사용하고 클러스터의 해당 역할에 IAM 정책을 연결하는 것이 좋습니다. 이렇게 하려면 IAM 역할을 사용해 Amazon S3 버킷에 액세스 단원의 절차를 따르세요.
   - Babelfish DB 클러스터에서 데이터를 내보내려면 클러스터에 Amazon S3 버킷에 대한 액세스 권한이 부여되어야 합니다. 가져오기와 마찬가지로 IAM 역할 및 정책을 사용하는 것이 좋습니다. 이렇게 하려면 Amazon S3 버킷에 대한 액세스 권한 설정 단원의 절차를 따르세요.

이제 Babelfish DB 클러스터에서 `aws_s3` 확장과 함께 Amazon S3를 사용할 수 있습니다.

Amazon S3에서 Babelfish로 데이터 가져오기 및 Amazon S3로 Babelfish 데이터 내보내기

1. Babelfish DB 클러스터와 함께 `aws_s3` 확장을 사용합니다.

   이때 PostgreSQL 컨텍스트에 있는 테이블을 참조해야 합니다. 즉, `[database].[schema].[tableA]`라는 Babelfish 테이블로 가져오려면 `aws_s3` 함수에서 해당 테이블을 `database_schema_tableA`로 참조합니다.

   - `aws_s3` 함수를 사용하여 데이터를 가져오는 예는 [Amazon S3에서 Aurora PostgreSQL DB 클러스터로 데이터 가져오기]() 섹션을 참조하세요.
   - `aws_s3` 함수를 사용하여 데이터를 내보내는 예는 [aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기]() 섹션을 참조하세요.
2. 다음 표와 같이 `aws_s3` 확장 및 Amazon S3를 사용할 때 PostgreSQL 이름 지정을 사용하여 Babelfish 테이블을 참조해야 합니다.

Babelfish 테이블	Aurora PostgreSQL 테이블
database.*schema*.*table*	*database*_*schema*_*table*

Aurora PostgreSQL과 함께 Amazon S3를 사용하는 방법에 대해 자세히 알아보려면 [PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터]() 및 [Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기]() 섹션을 참조하세요.

AWS Lambda와 함께 Babelfish 사용

`aws_lambda` 확장이 Babelfish DB 클러스터에 로드된 후 이 절차에 따라 Lambda에 DB 클러스터에 대한 액세스 권한을 부여해야 Lambda 함수를 호출할 수 있습니다.

Lambda와 작동하도록 Babelfish DB 클러스터에 대한 액세스 설정

이 절차에서는 AWS CLI를 사용하여 IAM 정책, 역할을 생성하고 이를 Babelfish DB 클러스터와 연결합니다.

1. Babelfish DB 클러스터에서 Lambda에 대한 액세스를 허용하는 IAM 정책을 생성합니다.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. 정책이 런타임에 수임할 수 있는 IAM 역할을 생성합니다.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. 책을 역할에 연결합니다.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. Babelfish DB 클러스터에 역할을 연결합니다.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

이러한 태스크를 완료한 후 Lambda 함수를 호출할 수 있습니다. AWS Lambda를 사용하여 Aurora PostgreSQL DB 클러스터에 대해 AWS Lambda를 설정하는 방법에 대한 자세한 내용과 예는 2단계: Aurora PostgreSQL DB 클러스터 및 AWS Lambda에 대한 IAM 구성 섹션을 참조하세요.

Babelfish DB 클러스터에서 Lambda 함수 호출

AWS Lambda는 Java, Node.js, Python, Ruby 및 기타 언어로 작성된 기능을 지원합니다. 함수가 호출될 때 텍스트를 반환하면 Babelfish DB 클러스터에서 이를 호출할 수 있습니다. 다음 예는 인사말을 반환하는 자리 표시자 python 함수입니다.

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

현재 Babelfish는 JSON을 지원하지 않습니다. 함수가 JSON을 반환하는 경우 래퍼를 사용하여 JSON을 처리합니다. 예를 들어, 앞에 표시된 `lambda_function.py`가 Lambda에 `my-function`으로 저장되어 있다고 가정해 보겠습니다.

1. `psql` 클라이언트 또는 pgAdmin 클라이언트를 사용하여 Babelfish DB 클러스터에 연결합니다. 자세한 내용은 psql을 사용하여 DB 클러스터에 연결 섹션을 참조하세요.
2. 래퍼를 생성합니다. 이 예에서는 SQL용 PostgreSQL의 절차적 언어인 `PL/pgSQL`을 사용합니다. 자세한 내용은 PL/pgSQL–SQL 절차적 언어를 참조하세요.

```
create or replace function master_dbo.lambda_wrapper()
returns text
language plpgsql
as
$$
declare
   r_status_code integer;
   r_payload text;
begin
   SELECT payload INTO r_payload
     FROM aws_lambda.invoke(   aws_commons.create_lambda_function_arn('my-function',
 'us-east-1')
                              ,'{"body": "Hello from Postgres!"}'::json );
   return r_payload ;
```

```
end;
$$;
```

이제 이 함수는 Babelfish TDS 포트(1433) 또는 PostgreSQL 포트(5433)에서 실행할 수 있습니다.

a. PostgreSQL 포트에서 이 함수 호출

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

출력은 다음과 유사합니다.

```
status_code |                        payload                         |
 executed_version | log_result
-------------+-------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. TDS 포트에서 이 기능을 호출하려면 SQL Server `sqlcmd` 명령줄 클라이언트를 사용하여 포트에 연결합니다. 자세한 내용은 SQL Server 클라이언트 도구를 사용하여 DB 클러스터에 연결을 참조하세요. 연결되면 다음을 실행합니다.

```
1> select lambda_wrapper();
2> go
```

이 명령은 다음과 비슷한 출력을 반환합니다.

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

Aurora PostgreSQL과 함께 Lambda를 사용하는 방법에 대해 자세히 알아보려면 Aurora PostgreSQL DB 클러스터에서 AWS Lambda 함수 호출 섹션을 참조하세요. Lambda 함수 작업에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 시작하기를 참조하세요.

Babelfish 에서 pg_stat_statements 사용

Babelfish for Aurora PostgreSQL은 3.3.0에서 `pg_stat_statements` 확장을 지원합니다. 자세한 내용은 [pg_stat_statements](#)를 참조하세요.

Aurora PostgreSQL에서 지원하는 이 확장에 대한 자세한 내용은 [확장 버전](#)을 참조하세요.

pg_stat_statements 확장 만들기

`pg_stat_statements`를 활성화하려면 쿼리 식별자 계산을 켜야 합니다. 이 작업은 파라미터 그룹에서 `compute_query_id`가 on 또는 `auto`로 설정되어 있으면 자동으로 수행됩니다. `compute_query_id` 파라미터의 기본값은 `auto`입니다. 이 기능을 켜려면 이 확장도 생성해야 합니다. 다음 명령을 사용하여 T-SQL 엔드포인트에서 확장을 설치하세요.

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

다음 쿼리를 사용하여 쿼리 통계에 액세스할 수 있습니다.

```
postgres=>select * from pg_stat_statements;
```

> **Note**
> 설치 중에 확장의 스키마 이름을 제공하지 않으면 기본적으로 확장이 퍼블릭 스키마에 생성됩니다. 액세스하려면 아래와 같이 스키마 한정자와 함께 대괄호를 사용해야 합니다.

```
postgres=>select * from [public].pg_stat_statements;
```

PSQL 엔드포인트에서 확장을 생성할 수도 있습니다.

확장 승인

기본적으로 승인 없이 T-SQL 데이터베이스 내에서 수행된 쿼리의 통계를 볼 수 있습니다.

다른 사람이 만든 쿼리 통계에 액세스하려면 `pg_read_all_stats` PostgreSQL 역할이 있어야 합니다. 아래에 설명된 단계에 따라 GRANT pg_read_all_stats 명령을 생성합니다.

1. T-SQL에서는 내부 PG 역할 이름을 반환하는 다음 쿼리를 사용합니다.

   ```
   SELECT rolname FROM pg_roles WHERE oid = USER_ID();
   ```

2. rds_superuser 권한으로 Babelfish for Aurora PostgreSQL 데이터베이스에 연결하고 다음 명령을 사용합니다.

   ```
   GRANT pg_read_all_stats TO <rolname_from_above_query>
   ```

예제

T-SQL 엔드포인트에서:

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```


rolname
-------
master_dbo
(1 rows affected)
```

PSQL 엔드포인트에서:

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

pg_stat_statements 뷰를 사용하여 쿼리 통계에 액세스할 수 있습니다.

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
```

```
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

쿼리 통계 재설정

`pg_stat_statements_reset()`을 사용하여 지금까지 pg_stat_statements에서 수집한 통계를 재설정할 수 있습니다. 자세한 내용은 [pg_stat_statements](#)를 참조하세요. 현재 PSQL 엔드포인트를 통해서만 지원됩니다. `rds_superuser` 권한을 사용하여 Babelfish for Aurora PostgreSQL에 연결하려면 다음 명령을 사용하세요.

```
SELECT pg_stat_statements_reset();
```

제한 사항

- 현재 T-SQL 엔드포인트를 통해서는 `pg_stat_statements()`가 지원되지 않습니다. 통계를 수집할 때는 `pg_stat_statements` 뷰를 사용하는 것이 좋습니다.
- 일부 쿼리는 Aurora PostgreSQL 엔진으로 구현된 T-SQL 파서에 의해 다시 작성될 수 있습니다. `pg_stat_statements` 뷰에는 원래 쿼리가 아닌 다시 작성된 쿼리가 표시됩니다.

  예제

  ```
  select next value for [dbo].[newCounter];
  ```

  위의 쿼리는 pg_stat_statements 뷰에서 다음과 같이 다시 작성됩니다.

  ```
  select nextval($1);
  ```

- 문의 실행 흐름에 따라 일부 쿼리는 pg_stat_statements에서 추적되지 않고 뷰에 표시되지 않을 수 있습니다. 여기에 포함되는 문은 `use dbname`, `goto`, `print`, `raise error`, `set`, `throw`, `declare cursor`입니다.
- CREATE LOGIN 및 ALTER LOGIN 문의 경우 쿼리 및 쿼리 ID가 표시되지 않습니다. 권한이 충분하지 않다고 표시됩니다.
- `pg_stat_statements` 뷰에는 항상 아래 두 항목이 포함됩니다. 이 두 항목은 `sqlcmd` 클라이언트가 내부적으로 실행하기 때문입니다.
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4,096

Babelfish에서 pgvector 사용

오픈 소스 확장인 pgvector를 사용하면 Postgres 데이터베이스 내에서 직접 유사한 데이터를 검색할 수 있습니다. Babelfish는 이제 버전 15.6 및 16.2부터 이 확장을 지원합니다. 자세한 내용은 [pgvector 오픈 소스 설명서]를 참조하세요.

사전 조건

pgvector 기능을 활성화하려면 다음 방법 중 하나를 사용하여 sys 스키마에 확장을 설치하세요.

- sqlcmd 클라이언트에서 다음 명령을 실행합니다.

```
exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
```

- `babelfish_db`에 연결하고 psql 클라이언트에서 다음 명령을 실행합니다.

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> pgvector 확장을 설치한 후에는 설정한 새 데이터베이스 연결에서만 해당 벡터 데이터 유형을 사용할 수 있습니다. 기존 연결에서는 새 데이터 유형을 인식하지 못합니다.

지원되는 기능

Babelfish는 T-SQL 기능을 확장하여 다음을 지원합니다.

- 저장

  Babelfish는 이제 벡터 데이터 유형 호환 구문을 지원하여 T-SQL 호환성을 개선합니다. pgvector를 사용하여 데이터를 저장하는 방법에 대한 자세한 내용은 [Storing](#)을 참조하세요.

- 쿼리

  Babelfish는 벡터 유사성 연산자를 포함하도록 T-SQL 표현식 지원을 확장합니다. 그러나 다른 모든 쿼리의 경우 표준 T-SQL 구문이 여전히 필요합니다.

  > **Note**
  >
  > T-SQL은 배열 유형을 지원하지 않으며 데이터베이스 드라이버에는 배열 유형을 처리하는 인터페이스가 없습니다. 해결 방법으로 Babelfish는 텍스트 문자열(varchar/nvarchar)을 사용하여 벡터 데이터를 저장합니다. 예를 들어 벡터 값 [1,2,3]을 요청하면 Babelfish는 '[1,2,3]' 문자열을 응답으로 반환합니다. 필요에 따라 애플리케이션 수준에서 이 문자열을 구문 분석하고 분할할 수 있습니다.

  pgvector를 사용하여 데이터를 쿼리하는 방법에 대한 자세한 내용은 [Querying](#)을 참조하세요.

- 인덱싱

  T-SQL `Create Index`는 이제 `USING INDEX_METHOD` 구문을 지원합니다. 이제 인덱스를 만들 때 특정 열에 사용할 유사성 검색 연산자를 정의할 수 있습니다.

  또한 문법이 필수 열에 대한 벡터 유사성 연산을 지원하도록 확장되었습니다(column_name_list_with_order_for_vector 문법 확인).

  ```
  CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
   vector_index_method] (<column_name_list_with_order_for_vector>)
  Where column_name_list_with_order_for_vector is:
      <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
   (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
   VECTOR_L2_OPS])
  ```

  pgvector를 사용하여 데이터를 인덱싱하는 방법에 대한 자세한 내용은 [Indexing](#)을 참조하세요.

- 성능

  - T-SQL 엔드포인트에서 쿼리 계획을 디버깅하는 데 `SET BABELFISH_STATISTICS PROFILE ON`을 사용합니다.

- T-SQL에서 지원되는 `set_config` 함수를 사용하여 `max_parallel_workers_get_gather`를 늘립니다.
- 대략적인 검색에 `IVFFlat`을 사용합니다. 자세한 내용은 IVFFlat을 참조하세요.

pgvector를 사용하여 성능을 개선하는 방법에 대한 자세한 내용은 Performance를 참조하세요.

제한 사항

- Babelfish는 하이브리드 검색에 전체 텍스트 검색을 지원하지 않습니다. 자세한 내용은 Hybrid Search를 참조하세요.
- Babelfish는 현재 재인덱싱 기능을 지원하지 않습니다. 그러나 여전히 PostgreSQL 엔드포인트를 사용하여 재인덱싱이 가능합니다. 자세한 내용은 Vacuuming을 참조하세요.

Babelfish와 함께 Amazon Aurora 기계 학습 사용

Amazon Aurora 기계 학습과 통합하면 Babelfish for Aurora PostgreSQL DB 클러스터의 기능을 확장할 수 있습니다. 이 원활한 통합을 통해 각각 고유한 기계 학습 요구 사항을 해결하도록 맞춤화된 Amazon Comprehend, Amazon SageMaker AI, Amazon Bedrock과 같은 다양하고 강력한 서비스에 액세스할 수 있습니다.

Babelfish 사용자는 Aurora 기계 학습을 사용할 때 T-SQL 구문 및 의미 체계에 대한 기존 지식을 사용할 수 있습니다. Aurora PostgreSQL에 대한 AWS 설명서에 나와 있는 지침을 따르세요. 자세한 내용은 Aurora PostgreSQL과 함께 Amazon Aurora 기계 학습 사용 섹션을 참조하세요.

사전 조건

- Aurora 기계 학습을 사용하도록 Babelfish for Aurora PostgreSQL DB 클러스터를 설정하기 전에 관련 요구 사항과 사전 조건을 이해해야 합니다. 자세한 내용은 Aurora PostgreSQL과 함께 Aurora 기계 학습을 사용할 때 요구 사항 섹션을 참조하세요.
- Postgres 엔드포인트 또는 `sp_execute_postgresql` 저장 프로시저를 사용하여 `aws_ml` 확장을 설치해야 합니다.

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> 현재 Babelfish는 Babelfish 내에서 `sp_execute_postgresql`을 사용한 캐스케이드 작업을 지원하지 않습니다. `aws_ml`은 `aws_commons`를 사용하기 때문에 Postgres 엔드포인트를 사용하여 별도로 설치해야 합니다.

```
create extension aws_common;
```

`aws_ml` 함수를 사용한 T-SQL 구문 및 의미 체계 처리

다음 예시는 T-SQL 구문과 의미 체계가 Amazon ML 서비스에 적용되는 방법을 설명합니다.

Example : aws_bedrock.invoke_model - Amazon Bedrock 함수를 사용하는 간단한 쿼리

```
aws_bedrock.invoke_model(
   model_id      varchar,
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

다음 예제는 invoke_model을 사용하여 Bedrock에 대한 Anthropic Claude 2 모델을 간접 호출하는 방법을 보여줍니다.

```

SELECT aws_bedrock.invoke_model (
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example : aws_comprehend.detect_sentiment - Amazon Comprehend 함수를 사용하는 간단한 쿼리

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

다음 예시는 Amazon Comprehend 서비스를 간접 호출하는 방법을 보여줍니다.

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example : aws_sagemaker.invoke_endpoint - Amazon SageMaker 함수를 사용하는 간단한 쿼리

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

model_input은 VARIADIC으로 표시되고 유형이 'any'이므로 사용자는 길이 및 데이터 유형과 관계없이 목록을 함수에 전달할 수 있으며, 이 함수는 모델에 대한 입력 역할을 합니다. 다음 예시는 Amazon SageMaker 서비스를 간접 호출하는 방법을 보여줍니다.

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

Aurora PostgreSQL과 함께 Aurora 기계 학습을 사용하는 방법에 대한 자세한 내용은 Aurora PostgreSQL과 함께 Amazon Aurora 기계 학습 사용 섹션을 참조하세요.

제한 사항

- Babelfish는 배열 생성을 허용하지 않지만 배열을 나타내는 데이터는 처리할 수 있습니다. 배열을 반환하는 `aws_bedrock.invoke_model_get_embeddings`와 같은 함수를 사용하면 결과가 배열 요소가 포함된 문자열로 전달됩니다.

연결된 서버를 지원하는 Babelfish

Babelfish for Aurora PostgreSQL은 버전 3.1.0에서 PostgreSQL `tds_fdw` 확장을 사용하여 연결된 서버를 지원합니다. 연결된 서버로 작업을 수행하려면 `tds_fdw` 확장을 설치해야 합니다. `tds_fdw`에 대한 자세한 내용은 [Amazon Aurora PostgreSQL용 지원되는 외부 데이터 래퍼 작업]을 참조하세요.

`tds_fdw` 확장 설치

다음 방법을 사용하여 `tds_fdw` 확장을 설치할 수 있습니다.

PostgreSQL 엔드포인트에서 CREATE EXTENSION 사용

1. PostgreSQL 포트에서 Babelfish 데이터베이스의 PostgreSQL DB 인스턴스에 연결합니다. rds_superuser 역할이 있는 계정을 사용합니다.

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=test --dbname=babelfish_db --password
```

2. `tds_fdw` 확장을 설치합니다. 이는 일회성 설치 프로세스입니다. DB 클러스터가 다시 시작될 때 재설치하지 않아도 됩니다.

```

babelfish_db=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

TDS 엔드포인트에서 **`sp_execute_postgresql`** 저장 프로시저 호출

Babelfish는 버전 3.3.0에서 `sp_execute_postgresql` 프로시저를 호출하여 `tds_fdw` 확장 설치를 지원합니다. T-SQL 포트를 종료하지 않고도 T-SQL 엔드포인트에서 PostgreSQL 문을 실행할 수 있습니다. 자세한 내용은 [Babelfish for Aurora PostgreSQL 작업 절차] 단원을 참조하세요.

1. T-SQL 포트에서 Babelfish 데이터베이스의 PostgreSQL DB 인스턴스에 연결합니다.

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. `tds_fdw` 확장을 설치합니다.

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

지원되는 기능

Babelfish는 원격 RDS for SQL Server 또는 Babelfish for Aurora PostgreSQL 엔드포인트를 연결된 서버로 추가하는 것을 지원합니다. 다른 원격 SQL Server 인스턴스를 연결된 서버로 추가할 수도 있습니다. 그런 다음, `OPENQUERY()`를 사용하여 이러한 연결된 서버에서 데이터를 검색합니다. Babelfish 버전 3.2.0부터 네 부분으로 구성된 이름도 지원됩니다.

연결된 서버를 사용하기 위해 다음과 같은 저장 프로시저 및 카탈로그 보기가 지원됩니다.

저장 프로시저

- sp_addlinkedserver – Babelfish는 `@provstr` 파라미터를 지원하지 않습니다.
- sp_addlinkedsrvlogin
  - 원격 데이터 소스에 연결하려면 명시적인 원격 사용자 이름과 암호를 입력해야 합니다. 사용자의 자체 보안 인증 정보로는 연결할 수 없습니다. Babelfish는 `@useself = false`만 지원합니다.
  - 로컬 로그인과 관련된 원격 서버 액세스 구성은 지원되지 않으므로, Babelfish는 `@locallogin` 파라미터를 지원하지 않습니다.
- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – 로컬 로그인과 관련된 원격 서버 액세스 구성은 지원되지 않으므로, Babelfish는 `@locallogin` 파라미터를 지원하지 않습니다.
- sp_serveroption - Babelfish는 다음과 같은 서버 옵션을 지원합니다.
  - 쿼리 제한 시간(Babelfish 버전 3.2.0 이상)
  - 연결 제한 시간(Babelfish 버전 3.3.0 이상)
- sp_testlinkedserver(Babelfish 버전 3.3.0 이상)
- sp_enum_oledb_providers(Babelfish 버전 3.3.0 이상)

카탈로그 보기

- sys.servers
- sys.linked_logins

연결에 전송 중 암호화 사용

소스 Babelfish for Aurora PostgreSQL에서 대상 원격 서버로의 연결은 원격 서버 데이터베이스 구성에 따라 전송 중 암호화(TLS/SSL)를 사용합니다. 원격 서버에 암호화가 구성되지 않은 경우 원격 데이터베이스에 대한 요청을 수행하는 Babelfish 서버는 암호화되지 않은 상태로 폴백됩니다.

연결 암호화를 적용하려면

- 대상 연결 서버가 RDS for SQL Server 인스턴스인 경우 대상 SQL Server 인스턴스에 대해 `rds.force_ssl = on`으로 설정합니다. RDS for SQL Server에 대한 SSL/TLS 구성과 관련된 자세한 내용은 Microsoft SQL Server DB 인스턴스와 함께 SSL 사용을 참조하세요.
- 대상 연결 서버가 Babelfish for Aurora PostgreSQL 클러스터인 경우 대상 서버에 대해 `babelfishpg_tds.tds_ssl_encrypt = on` 및 `ssl = on`으로 설정합니다. SSL/TLS 사용에 관한 자세한 내용은 Babelfish SSL 설정 및 클라이언트 연결 섹션을 참조하세요.

SQL Server에서 Babelfish를 연결된 서버로 추가

Babelfish for Aurora PostgreSQL은 SQL Server에서 연결된 서버로 추가할 수 있습니다. SQL Server 데이터베이스에서 ODBC용 Microsoft OLE DB 공급자인 MSDASQL을 사용하여 Babelfish를 연결된 서버로 추가할 수 있습니다.

MSDASQL 공급자를 사용하여 SQL Server에서 Babelfish를 연결된 서버로 구성하는 방법은 2가지가 있습니다.

- ODBC 연결 문자열을 공급자 문자열로 제공합니다.
- 연결된 서버를 추가하는 동안 ODBC 데이터 소스의 시스템 DSN을 제공합니다.

제한 사항

- OPENQUERY()는 SELECT에서만 사용할 수 있고 DML에서는 사용할 수 없습니다.
- 네 부분으로 구성된 객체 이름은 읽기에만 사용할 수 있고 원격 테이블 수정에는 사용할 수 없습니다. UPDATE는 수정하지 않고도 FROM 절의 원격 테이블을 참조할 수 있습니다.
- Babelfish 연결 서버에 대한 저장 프로시저 실행은 지원되지 않습니다.

- OPENQUERY()에 종속된 객체가 있거나 네 부분으로 구성된 이름을 통해 참조되는 객체가 있는 경우 Babelfish 메이저 버전 업그레이드가 작동하지 않을 수 있습니다. 메이저 버전 업그레이드 전에 OPENQUERY() 또는 네 부분으로 구성된 이름을 참조하는 모든 객체가 삭제되었는지 확인해야 합니다.
- 다음 데이터 유형은 원격 Babelfish 서버(nvarchar(max), varchar(max), varbinary(max), binary(max), time)에서 예상대로 작동하지 않습니다. CAST 함수를 사용하여 지원되는 데이터 유형으로 변환하는 것이 좋습니다.

예

다음은 클라우드에서 RDS for SQL Server의 인스턴스에 Babelfish for Aurora PostgreSQL 인스턴스가 연결된 예시입니다.

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

연결된 서버가 설치되면 T-SQL OPENQUERY() 또는 네 부분으로 구성된 표준 이름 지정을 사용하여 다음의 원격 서버 기반 테이블, 뷰 또는 기타 지원되는 객체를 참조할 수 있습니다.

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

연결된 서버 및 모든 관련 로그인을 중단하는 방법:

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

문제 해결

소스 서버와 원격 서버 모두에 동일한 보안 그룹을 사용하여 서로 통신하도록 허용할 수 있습니다. 보안 그룹은 TDS 포트(기본값 1433)의 인바운드 트래픽만 허용해야 하며 보안 그룹의 소스 IP를 보안 그룹 ID 자체로 설정할 수 있습니다. 보안 그룹이 동일한 다른 인스턴스에서 인스턴스에 연결하는 규칙을

설정하는 방법에 대한 자세한 내용은 [보안 그룹이 동일한 인스턴스에서 인스턴스에 대한 연결 규칙]을 참조하세요.

액세스 권한이 올바르게 구성되지 않은 경우 원격 서버를 쿼리하려고 할 때 다음 예시와 비슷한 오류 메시지가 나타납니다.

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

Babelfish에서 전체 텍스트 검색 사용

버전 4.0.0부터 Babelfish는 전체 텍스트 검색(FTS)에 대한 제한된 지원을 제공합니다. FTS는 텍스트가 많은 데이터를 효율적으로 검색하고 인덱싱할 수 있는 관계형 데이터베이스의 강력한 기능입니다. 이를 통해 복잡한 텍스트 검색을 수행하고 관련 결과를 빠르게 찾을 수 있습니다. FTS는 콘텐츠 관리 시스템, 전자 상거래 플랫폼, 문서 아카이브와 같이 대량의 텍스트 데이터를 처리하는 애플리케이션에 특히 유용합니다.

Babelfish 전체 텍스트 검색에서 지원하는 기능 이해

Babelfish는 다음과 같은 전체 텍스트 검색 기능을 지원합니다.

- CONTAINS 절:
  - CONTAINS 절에 대한 기본적인 지원.

```
CONTAINS (
     {
        column_name
     }
     , '<contains_search_condition>'
)
```

> **Note**
>
> 현재는 영어만 지원됩니다.

  - `simple_term` 검색 문자열의 포괄적인 처리 및 번역.
- `FULLTEXT INDEX` 절:

    - `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name` 문만 지원합니다.
    - 전체 `DROP FULLTEXT INDEX` 문을 지원합니다.

    > **Note**
    > 전체 텍스트 인덱스를 다시 인덱싱하려면 전체 텍스트 인덱스를 삭제하고 동일한 열에 새 인덱스를 생성해야 합니다.

- 검색 조건의 특수 문자:
    - Babelfish는 검색 문자열에서 특수 문자가 한 번 발생하는 경우를 효과적으로 처리할 수 있도록 합니다.

    > **Note**
    > Babelfish는 이제 검색 문자열에서 특수 문자를 식별하지만, 얻은 결과는 T-SQL로 얻은 결과와 다를 수 있다는 점을 알아둬야 합니다.

- column_name의 테이블 별칭:
    - 테이블 별칭 지원을 통해 사용자는 전체 텍스트 검색을 위해 더 간결하고 읽기 쉬운 SQL 쿼리를 만들 수 있습니다.

Babelfish 전체 텍스트 검색의 제한 사항

- 현재 Babelfish에서는 `CONTAINS` 절에 대해 다음 옵션을 지원하지 않습니다.
    - 영어 이외의 언어 및 특수 문자는 지원되지 않습니다. 지원되지 않는 문자 및 언어를 사용하는 경우 일반 오류 메시지가 표시됩니다.

    ```
    Full-text search conditions with special characters or languages other than English
     are not currently supported in Babelfish
    ```

    - `column_list`와 같은 여러 열
    - PROPERTY 속성
    - `prefix_term`, `generation_term`, `generic_proximity_term`, `custom_proximity_term` 및 `weighted_term`

- Boolean 연산자는 지원되지 않으며, 사용할 경우 다음과 같은 오류 메시지가 표시됩니다.

```
boolean operators not supported
```

- 점이 있는 식별자 이름은 지원되지 않습니다.

- 현재 Babelfish에서는 `CREATE FULLTEXT INDEX` 절에 대해 다음 옵션을 지원하지 않습니다.
  - [TYPE COLUMN type_column_name]
  - [LANGUAGE language_term]
  - [STATISTICAL_SEMANTICS]
  - catalog filegroup 옵션
  - with 옵션
- 전체 텍스트 카탈로그 생성은 지원되지 않습니다. 전체 텍스트 인덱스를 생성할 때는 전체 텍스트 카탈로그가 필요하지 않습니다.
- `CREATE FULLTEXT INDEX`는 점이 있는 식별자 이름을 지원하지 않습니다.
- Babelfish는 현재 검색 문자열에서 연속된 특수 문자를 지원하지 않습니다. 사용할 경우 다음과 같은 오류 메시지가 표시됩니다.

```
Consecutive special characters in the full-text search condition are not currently
 supported in Babelfish
```

Babelfish에서 지리공간 데이터 유형 지원

버전 3.5.0 및 4.1.0부터 Babelfish는 다음 두 가지 공간 데이터 유형에 대한 지원을 포함합니다.

- 기하학 데이터 유형 - 이 데이터 유형은 평면, 즉 유클리드(평평한 지구) 데이터를 저장하기 위한 것입니다.
- 지리학 데이터 유형 - 이 데이터 유형은 GPS 위도 및 경도 좌표와 같은 타원체, 즉 둥근 지구 데이터를 저장하는 데 사용됩니다.

이러한 데이터 유형을 사용하면 공간 데이터를 저장하고 조작할 수 있지만 제한이 있습니다.

Babelfish의 지리공간 데이터 유형 이해

- 지리공간 데이터 유형은 뷰, 프로시저 및 테이블과 같은 다양한 데이터베이스 객체에서 지원됩니다.

- 2차원 포인트 데이터 유형을 지원하여 위치 데이터를 위도, 경도 및 유효한 공간 참조 시스템 식별자(SRID)로 정의된 포인트로 저장할 수 있습니다.
- JDBC, ODBC, DOTNET, PYTHON과 같은 드라이버를 통해 Babelfish에 연결하는 애플리케이션은 이 지리공간 기능을 활용할 수 있습니다.

Babelfish에서 지원하는 기하학 데이터 유형 함수

- STGeomFromText(***geometry_tagged_text***, SRID) - WKT(Well-Known Text) 표현을 사용하여 기하학 인스턴스를 만듭니다.
- STPointFromText(***point_tagged_text***, SRID) - WKT 표현을 사용하여 포인트 인스턴스를 생성합니다.
- Point(X, Y, SRID) - x 및 y 좌표의 부동 소수점 값을 사용하여 포인트 인스턴스를 만듭니다.
- <geometry_instance>.STAsText ( ) - 기하학 인스턴스에서 WKT 표현을 추출합니다.
- <geometry_instance>.STDistance(other_geometry) - 두 기하학 인스턴스 사이의 거리를 계산합니다.
- <geometry_instance>.STX - 기하학 인스턴스의 X 좌표(경도)를 추출합니다.
- <geometry_instance>.STY - 기하학 인스턴스의 Y 좌표(위도)를 추출합니다.

Babelfish에서 지원하는 지리학 데이터 유형 함수

- STGeomFromText(***geography_tagged_text***, SRID) - WKT 표현을 사용하여 지리학 인스턴스를 만듭니다.
- STPointFromText(***point_tagged_text***, SRID) - WKT 표현을 사용하여 포인트 인스턴스를 생성합니다.
- Point(Lat, Long, SRID) - 위도 및 경도의 부동 소수점 값을 사용하여 포인트 인스턴스를 만듭니다.
- geography_instance>.STAsText( ) - 지리학 인스턴스에서 WKT 표현을 추출합니다.
- geography_instance>.STDistance(other_geography) - 두 지리학 인스턴스 사이의 거리를 계산합니다.
- <geography_instance>.Lat - 지리학 인스턴스의 위도 값을 추출합니다.
- <geography_instance>.Long - 지리학 인스턴스의 경도 값을 추출합니다.

Babelfish에서 지원하는 지리공간 데이터 유형의 한계

- 현재 Babelfish는 지리공간 데이터 유형의 포인트 인스턴스에 대한 Z-M 플래그와 같은 고급 기능을 지원하지 않습니다.
- 포인트 인스턴스 이외에 다음과 같은 기하학 유형은 현재 지원되지 않습니다.
  - LineString
  - CircularString
  - CompoundCurve
  - Polygon
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection
- 현재 공간 인덱싱은 지리공간 데이터 유형에 지원되지 않습니다.
- 현재 이러한 데이터 유형에는 나열된 함수만 지원됩니다. 자세한 내용은 Babelfish에서 지원하는 기하학 데이터 유형 함수 및 Babelfish에서 지원하는 지리학 데이터 유형 함수 단원을 참조하세요.
- 지리학 데이터에 대한 STDistance 함수 출력값은 T-SQL과 비교하여 정밀도 차이가 약간 있을 수 있습니다. 이는 기본 PostGIS 구현 때문입니다. 자세한 내용은 ST_Distance를 참조하세요.
- 성능을 최적화하려면 Babelfish에서 추가 추상화 계층을 만들지 말고 기본 제공되는 지리공간 데이터 유형을 사용하세요.

> **Tip**
>
> 사용자 지정 데이터 유형을 만들 수 있지만 지리공간 데이터를 기반으로 생성하는 것은 권장되지 않습니다. 이로 인해 복잡성이 발생하고 지원이 제한적이어서 예상치 못한 동작이 발생할 수 있습니다.

- Babelfish에서는 지리공간 함수 이름이 키워드로 사용되며 의도한 방식으로 사용하는 경우에만 공간 연산을 수행합니다.

> **Tip**
>
> Babelfish에서 사용자 정의 함수와 프로시저를 만들 때는 기본 제공되는 지리공간 함수와 같은 이름을 사용하지 마세요. 이름이 같은 기존 데이터베이스 객체가 있는 경우 `sp_rename`을 사용하여 이름을 바꾸세요.

Babelfish의 파티셔닝 이해

버전 4.3.0부터 Babelfish는 제한된 지원으로 테이블 및 인덱스 파티셔닝을 도입합니다. 다음 섹션에서는 Babelfish에서 파티션 함수 생성, 파티션 체계 정의, 파티션된 테이블 및 인덱스 구현에 대한 세부 정보를 제공합니다.

주제

Babelfish 파티셔닝 소개

- 파티션 함수:
  - `CREATE PARTITION FUNCTION`: 분할 열 유형과 각 파티션의 값 범위를 지정하여 테이블 또는 인덱스를 파티셔닝하는 방법을 정의합니다.
  - `DROP PARTITION FUNCTION`: 기존 파티션 함수를 제거합니다.
- 파티셔닝 체계:
  - `CREATE PARTITION SCHEME`: 파티션과 파일 그룹 간의 매핑을 정의합니다.

    > **Note**
    >
    > Babelfish에서 파일 그룹은 더미 객체로 취급되며 물리적 스토리지 위치를 나타내지 않습니다.

  - `DROP PARTITION SCHEME`: 기존 파티션 체계를 제거합니다.
- 시스템 함수:
  - `$PARTITION`: 이 시스템 함수는 파티셔닝 열의 지정된 값이 지정된 파티셔닝된 테이블에 속하는 파티션 번호를 반환합니다.

- 파티셔닝된 테이블 및 인덱스:
  - `CREATE TABLE ... ON partition_scheme_name (partition_column_name)`: 지정된 파티션 체계 및 파티셔닝 열을 기반으로 파티셔닝된 테이블을 생성합니다.
  - `CREATE INDEX ... ON partition_scheme_name (partition_column_name)`: 지정된 파티션 체계 및 파티셔닝 열을 기반으로 파티셔닝된 인덱스를 생성합니다.
- 메타데이터 파티셔닝을 위한 시스템 뷰:

  파티셔닝과 관련된 메타데이터를 제공하기 위해 다음 시스템 뷰가 추가됩니다.
  - `sys.destination_data_spaces`
  - `sys.partitions`
  - `sys.partition_functions`
  - `sys.partition_parameters`
  - `sys.partition_range_values`
  - `sys.partition_schemes`

제한 사항 및 해결 방법

다음 SQL Server 파티셔닝 기능은 Babelfish에서 아직 지원되지 않습니다.

- `ALTER PARTITION FUNCTION` 및 `ALTER PARTITION SCHEME`.

> **Note**
>
> Babelfish는 분할 및 병합 작업을 지원하지 않습니다. 나중에 파티션을 추가하거나 제거할 수 없으므로 만드는 중에 파티션 함수의 모든 파티션을 정의합니다.

- 파티셔닝 열로 계산된 열.
- 파티셔닝된 테이블에 대한 `INSERT BULK` 및 BCP 유틸리티.
- 파티션 함수의 `LEFT` 경계 옵션.
- 파티션 함수의 `SQL_VARIANT` 데이터 유형.
- `TRUNCATE TABLE ... WITH PARTITION`.
- `ALTER TABLE ... SWITCH PARTITION`.
- 파티션 체계 및 파티션 테이블과 다른 파티션 열과 같이 정렬되지 않은 파티셔닝된 인덱스.
- Babelfish 소스에서의 DMS 마이그레이션은 파티셔닝된 테이블의 전체 로드 작업에만 지원됩니다.

- 파티션 함수의 데이터 정렬 사용.
- 기본 데이터베이스 데이터 정렬 이외의 데이터 정렬과 함께 파티셔닝 열 사용.
- Babelfish는 이러한 구문 옵션을 지원하지 않지만 다음의 해결 방법을 제공합니다.
  - CREATE TABLE 문에 제약 조건 또는 인덱스가 있는 파티션 체계를 사용합니다.
  - ALTER TABLE ... ADD CONSTRAINT ... ON partition_scheme_name (partition_column_name).

> **Note**
>
> `babelfishpg_tsql.escape_hatch_storage_on_partition` 이스케이프 해치를 무시하도록 설정할 수 있습니다. 이렇게 하면 파서가 제약 조건 또는 인덱스와 함께 사용되는 파티션 체계 옵션을 무시할 수 있으며 백엔드는 각 파티션에 대해 개별 제약 조건 또는 인덱스를 생성합니다.

Babelfish 문제 해결

다음에서 일부 Babelfish DB 클러스터 문제에 대한 문제 해결 아이디어와 해결 방법을 찾아볼 수 있습니다.

주제

연결 실패

Babelfish가 포함된 새 Aurora DB 클러스터에 공통적으로 발생하는 연결 실패의 원인은 다음과 같습니다.

- 보안 그룹에서 액세스 허용하지 않음 - Babelfish 연결에 문제가 있는 경우 기본 Amazon EC2 보안 그룹에 IP 주소를 추가했는지 확인합니다. https://checkip.amazonaws.com/을 사용하여 IP 주소를 확인한 다음, 해당 주소를 TDS 포트 및 PostgreSQL 포트의 인바운드 규칙에 추가할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하세요.
- SSL 구성 불일치 - Aurora PostgreSQL에서 `rds.force_ssl` 파라미터가 활성화되어 있으면(1로 설정) 클라이언트는 SSL을 통해 Babelfish에 연결해야 합니다. 클라이언트가 올바르게 설정되지 않은 경우 다음과 같은 오류 메시지가 표시됩니다.

```
Cannot connect to your-Babelfish-DB-cluster, 1433
----------------------
ADDITIONAL INFORMATION:
no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
"database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
...
```

  이 오류는 로컬 클라이언트와 Babelfish DB 클러스터 간에 SSL 구성 문제가 있을 수 있으며 클러스터에서 클라이언트가 SSL을 사용해야 함을 나타냅니다(`rds.force_ssl` 파라미터가 1로 설정됨). SSL 구성에 대한 자세한 내용은 Amazon RDS 사용 설명서의 PostgreSQL DB 인스턴스와 함께 SSL 사용을 참조하세요.

  SQL Server Management Studio(SSMS)를 사용하여 Babelfish에 연결할 때 이 오류가 표시되는 경우 연결 속성 창에서 연결 암호화(Encrypt connection) 및 서버 인증서 신뢰(Trust server certificate) 연결 옵션을 선택하고 다시 시도할 수 있습니다. 이러한 설정은 SSMS에 대한 SSL 연결 요구 사항을 해결합니다.

Aurora 연결 문제 해결에 대한 자세한 내용은 Amazon RDS DB 인스턴스에 연결할 수 없음 단원을 참조하세요.

Babelfish 끄기

더 이상 Babelfish가 필요 없는 경우 Babelfish 기능을 끌 수 있습니다.

일부 고려 사항을 유의하세요.

- 경우에 따라 Aurora PostgreSQL로의 마이그레이션을 완료하기 전에 Babelfish를 끌 수 있습니다. DDL이 SQL Server 데이터 형식에 의존하거나 코드에 T-SQL 기능을 사용하는 경우 해당 코드는 실패합니다.
- Babelfish 인스턴스를 프로비저닝한 후 Babelfish 확장을 끄는 경우 동일한 클러스터에서 동일한 데이터베이스를 다시 프로비저닝할 수 없습니다.

Babelfish를 끄려면 `rds.babelfish_status`를 `OFF`로 설정한 파라미터 그룹을 수정합니다. `rds.babelfish_status`를 `datatypesonly`로 설정하여 Babelfish가 꺼진 상태로 SQL Server 데이터 유형을 계속 사용할 수 있습니다.

파라미터 그룹에서 Babelfish를 끄는 경우 해당 파라미터 그룹을 사용하는 모든 클러스터에서 Babelfish 기능이 사라집니다.

파라미터 그룹 수정에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 섹션을 참조하세요. Babelfish별 파라미터에 대한 자세한 내용은 [Babelfish용 DB 클러스터 파라미터 그룹 설정](#) 섹션을 참조하세요.

Babelfish for Aurora PostgreSQL 버전 업데이트 관리

Babelfish는 Aurora PostgreSQL 버전 13.4 이상에서 사용할 수 있는 옵션입니다. Babelfish에 대한 업데이트는 Aurora PostgreSQL 데이터베이스 엔진의 특정 새 릴리스로 제공됩니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

> **Note**
>
> Aurora PostgreSQL 13의 모든 버전에서 실행 중인 Babelfish DB 클러스터는 Aurora PostgreSQL 14.3, 14.4 및 14.5로 업그레이드할 수 없습니다. 또한 Babelfish는 13.x에서 15.x로의 직접 업그레이드를 지원하지 않습니다. 먼저 13.x DB 클러스터를 14.6 이상 버전으로 업그레이드한 다음 15.x 버전으로 업그레이드해야 합니다.

다양한 Babelfish 릴리스에서 지원되는 기능 목록은 버전별 Babelfish에서 지원되는 기능 섹션을 참조하세요.

현재 지원되지 않는 기능 목록은 Babelfish에서 지원되지 않는 기능 섹션을 참조하세요.

다음과 같이 describe-db-engine-versions AWS CLI 명령을 사용해 AWS 리전을 쿼리하여 Babelfish를 지원하는 Aurora PostgreSQL 버전 목록을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
```

자세한 정보는 AWS CLI 명령 참조의 describe-db-engine-versions 섹션을 참조하세요.

다음 주제에서는 Aurora PostgreSQL DB 클러스터에서 실행 중인 Babelfish 버전을 식별하는 방법과 새 버전으로 업그레이드하는 방법을 알아볼 수 있습니다.

목차

사용 중인 Babelfish 버전 식별

Babelfish를 쿼리하여 Babelfish 버전, Aurora PostgreSQL 버전 및 호환되는 Microsoft SQL Server 버전에 대한 세부 정보를 찾을 수 있습니다. TDS 포트 또는 PostgreSQL 포트를 사용할 수 있습니다.

TDS 포트를 사용하여 버전 정보 쿼리

1. sqlcmd 또는 ssms을 사용하여 Babelfish DB 클러스터의 엔드포인트에 연결합니다.

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    login-id -P password -d db_name
```

2. Babelfish 버전을 식별하려면 다음 쿼리를 실행합니다.

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
2> GO
```

   쿼리는 다음과 비슷한 결과를 반환합니다.

```
serverproperty
------------------------------
3.4.0

(1 rows affected)
```

3. Aurora PostgreSQL DB 클러스터의 버전을 식별하려면 다음 쿼리를 실행합니다.

```
1> SELECT aurora_version() AS aurora_version
2> GO
```

   쿼리는 다음과 비슷한 결과를 반환합니다.

```
aurora_version



-------------------------------------------------
15.5.0

(1 rows affected)
```

4. 호환되는 Microsoft SQL Server 버전을 식별하려면 다음 쿼리를 실행합니다.

```
1> SELECT @@VERSION AS version
2> GO
```

   쿼리는 다음과 비슷한 결과를 반환합니다.

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

다음 쿼리를 실행하면 Babelfish와 Microsoft SQL Server 간의 근소한 차이를 알 수 있습니다. Babelfish에서는 쿼리가 1을 반환하지만, Microsoft SQL Server에서는 NULL를 반환합니다.

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

다음 절차에서와 같이 PostgreSQL 포트를 사용하여 버전 정보를 확인할 수도 있습니다.

PostgreSQL 포트를 사용하여 버전 정보 쿼리

1. psql 또는 pgAdmin을 사용하여 Babelfish DB 클러스터의 엔드포인트에 연결합니다.

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
     port=5432 dbname=babelfish_db user=sa
```

2. psql의 확장 기능(\x)을 켜면 출력이 보다 읽기 편해집니다.

```
babelfish_db=> \x
babelfish_db=> SELECT
babelfish_db=> aurora_version() AS aurora_version,
babelfish_db=> version() AS postgresql_version,
babelfish_db=> sys.version() AS Babelfish_compatibility,
babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
```

이 쿼리는 다음과 비슷한 출력을 반환합니다.

```

-[ RECORD 1 ]-----------
+------------------------------------------------------------------------------------------------
aurora_version          | 15.5.0
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                     +
                        | Dec 7 2023 09:43:06
                                    +
                        | Copyright (c) Amazon Web Services
                                     +
                        | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

Babelfish 클러스터를 새 버전으로 업그레이드

새로운 버전의 Babelfish는 버전 13.4 이후 Aurora PostgreSQL 데이터베이스 엔진의 일부 새 릴리스로 제공됩니다. Babelfish는 새로운 릴리스마다 버전 번호가 고유합니다. Aurora PostgreSQL과 마찬가지로 Babelfish는 버전에 *major.minor.patch* 명명 체계를 사용합니다. 예를 들어 첫 번째 Babelfish 릴리스인 Babelfish 버전 1.0.0은 Aurora PostgreSQL 13.4.0의 일부로 제공되었습니다.

Babelfish는 별도의 설치 과정이 필요하지 않습니다. Babelfish for Aurora PostgreSQL DB 클러스터 생성에서 설명한 것처럼 Turn on Babelfish(Babelfish 켜기)는 Aurora PostgreSQL DB 클러스터를 생성할 때 선택하는 옵션입니다.

마찬가지로 Babelfish는 지원하는 Aurora DB 클러스터와 별도로 업그레이드할 수 없습니다. 기존 Babelfish for Aurora PostgreSQL DB 클러스터를 새 버전의 Babelfish로 업그레이드하려면 사용하려는 Babelfish 버전을 지원하는 새 버전으로 | Aurora PostgreSQL DB 클러스터를 업그레이드합니다. 업그레이드 절차는 다음과 같이 Babelfish 배포를 지원하는 Aurora PostgreSQL 버전에 따라 달라집니다.

메이저 버전 업그레이드

Aurora PostgreSQL 15.2 버전으로 업그레이드하기 전에 아래의 Aurora PostgreSQL 버전을 Aurora PostgreSQL 14.6 이상 버전으로 업그레이드해야 합니다.

- Aurora PostgreSQL 13.8 이상의 모든 버전
- Aurora PostgreSQL 13.7.1 이상의 모든 마이너 버전
- Aurora PostgreSQL 13.6.4 이상의 모든 마이너 버전

Aurora PostgreSQL 14.6 이상 버전을 Aurora PostgreSQL 15.2 이상 버전으로 업그레이드할 수 있습니다.

Aurora PostgreSQL DB 클러스터를 새 메이저 버전으로 업그레이드하려면 몇 가지 예비 작업이 필요합니다. 자세한 내용은 메이저 버전 업그레이드 수행 단원을 참조하십시오. Babelfish for Aurora PostgreSQL DB 클러스터를 성공적으로 업그레이드하려면 새 Aurora PostgreSQL 버전에 대한 사용자 지정 DB 클러스터 파라미터 그룹을 생성해야 합니다. 이 새 파라미터 그룹은 업그레이드 중인 클러스터와 Babelfish 파라미터 설정이 동일해야 합니다. 자세한 내용과 메이저 버전 업그레이드 소스 및 대상 테이블은 Babelfish를 새 메이저 버전으로 업그레이드를 참조하세요.

마이너 버전 업그레이드 및 패치

마이너 버전 및 패치는 업그레이드를 위해 새 DB 클러스터 파라미터 그룹을 생성할 필요가 없습니다. 마이너 버전 및 패치는 자동 적용이든 수동 적용이든 마이너 버전 업그레이드 절차를 따르면 됩니다. 자세한 내용과 버전 소스 및 대상 테이블은 Babelfish를 새 마이너 버전으로 업그레이드를 참조하세요.

> **Note**
>
> 메이저 또는 마이너 업그레이드를 수행하기 전에 대기 중인 모든 유지 관리 작업을 Babelfish for Aurora PostgreSQL 클러스터에 적용하세요.

주제

- Babelfish를 새 마이너 버전으로 업그레이드

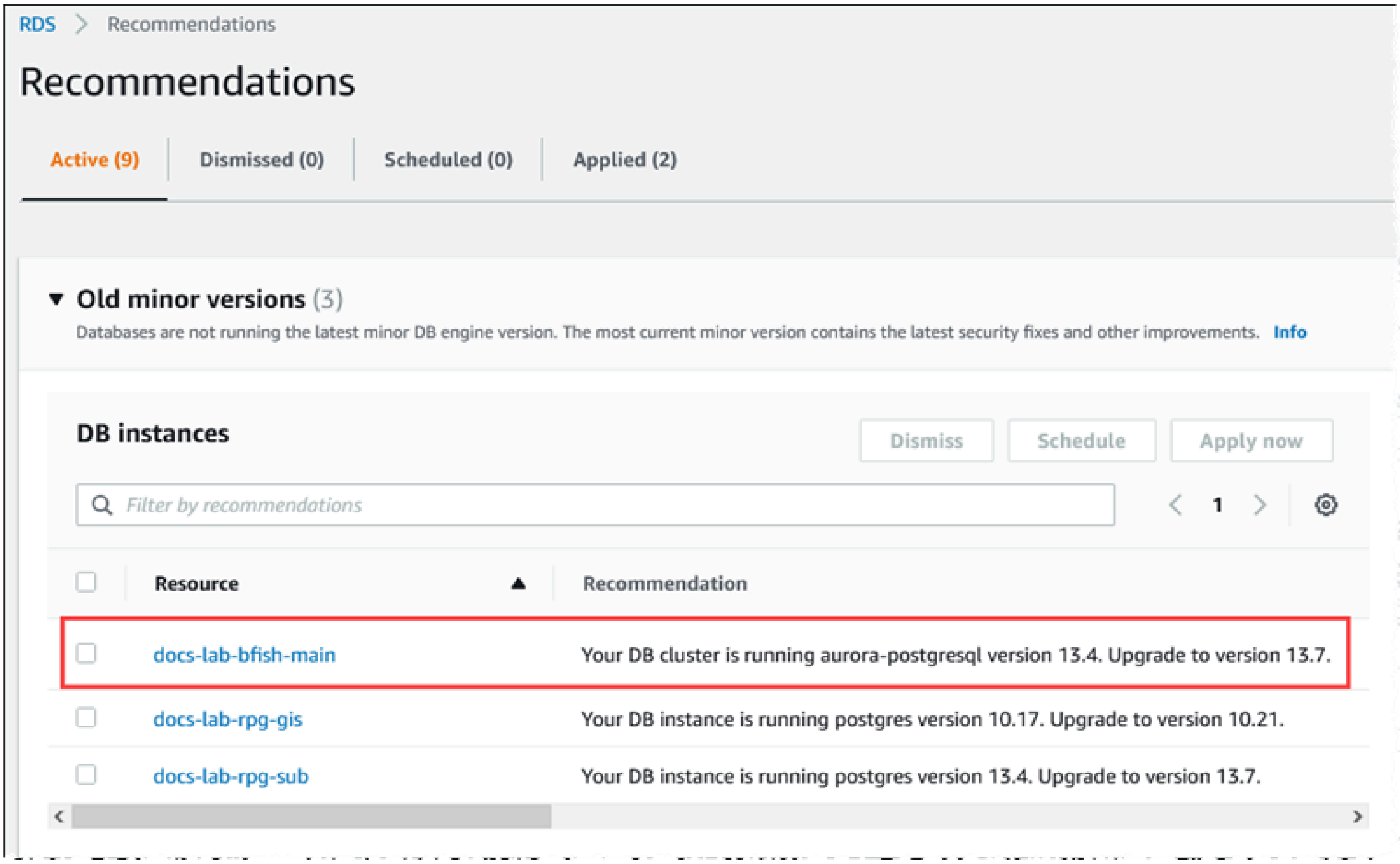
- [Babelfish를 새 메이저 버전으로 업그레이드]

Babelfish를 새 마이너 버전으로 업그레이드

마이너 버전 업데이트는 이전 버전과의 호환성을 유지하는 것을 목표로 합니다. 그러나 경우에 따라 중요한 보안 수정 또는 중요한 버그 수정은 이전 버전과 완전히 호환되지 않을 수 있습니다. 패치 버전에는 릴리스 후 마이너 버전에 대한 중요한 수정 사항이 포함되어 있습니다. 예를 들어 Aurora PostgreSQL 13.4의 첫 번째 릴리스에 대한 버전 레이블은 Aurora PostgreSQL 13.4.0이었습니다. Aurora PostgreSQL 13.4.1, 13.4.2 및 13.4.4를 포함하여 현재까지 해당 마이너 버전에 대한 여러 패치가 릴리스되었습니다. 해당 버전의 Aurora PostgreSQL 릴리스 정보 상단에 있는 Patch releases(패치 릴리스) 목록에서 각 Aurora PostgreSQL 버전에 사용 가능한 패치를 찾을 수 있습니다. 예는 Aurora PostgreSQL 릴리스 정보에서 PostgreSQL 14.3을 참조하세요.

Aurora PostgreSQL DB 클러스터가 Auto minor version upgrade(자동 마이너 버전 업그레이드) 옵션으로 구성된 경우 Babelfish for Aurora PostgreSQL DB 클러스터는 클러스터의 유지 관리 기간 동안 자동으로 업그레이드됩니다. 마이너 버전 자동 업그레이드(AmVU) 및 사용 방법에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 단원을 참조하세요. 클러스터에서 AmVU를 사용하지 않는 경우 유지 관리 작업에 응답하거나 새 버전을 사용하도록 클러스터를 수정하여 Babelfish for Aurora PostgreSQL DB 클러스터를 새 마이너 버전으로 수동 업그레이드할 수 있습니다.

설치할 Aurora PostgreSQL 버전을 선택하고 AWS Management Console에서 기존 Aurora PostgreSQL DB 클러스터를 보면 버전에 *major*.*minor* 숫자만 표시됩니다. 예를 들어 Aurora PostgreSQL 13.4가 있는 기존 Babelfish for Aurora PostgreSQL DB 클러스터에 대한 콘솔의 다음 이미지는 클러스터를 Aurora PostgreSQL의 새로운 마이너 릴리스인 버전 13.7로 업그레이드하도록 권장합니다.



수준을 포함한 전체 버전 세부 정보를 얻으려면 `aurora_version` Aurora PostgreSQL 함수를 사용하여 Aurora PostgreSQL DB 클러스터를 쿼리할 수 있습니다. 자세한 내용은 Aurora PostgreSQL 함수 참조의 aurora_version 섹션을 참조하세요. 사용 중인 Babelfish 버전 식별의 To use the PostgreSQL port to query for version information 절차에서 이 함수를 사용하는 예를 찾을 수 있습니다.

다음 표에는 Aurora PostgreSQL 및 Babelfish 버전과 마이너 버전 업그레이드 절차를 지원할 수 있는 사용 가능한 대상 버전이 나와 있습니다.

현재 소스 버전	최신 업그레이드 대상
Aurora PostgreSQL(Babelfish)	Aurora PostgreSQL(Babelfish)
17.4(5.1)	17.5(5.2)
16.8(4.5)	16.9(4.6)
16.6(4.4.0)	16.9(4.6), 16.8(4.5.0)

현재 소스 버전	최신 업그레이드 대상
16.4(4.3.0)	16.9(4.6), 16.8(4.5.0), 16.6(4.4.0)
16.3(4.2.0)	16.9(4.6), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0)
16.2(4.1.0)	16.9(4.6), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0)
16.1(4.0.0)	16.9(4.6), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0)
15.12(3.9.0)	15.13(3.10)
15.10(3.8.0)	15.13(3.10), 15.12(3.9.0)
15.8(3.7.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0)
15.7(3.6.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0)
15.6(3.5.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0)
15.5(3.4.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0)
15.4(3.3.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0)
15.12(3.9.0), 15.3(3.2.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0)
15.2(3.1.0)	15.13(3.10), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0), 15.3(3.2.0)
14.17(2.12.0)	14.18(2.13.0)

현재 소스 버전	최신 업그레이드 대상
14.15(2.11.0)	14.18(2.13.0), 14.17(2.12.0)
14.13(2.10.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0)
14.12(2.9.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0)
14.11(2.8.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0)
14.10(2.7.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0)
14.9(2.6.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0), 14.10(2.7.0)
14.8(2.5.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0), 14.10(2.7.0), 14.9(2.6.0)
14.7(2.4.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0), 14.10(2.7.0), 14.9(2.6.0), 14.8(2.5.0)
14.6(2.3.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0), 14.10(2.7.0), 14.9(2.6.0), 14.8(2.5.0), 14.7(2.4.0)
14.5(2.2.0)	14.18(2.13.0), 14.17(2.12.0), 14.15(2.11.0), 14.13(2.10.0), 14.12(2.9.0), 14.11(2.8.0), 14.10(2.7.0), 14.9(2.6.0), 14.8(2.5.0), 14.7(2.4.0), 14.6(2.3.0)
14.3(2.1.0)	14.18(2.13.0), 14.6(2.3.0)

현재 소스 버전	최신 업그레이드 대상
13.8(1.4.0)	13.9(1.5.0)
13.7(1.3.0)	13.9(1.5.0), 13.8(1.4.0)

Babelfish를 새 메이저 버전으로 업그레이드

메이저 버전 업그레이드에서는 먼저 Babelfish for Aurora PostgreSQL DB 클러스터를 메이저 버전 업그레이드를 지원하는 버전으로 업그레이드해야 합니다. 이를 위해서 패치 업데이트 또는 마이너 버전 업그레이드를 DB 클러스터에 적용하세요. 자세한 내용은 Babelfish를 새 마이너 버전으로 업그레이드를 참조하세요.

다음 테이블에는 메이저 버전 업그레이드를 지원할 수 있는 Aurora PostgreSQL 버전과 Babelfish 버전이 나와 있습니다.

현재 소스 버전	최신 업그레이드 대상
Aurora PostgreSQL(Babelfish)	Aurora PostgreSQL(Babelfish)
16.9(4.6.0)	17.5(5.2.0)
16.8(4.5.0)	17.5(5.2.0), 17.4(5.1.0)
16.6(4.4.0)	17.5(5.2.0), 17.4(5.1.0)
16.4(4.3.0)	17.5(5.2.0), 17.4(5.1.0)
16.3(4.2.0)	17.5(5.2.0), 17.4(5.1.0)
16.2(4.1.0)	17.5(5.2.0), 17.4(5.1.0)
16.1(4.0.0)	17.5(5.2.0), 17.4(5.1.0)
15.13(3.10)	17.5(5.2.0) 16.9(4.6.0)
15.12(3.9.0)	17.5(5.2.0), 17.4(5.1.0)

현재 소스 버전	최신 업그레이드 대상
	16.9(4.6.0), 16.8(4.5.0)
15.10(3.8.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0)
15.8(3.7.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0)
15.7(3.6.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0)
15.6(3.5.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0)
15.5(3.4.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0)
15.4(3.3.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0)
15.3(3.2.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0)
15.2(3.1.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0)

현재 소스 버전	최신 업그레이드 대상
14.18(2.13.0)	17.5(5.2.0) 16.9(4.6.0) 15.13(3.10.0)
14.17(2.12.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0) 15.13(3.10.0), 15.12(3.9.0)
14.15(2.11.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0)
14.13(2.10.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0)
14.12(2.9.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0)
14.11(2.8.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0)

현재 소스 버전	최신 업그레이드 대상
14.10(2.7.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0)
14.9(2.6.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0)
14.8(2.5.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0), 15.3(3.2.0)
14.7(2.4.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0), 15.3(3.2.0), 15.2(3.1.0)

현재 소스 버전	최신 업그레이드 대상
14.6(2.3.0)	17.5(5.2.0), 17.4(5.1.0) 16.9(4.6.0), 16.8(4.5.0), 16.6(4.4.0), 16.4(4.3.0), 16.3(4.2.0), 16.2(4.1.0), 16.1(4.0.0) 15.13(3.10.0), 15.12(3.9.0), 15.10(3.8.0), 15.8(3.7.0), 15.7(3.6.0), 15.6(3.5.0), 15.5(3.4.0), 15.4(3.3.0), 15.3(3.2.0), 15.2(3.1.0)
13.9(1.5.0)	14.6(2.3.0)
13.8(1.4.0)	14.6(2.3.0)
13.7(1.3.0)	14.6(2.3.0)

Babelfish를 새 메이저 버전으로 업그레이드하기 전

업그레이드하는 과정에서 일시적인 중단이 발생할 수 있습니다. 따라서 유지 관리 기간 동안 또는 사용량이 많지 않은 기간에 업그레이드를 수행하거나 예약하는 것이 좋습니다.

메이저 버전 업그레이드를 수행하기 전

1. 사용 중인 Babelfish 버전 식별에 나와 있는 명령을 사용하여 기존 Aurora PostgreSQL DB 클러스터의 Babelfish 버전을 식별합니다. Aurora PostgreSQL 버전 및 Babelfish 버전 정보는 PostgreSQL에서 처리하므로 To use the PostgreSQL port to query for version information 절차에 설명된 단계에 따라 세부 정보를 가져옵니다.
2. 사용 중인 버전이 메이저 버전 업그레이드를 지원하는지 확인합니다. 메이저 버전 업그레이드 기능을 지원하는 버전 목록은 Babelfish를 새 마이너 버전으로 업그레이드를 참조하여 필요한 사전 업그레이드 작업을 수행하세요.

   예를 들어 Babelfish 버전이 Aurora PostgreSQL 13.5 DB 클러스터에서 실행 중이고 Aurora PostgreSQL 15.2로 업그레이드하려는 경우 먼저 모든 마이너 릴리스 및 패치를 적용하여 클러스터를 Aurora PostgreSQL 14.6 이상 버전으로 업그레이드하세요. 클러스터 버전이 14.6 이상이면 메이저 버전 업그레이드 절차를 계속 진행합니다.
3. 현재 Babelfish DB 클러스터의 수동 스냅샷을 백업으로 생성합니다. 백업을 통해 클러스터를 Aurora PostgreSQL 버전, Babelfish 버전으로 복원하고 모든 데이터를 업그레이드 전 상태로 복

원할 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷 생성 섹션을 참조하세요. 이 클러스터를 업그레이드 전 상태로 복원하려면 기존 사용자 지정 DB 클러스터 파라미터 그룹을 다시 사용할 수 있도록 유지해야 합니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 및 파라미터 그룹 고려 사항(을)를 참조하세요.

4. 대상 Aurora PostgreSQL DB 버전에 대한 사용자 지정 DB 클러스터 파라미터 그룹을 준비합니다. 현재 Babelfish for Aurora PostgreSQL DB 클러스터에서 Babelfish 파라미터 설정을 복제합니다. 모든 Babelfish 파라미터 목록을 보려면 Babelfish용 DB 클러스터 파라미터 그룹 설정을 참조하세요. 메이저 버전 업그레이드에서 다음 파라미터에 소스 DB 클러스터와 동일한 설정이 필요합니다. 업그레이드에 성공하려면 모든 설정이 동일해야 합니다.

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql.default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

   > **Warning**
   >
   > 새 Aurora PostgreSQL 버전에 대한 사용자 지정 DB 클러스터 파라미터 그룹의 Babelfish 파라미터 설정이 업그레이드 중인 클러스터의 파라미터 값과 일치하지 않으면 `ModifyDBCluster` 작업이 실패합니다. AWS Management Console 또는 `modify-db-cluster` AWS CLI 명령에서 나온 출력값에 `InvalidParameterCombination` 오류 메시지가 나타납니다.

5. AWS Management Console 또는 AWS CLI에서 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 업그레이드하려는 Aurora PostgreSQL 버전에 해당하는 Aurora PostgreSQL 패밀리를 선택합니다.

   > **Tip**
   >
   > 파라미터 그룹은 AWS 리전 수준에서 관리됩니다. AWS CLI로 작업할 때 명령에서 `--region`을 지정하는 대신 기본 리전으로 구성할 수 있습니다. AWS CLI 사용에 대한 자세한 내용은 AWS Command Line Interface사용 설명서의 빠른 설정을 참조하세요.

메이저 버전 업그레이드 수행

1. Aurora PostgreSQL DB 클러스터를 새 메이저 버전으로 업그레이드합니다. 자세한 내용은 Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드 섹션을 참조하세요.
2. 파라미터 설정을 적용할 수 있도록 클러스터의 라이터 인스턴스를 재부팅합니다.

새 메이저 버전으로 업그레이드한 후

새 Aurora PostgreSQL 버전으로 메이저 버전을 업그레이드한 후 `IDENTITY` 열이 있는 테이블의 `IDENTITY` 값은 업그레이드 전 값보다 클 수 있습니다(+32). 따라서 다음 행이 이러한 테이블에 삽입될 때 생성된 ID 열 값이 +32 숫자로 이동하고 거기에서 시퀀스를 시작합니다. 이 조건은 Babelfish DB 클러스터의 기능에 부정적인 영향을 미치지 않습니다. 하지만 원하는 경우 열의 최대값을 기준으로 시퀀스 객체를 재설정할 수 있습니다. 그러려면 `sqlcmd` 또는 다른 SQL Server 클라이언트를 사용하여 Babelfish 작성자 인스턴스의 T-SQL 포트에 연결합니다. 자세한 내용은 SQL Server 클라이언트 도구를 사용하여 DB 클러스터에 연결 섹션을 참조하세요.

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
      sa -P ******** -d dbname
```

연결되면 다음 SQL 명령을 사용하여 연결된 시퀀스 객체를 시드하는 데 사용할 수 있는 문을 생성합니다. 이 SQL 명령은 단일 데이터베이스 및 다중 데이터베이스 Babelfish 구성 모두에서 작동합니다. 이러한 두 가지 배포 모델에 대한 자세한 내용은 단일 데이터베이스 또는 여러 데이터베이스에서 Babelfish 사용을 참조하세요.

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

쿼리는 최대 IDENTITY 값을 재설정하고 간격을 좁히기 위해 실행할 수 있는 일련의 SELECT 문을 생성합니다. 다음은 Babelfish 클러스터에서 실행되는 샘플 SQL Server 데이터베이스인 Northwind를 사용할 때 나오는 출력값을 보여줍니다.

```
--------------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

명령문을 하나씩 실행하여 시퀀스 값을 재설정합니다.

예: Babelfish DB 클러스터를 메이저 릴리스로 업그레이드

이 예시에서는 Babelfish 버전 1.2.2를 실행하는 Aurora PostgreSQL 13.6.4 DB 클러스터를 Aurora PostgreSQL 14.6으로 업그레이드하는 방법을 설명하는 일련의 AWS CLI 명령을 확인할 수 있습니다. 먼저 Aurora PostgreSQL 14에 사용할 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 다음으로 Aurora PostgreSQL 버전 13 소스의 파라미터 값과 일치하도록 파라미터 값을 수정합니다. 마지막으로 소스 클러스터를 수정하여 업그레이드를 수행합니다. 자세한 내용은 Babelfish용 DB 클러스터 파라미터 그룹 설정 섹션을 참조하세요. 해당 주제에서는 AWS Management Console에서 업그레이드를 수행하는 방법에 대한 내용도 찾을 수 있습니다.

[create-db-cluster-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/create-db-cluster-parameter-group.html) CLI 명령을 사용하여 새 버전에 사용할 DB 클러스터 파라미터 그룹을 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

이 명령을 실행하면 사용자 지정 DB 클러스터 파라미터 그룹이 AWS 리전에 생성됩니다. 출력은 다음과 비슷합니다.

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 섹션을 참조하세요.

[modify-db-cluster-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster-parameter-group.html) CLI 명령을 사용하여 소스 클러스터와 일치하도록 설정을 수정합니다.

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
```

```
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

응답은 다음과 비슷합니다.

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

modify-db-cluster CLI 명령을 사용하여 새 버전과 새 사용자 지정 DB 클러스터 파라미터 그룹을 사용하도록 클러스터를 수정합니다. 다음 샘플과 같이 `--allow-major-version-upgrade` 인수도 지정합니다.

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

파라미터 설정을 적용할 수 있도록 reboot-db-instance CLI 명령을 사용하여 클러스터의 라이터 인스턴스를 재부팅합니다.

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

Babelfish 제품 버전 파라미터 사용

`babelfishpg_tds.product_version`이라는 새로운 Grand Unified Configuration(GUC) 파라미터가 Babelfish 2.4.0 및 3.1.0 버전에서 도입되었습니다. 이 파라미터를 사용하면 SQL Server 제품 버전 번호를 Babelfish의 출력으로 설정할 수 있습니다.

이 파라미터는 4부분으로 구성된 버전 ID 문자열이며, 각 부분은 “.”로 구분해야 합니다.

구문

```
Major.Minor.Build.Revision
```

- 메이저 버전: 11에서 16 사이의 숫자.
- 메이저 버전: 0에서 255 사이의 숫자.
- 빌드 버전: 0에서 65535 사이의 숫자.
- 개정: 0 및 양수.

Babelfish 제품 버전 파라미터 구성

클러스터 파라미터 그룹을 사용하여 콘솔에서 babelfishpg_tds.product_version 파라미터를 설정해야 합니다. DB 클러스터 파라미터 수정 방법에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

제품 버전 파라미터를 잘못된 값으로 설정하면 변경 사항이 적용되지 않습니다. 콘솔에 새 값이 표시되더라도 파라미터는 이전 값을 유지합니다. 오류 메시지의 자세한 내용을 보려면 엔진 로그 파일을 확인하세요.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

영향을 받는 쿼리 및 파라미터

쿼리/파라미터	결과	유효 시간
SELECT @@VERSION	사용자 정의 SQL 서버 버전 (babelfishpg_tsql.version value = 기본값)을 반환함	즉시
SELECT SERVERPROPERTY('ProductVersion')	사용자 정의 SQL Server 버전을 반환함	즉시
SELECT SERVERPROPERTY('ProductMajorVersion')	사용자 정의 SQL Server 버전의 메이저 버전을 반환함	즉시
PRELOGIN 응답 메시지의 VERSION 토큰	서버가 사용자 정의 SQL Server 버전과 함께 PRELOGIN 메시지를 반환함	사용자가 새 세션을 만들 때 적용됨
JDBC 사용 시 LoginAck의 SQLServerVersion	DatabaseMetaData.getDatabaseProductVersion()은 사용자 정의 SQL Server 버전을 반환함	사용자가 새 세션을 만들 때 적용됨

babelfishpg_tsql.version 파라미터가 있는 인터페이스

babelfishpg_tsql.version 및 babelfishpg_tds.product_version 파라미터를 사용하여 @@VERSION의 출력을 설정할 수 있습니다. 다음 예시는 이 두 가지 파라미터가 어떤 방식으로 상호 작용하는지 보여 줍니다.

- babelfishpg_tsql.version 파라미터가 '기본값'이고 babelfishpg_tds.product_version이 15.0.2000.8인 경우.
  - @@version의 출력 — 15.0.2000.8.
- babelfishpg_tsql.version 파라미터가 13.0.2000.8로 설정되어 있고 babelfishpg_tds.product_version 파라미터가 15.0.2000.8로 설정된 경우.
  - @@version의 출력 — 13.0.2000.8.

Babelfish for Aurora PostgreSQL 참조

주제

- 버전별 Babelfish에서 지원되는 기능
- Babelfish에서 지원되지 않는 기능
- Babelfish for Aurora PostgreSQL 작업 절차

버전별 Babelfish에서 지원되는 기능

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 모든 기능 목록을 찾아볼 수 있습니다. 지원되지 않는 기능 목록은 Babelfish에서 지원되지 않는 기능 단원을 참조하세요. 다양한 Babelfish 릴리스에 대한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

Aurora 및 PostgreSQL 기능

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 Aurora 및 PostgreSQL 기능을 찾아볼 수 있습니다.

Aurora 및 PostgreSQL 기능	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
Aurora ML 서비스	예	예	예	예	예	예	예	예	아니요	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
AWS Directory Service를 사용하여 Kerberos로 데이터베이스 인증	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요
덤프 및 복원	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요

Aurora 및 PostgreSQL 기능	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
pg_stat_statement 확장으로 데이터베이스를 이동하는 pg_dump/pg_restore 유틸리티	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	아니요
pgvector	0	0	0	0	0	0	0	0	C ㄴ ς	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	아니요
제로 가동 중지 패치 적용(ZDP)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	아니요

T-SQL 기능 또는 구문

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 T-SQL 기능 또는 구문을 찾아볼 수 있습니다.

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
SELECT 문에 대해 4부분으로 구성된 객체 이름 참조	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	아니요
데이터베이스 소유자를 변경하기 위한 ALTER AUTHORIZATION 구문	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
																	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
ALTER DATABASE *<db_name>* MODIFY NAME = *<new_db_name>*	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
ALTER FUNCTION	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
ALTER PROCEDURE	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
역할 변경	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ALTER USER...WITH LOGIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
ALTER VIEW	0	C L ʕ	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
CREATE FUNCTION의 AS 키워드	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
AT TIME ZONE 절	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
연결된 서버로서의 Babelfish 인스턴스	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
비교 연산자 !< and !>	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
CREATE INDEX ... ON partition_scheme_name (partition_column_name)	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
SQL Server 뷰에서 트리거 대신 생성(DML)	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
CREATE PARTITION FUNCTION	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
CREATE PARTITION SCHEME	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
역할 생성	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CREATE TABLE ... ON partition_scheme_name (partition_column_name)	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
CREATE TRIGGER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CREATE OR ALTER VIEW	0	C L ς	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
고유 인덱스 생성	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
데이터베이스 간 프로시저 실행	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
데이터베이스 간 참조 SELECT, INSERT, UPDATE, DELETE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
커서로 입력된 파라미터는 입력 파라미터에만 해당(출력은 아님)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
bcp 클라이언트 유틸리티를 사용한 데이터 마이그레이션	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
TIMESTAMP, ROWVERSION 데이터 유형(사용 정보는 구현이 제한된 기능 참조)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
저장 프로시저 및 함수 직접 호출 시 DEFAULT 키워드	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DBCC CHECKIDENT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
DROP DATABASE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	아니요
DROP IF EXISTS(SCHEMA, DATABASE, USER 객체의 경우)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
DROP INDEX 인덱스 ON schema.table	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
DROP INDEX schema.table.index	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
DROP PARTITION FUNCTION	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
DROP PARTITION SCHEME	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
DROP ROLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ENABLE/DISABLE TRIGGER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
FULL TEXT SEARCH	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
CONTAINS 절을 사용한 전체 텍스트 검색	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
지오메트리 및 지리 공간 데이터 유형	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
GRANT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
GRANT 권한 [,...n] ON SCHEMA	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
GUC babelfish pg_tds.product_version	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	아니요
선행 도트 문자가 있는 식별자	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
테이블의 INSTEAD OF 트리거	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
뷰의 INSTEAD OF 트리거	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
KILL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
$PARTITION.partition_function_name(partition_column_value)	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
PIVOT(4.4.0 및 3.8.0의 뷰 정의에서 사용, 3.4.0의 공통 테이블 표현식, 3.4.0의 조인)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
REVOKE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
REVOKE 권한 [,...n] ON SCHEMA	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
SELECT... OFFSET...FETCH 절	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
SELECT FOR JSON AUTO	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
SET BABELFISH _SHOWPLAN_ALL ON(및 OFF)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
SET BABELFISH _STATISTICS PROFILE ON(OFF)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
SET CONTEXT_INFO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요	0	0	0	0	0	0	0	아니요	아니요	아니요	아니요	아니요	아니요
SET LOCK_TIMEOUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
SET NO_BROWSETABLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
SET 행 수	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요	0	0	0	0	0	0	0	아니요	아니요	아니요	아니요	아니요	아니요
SET SHOWPLAN_ALL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
SET STATISTICS IO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
SET TRANSACTION ISOLATION LEVEL 구문	예	예	예	예	예	예	예	예	아니요	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
SSMS: 객체 탐색기 연결 대화 상자를 사용하여 연결	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
SSMS: 가져오기/내보내기 마법사를 통한 데이터 마이그레이션	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
SSMS: 객체 탐색기에 대한 일부 지원	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
STDEV	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
STDEVP	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
여러 DML 작업이 있는 트리거는 전환 테이블 참조 가능	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
T-SQL 힌트(조인 메서드, 인덱스 사용, MAXDOP)	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요
LIKE 조건자를 사용한 T-SQL 대괄호 구문	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
UNPIVOT	예	아니요	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
저장 프로시저 호출의 따옴표 없는 문자열 값 및 기본값	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요

T-SQL 기능 또는 구문	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
VAR	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
VARP	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요

T-SQL 기본 제공 함수

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 T-SQL 기본 제공 함수를 찾아볼 수 있습니다.

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
APP_NAME	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요
ATN2	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요
CHARINDEX	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
CHOOSE																																

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
COL_LENGTH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
COL_NAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
COLUMNS_UPDATED	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
COLUMNPROPERTY(CharMaxLen, AllowsNull 만)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CONCAT_WS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CONTEXT_INFO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	아니요
CURSOR_STATUS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
DATABASE_PRINCIPAL_ID	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	ㅇ ㄴ 오	ㅇ ㄴ 오	ㅇ ㄴ 오	ㅇ ㄴ 오	아니요
DATEADD	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	아니요
DATEDIFF	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	아니요
DATEDIFF_BIG	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	ㅇ ㄴ 오	아니요
DATEFROMPARTS	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
DATENAME	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ 오	아니요

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DATEPART	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요
DATETIMEFROMPARTS	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
DATETIME2FROMPARTS	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
DATETIMEOFFSETFROMPARTS	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요
DATETRUNC	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요
DATE_BUCKET	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
EOMONTH	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요
EXECUTE AS CALLER	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요
fn_listextendedproperty	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
FOR JSON	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요
FULLTEXTSERVICEPROPERTY	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
HAS_DBACCESS	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
HAS_PERMS_BY_NAME	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
HOST_NAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
HOST_ID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
IDENTITY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
IS_MEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IS_ROLEMEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IS_SRVROL EMEMBER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ISJSON	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
JSON_MODIFY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ 오	아니요
JSON_QUERY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
JSON_VALUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
NEXT VALUE FOR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ 오	C ㄴ 오	아니요
OBJECT_DEFINITION	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ 오	C ㄴ 오	아니요
OBJECT_SCHEMA_NAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ 오	C ㄴ 오	아니요
OPENJSON	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
OPENQUERY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ORIGINAL_LOGIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
PARSENAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	아니요
PATINDEX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ROWCOUNT_BIG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	아니요
SCHEMA_NAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
SESSION_CONTEXT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	아니요
SESSION_USER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
SID_BINARY(항상 NULL 반환)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
SMALLDATETIMEFROMPARTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	아니요
SQUARE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
STR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	아니요
STRING_AGG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	0	0	0	0	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	C ㄴ ʕ	아니요
STRING_SPLIT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
SUSER_SID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
SUSER_SNAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
SWITCHOFFSET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
SYSTEM_USER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
TIMEFROMPARTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
TODATETIMEOFFSET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
TO_CHAR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	아니요
TRIGGER_NESTLEVEL(인수 제외 시에만)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 기본 제공 함수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
TRY_CONVERT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	아니요
TYPE_ID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
TYPE_NAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
UPDATE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL INFORMATION_SCHEMA 카탈로그

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 SQL INFORMATION_SCHEMA 카탈로그를 찾아볼 수 있습니다.

T-SQL INFORMATION_SCHEMA 카탈로그	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
CHECK_CONSTRAINTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요

T-SQL INFORMATION_SCHEMA 카탈로그	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	21.0
COLUMN_DOMAIN_USAGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
COLUMNS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CONSTRAINT_COLUMN_USAGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
DOMAINS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
KEY_COLUMN_USAGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
ROUTINES	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
TABLES	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
TABLE_CONSTRAINTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL INFORMATION_SCHEMA 카탈로그	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
VIEWS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 정의 @@ 변수

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 T-SQL 시스템 정의 @@ 변수를 찾아볼 수 있습니다.

T-SQL 시스템 정의 @@ 변수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
@@CURSOR_ROWS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@DATEFIRST	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@DBTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@ERROR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@ERROR=213	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@FETCH_STATUS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
@@IDENTITY																																	

T-SQL 시스템 정의 @@ 변수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@LANGUAGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@LOCK_TIMEOUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@MAX_CONNECTIONS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@MAX_PRECISION	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@MICROSOFTVERSION	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@NESTLEVEL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@PROCID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@ROWCOUNT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
@@SERVERNAME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예

T-SQL 시스템 정의 @@ 변수	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
@@SERVICENAME	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
@@SPID	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
@@TRANCOUNT	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예
@@VERSION ([Babelfish에서의 T-SQL 차이점](#)에 설명된 형식 차이에 유의)	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예

T-SQL 시스템 저장 프로시저

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 T-SQL 시스템 저장 프로시저를 찾아볼 수 있습니다.

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sp_addextendedproperty	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
sp_addlinkedserver	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sp_addlinkedsrvlogin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	0 ㄴ 요	아니요
sp_addrole	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	아니요
sp_addrolemember	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	아니요
sp_babelfish_volatility	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	0 ㄴ 요	아니요
sp_column_privileges	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_columns	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_columns_100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_column s_managed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursor_list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursorclose	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursorexecute	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursorfetch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursoropen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursoroption	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursorprepare	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_cursorprepexec	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_cursorunprepare	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_databases	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_datatype_info	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_datatype_info_100	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_describe_cursor	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_describe_first_result_set	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_describe_undeclared_parameters	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_dropextendedproperty	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	아니요
sp_droplinkedsrvlogin	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ ㄴ ㅇ	ㅇ ㄴ ㅇ	아니요

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sp_droprole	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	아니요
sp_droprolemember	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	아니요
sp_dropserver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	아니요
sp_enum_o ledb_providers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	아니요
sp_execute	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_execute_postgre sql(CREATE, ALTER, DROP)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	0	0	0	0	0	0	0	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	C ㄴ ς	아니요
sp_executesql	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sp_fkeys	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_getapplock	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_helpdb	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_helpdbfixedrole	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇㄴㅇ	아니요
sp_helplinkedsrvlogin	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇㄴㅇ	아니요
sp_helprole	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_helprolemember	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	예
sp_helpsrvrolemember	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	ㅇ	아니요

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sp_helpuser	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_linkedservers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ᄋ	C L ᄋ	아니요
sp_oledb_ro_usrname	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_pkeys	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_prefix	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
sp_prepare	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_procedure_params_100_managed	0	0	0	0	0	0	0	0	C L ᄋ	0	0	0	0	0	0	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	C L ᄋ	아니요
sp_releaseapplock	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	21.0
sp_rename	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요
sp_renamedb	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
sp_rename(object_type 'DATABASE'의 경우)	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
sp_reset_connection	예	예	예	예	예	아니요	아니요	아니요	아니요	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요	아니요
sp_serveroption(connect_timeout 옵션)	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	예	예	예	예	예	예	예	아니요	아니요	아니요	아니요	아니요	아니요
sp_set_session_context	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	예	아니요	아니요	아니요

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_special_columns	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_sproc_columns	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_sproc_columns_100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_statistics	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_statistics_100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_stored_procedures	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_table_privileges	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_tablecollations_100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_tables	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

T-SQL 시스템 저장 프로시저	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_testlinkedserver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
sp_unprepare	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sp_updateextendedproperty	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
sp_who	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
xp_qv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

CONNECTIONPROPERTY 시스템 함수에서 지원되는 T-SQL 속성

다음 테이블에서는 다양한 Babelfish 버전에서 사용 가능한 CONNECTIONPROPERTY 시스템 함수에서 지원되는 T-SQL 속성을 찾아볼 수 있습니다.

CONNECTIONPROPERTY 시스템 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2.0	1.0
auth_scheme	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
client_net_address	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
local_net_address	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
local_tcp_port	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
net_transport	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
protocol_type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
physical_net_transport	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

OBJECTPROPERTY 시스템 함수에서 지원되는 T-SQL 속성

다음 테이블에서는 다양한 Babelfish 버전에서 사용 가능한 OBJECTPROPERTY 시스템 함수에서 지원되는 T-SQL 속성을 찾아볼 수 있습니다.

OBJECTPROPERTY 시스템 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
IsInlineFunction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsScalarFunction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsTableFunction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

SERVERPROPERTY 함수에서 지원되는 T-SQL 속성

다음 테이블에서는 다양한 Babelfish 버전에서 사용 가능한 SERVERPROPERTY 함수에서 지원되는 T-SQL 속성을 찾아볼 수 있습니다.

SERVERPROPERTY 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
Babelfish	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
콜레이션	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
CollationID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
Edition																																	

SERVERPROPERTY 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
EditionID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
EngineEdition	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
InstanceName	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요
IsAdvancedAnalyticsInstalled	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsBigDataCluster	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsFullTextInstalled	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsIntegratedSecurityOnly	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsLocalDB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

SERVERPROPERTY 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
IsPolyBaseInstalled	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsSingleUser	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
IsXTPSupported	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
Japanese_CI_AI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
Japanese_CI_AS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
Japanese_CS_AS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
LicenseType	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
MachineName	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	아니요

SERVERPROPERTY 함수에서 지원되는 T-SQL 속성	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	201.0
ProductLevel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	0 ㄴ 요	아니요
ProductMajorVersion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ProductMinorVersion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ProductUpdateLevel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	0 ㄴ 요	아니요
ProductVersion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
ServerName	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

Babelfish에서 지원하는 SQL Server 뷰

다음 테이블에서는 다양한 Babelfish 버전에서 지원되는 SQL Server 뷰를 찾아볼 수 있습니다.

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2 1.0
information_schema.key_column_usage	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	C L ς	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
information_schema.routines	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	아니요
information_schema.schemata	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	아니요
information_schema.sequences	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	C L ς	아니요
sys.all_columns	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.all_objects	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.all_parameters	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ς	아니요

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	21.0
sys.all_sql_modules	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.all_views	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.columns	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.configurations	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.data_spaces	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.database_files	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.database_mirroring	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.database_principals	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.database_role_members	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예
sys.databases	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0예

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sys.destination_data_spaces	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
sys.dm_exec_connections	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.dm_exec_sessions	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.dm_hadr_database_replica_states	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.dm_os_host_info	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.dm_os_sys_info	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요
sys.endpoints	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.extended_properties	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L ʕ	C L ʕ	0	0	0	0	0	0	0	C L ʕ	C L ʕ	C L ʕ	C L ʕ	C L ʕ	아니요

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sys.indexes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.partitions	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
sys.partition_functions	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
sys.partition_parameters	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
sys.partition_range_values	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
sys.partition_schemes	0	0	0	0	0	0	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	C L ς	아니요
sys.schemas	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	201.0
sys.server_principals	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.server_role_members	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	C L S	아니요
sys.sql_modules	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.sysconfigures	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.syscurconfigs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.syslogins	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	아니요
sys.sysprocesses	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.sysusers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C L S	0	0	0	0	0	0	0	C L S	C L S	C L S	C L S	C L S	아니요

Babelfish에서 지원하는 SQL Server 뷰	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2.1.0
sys.table_types	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.tables	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sys.types	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	0 ㄴ 요	아니요
sys.xml_schema_collections	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
syslanguages	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	예
sysobjects.crdate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ㄴ 요	아니요

Babelfish에서 지원되지 않는 기능

다음 표와 목록에서 현재 Babelfish에서 지원되지 않는 기능을 확인할 수 있습니다. Babelfish의 업데이트는 Aurora PostgreSQL 버전에 포함되어 있습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

주제

- 현재 지원되지 않는 기능
- 지원되지 않는 설정

- [지원되지 않는 명령]
- [지원되지 않는 열 이름 또는 속성]
- [지원되지 않는 데이터 형식]
- [지원되지 않는 객체 유형]
- [지원되지 않는 함수]
- [지원되지 않는 구문]

현재 지원되지 않는 기능

이 표에서는 현재 지원되지 않는 특정 기능에 대한 정보를 확인할 수 있습니다.

기능 또는 구문	설명
어셈블리 모듈 및 SQL Common Language Runtime(CLR) 루틴	어셈블리 모듈 및 CLR 루틴과 관련된 기능은 지원되지 않습니다.
열 속성	ROWGUIDCOL, SPARSE, FILESTREAM 및 MASKED는 지원되지 않습니다.
포함된 데이터베이스	서버 수준이 아닌 데이터베이스 수준에서 인증된 로그인이 포함된 데이터베이스는 지원되지 않습니다.
데이터베이스 간 DDL	여러 데이터베이스의 객체를 참조하거나 조작하는 DDL 문을 수행하는 것은 아직 지원되지 않습니다.
커서(업데이트 가능)	업데이트 가능한 커서는 지원되지 않습니다.
커서(전역적)	GLOBAL 커서는 지원되지 않습니다.
커서(가져오기 동작)	FETCH PRIOR, FIRST, LAST, ABSOLUTE, RELATIVE와 같은 커서 가져오기 동작은 지원되지 않습니다.
커서 유형 출력 파라미터	커서 유형 변수 및 파라미터는 출력 파라미터는 지원되지 않습니다 (오류 발생).

기능 또는 구문	설명
커서 옵션	SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING, FOR UPDATE
데이터 암호화	데이터 암호화는 지원되지 않습니다.
데이터 계층 애플리케이션(DAC)	DAC 패키지(.dacpac) 또는 DAC 백업(.bacpac) 파일을 사용한 데이터 계층 애플리케이션(DAC) 가져오기 또는 내보내기 작업은 지원되지 않습니다.
DBCC 명령	Microsoft SQL Server 데이터베이스 콘솔 명령(DBCC)은 지원되지 않습니다. DBCC CHECKIDENT는 Babelfish 3.4.0 이상 릴리스에서 지원됩니다.
DROP IF EXISTS	이 구문은 USER 및 SCHEMA 객체에는 지원되지 않습니다. TABLE, VIEW, PROCEDURE, FUNCTION 및 DATABASE 객체에 대해 지원됩니다.
암호화	기본 제공 함수 및 명령문은 암호화를 지원하지 않습니다.
ENCRYPT_CLIENT_CERT 연결	클라이언트 인증서 연결은 지원되지 않습니다.
EXECUTE AS 문	이 명령문은 지원되지 않습니다.
EXECUTE AS SELF 절	이 절은 함수, 프로시저 또는 트리거에는 지원되지 않습니다.
EXECUTE AS USER 절	이 절은 함수, 프로시저 또는 트리거에는 지원되지 않습니다.
데이터베이스 이름을 참조하는 외래 키 제약 조건	데이터베이스 이름을 참조하는 외래 키 제약 조건은 지원되지 않습니다.
FORMAT	사용자 정의 유형은 지원되지 않습니다.
파라미터가 100개 이상인 함수 선언	100개 이상의 파라미터를 포함하는 함수 선언은 지원되지 않습니다.

기능 또는 구문	설명
DEFAULT를 파라미터 값으로 포함하는 함수 호출	DEFAULT는 함수 호출을 지원하는 파라미터 값이 아닙니다. Babelfish 3.4.0 이상 릴리스에서는 함수 호출을 위한 파라미터 값으로 DEFAULT가 지원됩니다.
함수, 외부 정의	SQL CLR 함수를 포함한 외부 함수는 지원되지 않습니다.
전역 임시 테이블(이름이 ##으로 시작하는 테이블)	전역 임시 테이블은 지원되지 않습니다.
그래프 기능	모든 SQL 그래프 기능은 지원되지 않습니다.
일반 확장 저장 프로시저	다양한 유지 관리 작업을 위해 SQL Server 인스턴스에서 외부 프로그램으로 인터페이스를 제공하는 시스템 저장 프로시저는 지원되지 않습니다. 여기에는 `xp_cmdshell` 및 기타 시스템 저장 프로시저가 포함됩니다. 자세한 내용은 [일반 확장 저장 프로시저](#)를 참조하세요.
여러 개의 선행 @ 문자가 있는 식별자(변수 또는 파라미터)	둘 이상의 선행 `@`으로 시작하는 식별자는 지원되지 않습니다.
@ 또는]] 문자를 포함하는 식별자, 테이블 또는 열 이름	`@` 기호 또는 대괄호를 포함하는 테이블 또는 열 이름은 지원되지 않습니다.
인라인 인덱스	인라인 인덱스는 지원되지 않습니다.
이름이 변수인 프로시저 호출	변수를 프로시저 이름으로 사용하는 것은 지원되지 않습니다.
구체화된 뷰	구체화된 뷰는 지원되지 않습니다.
NOT FOR REPLICATION 절	이 구문은 허용되고 무시됩니다.
ODBC 이스케이프 함수	ODBC 이스케이프 함수는 지원되지 않습니다.
DEFAULT를 파라미터 값으로 포함하는 프로시저 호출	DEFAULT는 지원되는 파라미터 값이 아닙니다. Babelfish 3.4.0 이상 릴리스에서는 함수 호출을 위한 파라미터 값으로 DEFAULT가 지원됩니다.

기능 또는 구문	설명
파라미터가 100개 이상인 프로시저 선언	100개 이상의 파라미터를 포함하는 선언은 지원되지 않습니다.
프로시저, 외부에서 정의됨	SQL CLR 프로시저를 포함한 외부에서 정의된 프로시저는 지원되지 않습니다.
프로시저 버전 관리	프로시저 버전 관리는 지원되지 않습니다.
WITH RECOMPILE 프로시저	WITH RECOMPILE(DECLARE 및 EXECUTE 문과 함께 사용되는 경우)은 지원되지 않습니다.
원격 객체 참조	Babelfish 연결 서버에 대한 저장 프로시저 실행은 지원되지 않습니다. 네 부분으로 구성된 객체 이름은 읽기에만 사용할 수 있고 원격 표 수정에는 사용할 수 없습니다. UPDATE는 수정하지 않고도 FROM 절의 원격 테이블을 참조할 수 있습니다. 자세한 내용은 연결된 서버를 지원하는 Babelfish 섹션을 참조하세요.
행 수준 보안	CREATE SECURITY POLICY 및 인라인 테이블 값 함수를 사용한 행 수준 보안은 지원되지 않습니다.
서비스 브로커 기능	서비스 브로커 기능은 지원되지 않습니다.
SESSIONPROPERTY	지원되지 않는 속성: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS_NULL, NUMERIC_ROUNDABORT
SET LANGUAGE	이 구문은 `english` 또는 `us_english` 이외의 값을 통해서는 지원되지 않습니다.
SP_CONFIGURE	이 시스템 저장 프로시저는 지원되지 않습니다.
SQL 키워드 SPARSE	키워드 SPARSE는 승인되고 무시됩니다.
테이블 값 생성자 구문(FROM 절)	지원되지 않는 구문은 FROM 절로 구성된 파생 테이블에 대한 구문입니다.
임시 테이블	임시 테이블은 지원되지 않습니다.

기능 또는 구문	설명
임시 프로시저는 자동으로 삭제되지 않습니다.	이 기능은 지원되지 않습니다.
트리거, 외부 정의	SQL 공용 언어 런타임(CLR)을 포함하는 이런 트리거는 지원되지 않습니다.
SCHEMABINDING 절 제외	SCHEMABINDING 없이 뷰를 생성하는 것은 지원되지 않지만 WITH SCHEMABINDING이 지정된 것처럼 뷰가 생성됩니다. 함수, 프로시저, 트리거를 생성할 때 SCHEMABINDING을 사용하면 자동으로 무시됩니다.

지원되지 않는 설정

다음 설정은 지원됩니다.

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF
- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF
- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON(명령이 예상대로 작동하지 않을 경우)
- SET FMTONLY ON(명령이 예상대로 작동하지 않습니다. 다른 문이 아닌 SELECT 문의 실행만 억제합니다.)

지원되지 않는 명령

다음 명령에 대한 특정 기능은 지원되지 않습니다.

- ADD SIGNATURE
- ALTER DATABASE, ALTER DATABASE SET

- 데이터베이스/로그 백업/복원
- BACPAC 및 DACPAC 파일 복원
- CREATE, ALTER, DROP AUTHORIZATION. ALTER AUTHORIZATION은 데이터베이스 객체에 대해 지원됩니다.
- CREATE, ALTER, DROP AVAILABILITY GROUP
- CREATE, ALTER, DROP BROKER PRIORITY
- CREATE, ALTER, DROP COLUMN ENCRYPTION KEY
- CREATE, ALTER, DROP DATABASE ENCRYPTION KEY
- CREATE, ALTER, DROP, BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

지원되지 않는 열 이름 또는 속성

다음 열 이름은 지원되지 않습니다.

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

지원되지 않는 데이터 형식

다음 데이터 형식은 지원됩니다.

- HIERARCHYID

지원되지 않는 객체 유형

다음 객체 유형은 지원되지 않습니다.

- COLUMN MASTER KEY
- CREATE, ALTER EXTERNAL DATA SOURCE
- CREATE, ALTER, DROP DATABASE AUDIT SPECIFICATION
- CREATE, ALTER, DROP EXTERNAL LIBRARY

- CREATE, ALTER, DROP SERVER AUDIT
- CREATE, ALTER, DROP SERVER AUDIT SPECIFICATION
- CREATE, ALTER, DROP, OPEN/CLOSE SYMMETRIC KEY
- CREATE, DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- 인덱싱된 뷰
- SERVICE MASTER KEY
- SYNONYM

지원되지 않는 함수

다음 내장 함수는 지원되지 않습니다.

집계 함수

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- WITHIN GROUP 절을 사용하는 STRING_AGG

암호화 함수

- CERTENCODED 함수
- CERTID 함수
- CERTPROPERTY 함수

메타데이터 함수

- COLUMNPROPERTY
- TYPEPROPERTY
- SERVERPROPERTY 함수 - 다음 속성은 지원되지 않습니다.
  - BuildClrVersion

- ComparisonStyle
- ComputerNamePhysicalNetBIOS
- HadrManagerStatus
- InstanceDefaultDataPath
- InstanceDefaultLogPath
- IsClustered
- IsHadrEnabled
- LCID
- NumLicenses
- ProcessID
- ProductBuild
- ProductBuildType
- ProductUpdateReference
- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

보안 함수

- CERTPRIVATEKEY
- LOGINPROPERTY

명령문, 연산자, 기타 함수

- EVENTDATA 기능

- GET_TRANSMISSION_STATUS
- OPENXML

지원되지 않는 구문

다음 구문은 지원되지 않습니다.

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER INDEX
- ALTER PARTITION FUNCTION
- ALTER PARTITION SCHEME
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER SERVICE, BACKUP/RESTORE SERVICE MASTER KEY 절
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE, ALTER, DROP APPLICATION ROLE
- CREATE, ALTER, DROP ASSEMBLY
- CREATE, ALTER, DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE, ALTER, DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE

- CREATE, ALTER, DROP EXTERNAL RESOURCE POOL
- CREATE, ALTER, DROP FULLTEXT CATALOG
- CREATE, ALTER, DROP FULLTEXT INDEX
- CREATE, ALTER, DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE, ALTER, DROP, OPEN/CLOSE, BACKUP/RESTORE MASTER KEY
- CREATE, ALTER, DROP QUEUE
- CREATE, ALTER, DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE, ALTER, DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE, ALTER, DROP WORKLOAD GROUP
- ALTER TRIGGER
- CREATE TABLE... GRANT 절
- CREATE TABLE... IDENTY 절
- CREATE USER - 이 구문은 지원되지 않습니다. PostgreSQL 문 CREATE USER는 SQL Server CREATE USER 구문과 동일한 사용자를 만들지 않습니다. 자세한 내용은 Babelfish에서의 T-SQL 차이점 섹션을 참조하세요.
- DENY
- END, MOVE CONVERSATION
- EXECUTE with AS LOGIN or AT option
- GET CONVERSATION GROUP

- GROUP BY ALL clause
- GROUP BY CUBE clause
- GROUP BY ROLLUP clause
- INSERT... DEFAULT VALUES
- MERGE
- READTEXT
- REVERT
- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SELECT... FOR XML PATH
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- SET STATISTICS TIME
- SET STATISTICS XML
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT

- Using EXECUTE to call a SQL function
- VIEW... CHECK OPTION clause
- VIEW... VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- WITH XMLNAMESPACES construct
- WRITETEXT
- XPATH expressions

Babelfish for Aurora PostgreSQL 작업 절차

개요

더 나은 쿼리 성능을 위해 Babelfish for Aurora PostgreSQL을 실행하는 Amazon RDS DB 인스턴스에 다음 프로시저를 사용할 수 있습니다.

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

PostgreSQL 함수 변동성은 최적화 프로그램이 특정 절의 일부에 사용될 때 쿼리 성능에 큰 영향을 미치는 더 나은 쿼리 실행을 할 수 있도록 지원합니다.

구문

```
sp_babelfish_volatility 'function_name', 'volatility'
```

인수

function_name(선택 사항)

두 부분으로 구성된 이름으로 이 인수의 값을 `schema_name.function_name`으로 지정하거나 `function_name`으로만 지정할 수 있습니다. `function_name`으로만 지정하는 경우 스키마 이름은 현재 사용자의 기본 스키마입니다.

volatility(선택 사항)

PostgreSQL의 유효한 변동성 값은 `stable`, `volatile` 또는 `immutable`입니다. 자세한 내용은 https://www.postgresql.org/docs/current/xfunc-volatility.html 단원을 참조하세요.

> **Note**
>
> `sp_babelfish_volatility`에서 여러 개의 정의를 가진 `function_name`을 호출하면 오류가 발생합니다.

결과 집합

파라미터가 언급되지 않은 경우 결과 집합이 schemaname, functionname, volatility 열 아래에 표시됩니다.

사용 노트

PostgreSQL 함수 변동성은 최적화 프로그램이 특정 절의 일부에 사용될 때 쿼리 성능에 큰 영향을 미치는 더 나은 쿼리 실행을 할 수 있도록 지원합니다.

예시

다음 예에서는 간단한 함수를 만드는 방법을 보여 주고 나중에 다른 방법을 사용하여 이러한 함수에서 sp_babelfish_volatility를 사용하는 방법을 설명합니다.

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

다음 예제는 함수의 변동성을 표시합니다.

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

다음 예제는 함수의 변동성을 변경하는 방법을 보여줍니다.

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
```

```
2> go
```

function_name만 지정하면 해당 함수의 스키마 이름, 함수 이름 및 변동성이 표시됩니다. 다음 예제는 값을 변경한 후 함수의 변동성을 표시합니다.

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

인수를 지정하지 않으면 현재 데이터베이스에 있는 함수 목록(스키마 이름, 함수 이름, 함수 변동성)이 표시됩니다.

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

T-SQL 엔드포인트에서 PostgreSQL 문을 실행할 수 있습니다. 이렇게 하면 문을 실행하기 위해 T-SQL 포트를 종료할 필요가 없으므로 애플리케이션이 간소화됩니다.

구문

```
sp_execute_postgresql [ @stmt = ] statement
```

인수

[@stmt] 문

인수는 데이터 유형 varchar입니다. 이 인수는 PG 방언 문을 허용합니다.

> **Note**
>
> 하나의 PG 방언 문만 인수로 전달할 수 있습니다. 여러 개를 전달하면 다음 오류가 발생합니다.

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```

Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

사용 노트

CREATE EXTENSION

새 확장을 만들어 현재 데이터베이스에 로드합니다.

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

다음 예시에서는 확장을 생성하는 방법을 보여줍니다.

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

다음 명령을 사용하여 확장 객체에 액세스합니다.

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> 확장 생성 시 스키마 이름을 명시적으로 제공하지 않으면 기본적으로 확장이 퍼블릭 스키마에 설치됩니다. 다음과 같이 확장 객체에 액세스하려면 스키마 한정자를 제공해야 합니다.

```
1>select * from [public].pg_stat_statements;
2>go
```

지원되는 확장

Aurora PostgreSQL에서 사용할 수 있는 다음 확장은 Babelfish와 호환됩니다.

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

제한 사항

- 확장을 설치하려면 T-SQL에서는 sysadmin 역할이, postgres에서는 rds_superuser 역할이 있어야 합니다.
- 확장은 사용자가 만든 스키마에는 설치할 수 없으며 마스터, tempdb 및 msdb 데이터베이스의 dbo 및 게스트 스키마에도 설치할 수 없습니다.
- CASCADE 옵션은 지원되지 않습니다.

ALTER EXTENSION

ALTER EXTENSION을 사용하여 새 확장 버전으로 업그레이드할 수 있습니다.

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
```

```
2>go
```

제한 사항

- ALTER EXTENSION 문을 사용해서만 확장 버전을 업그레이드할 수 있습니다. 다른 작업은 지원되지 않습니다.

DROP EXTENSION

지정된 확장을 삭제합니다. `if exists` 또는 `restrict` 옵션을 사용하여 확장을 삭제할 수도 있습니다.

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

제한 사항

- CASCADE 옵션은 지원되지 않습니다.

Amazon Aurora PostgreSQL의 성능 및 규모 조정

다음 섹션에서는 Amazon Aurora PostgreSQL DB 클러스터의 성능 및 크기 조정 관리에 대해 설명합니다. 또한 다른 유지 관리 태스크에 대한 정보도 포함됩니다.

주제

Aurora PostgreSQL DB 인스턴스 조정

Aurora PostgreSQL DB 인스턴스는 인스턴스 조정과 읽기 조정, 이렇게 두 가지 방식으로 조정할 수 있습니다. 읽기 조정에 대한 자세한 내용은 읽기 확장 단원을 참조하십시오.

DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora PostgreSQL DB 클러스터의 크기를 조정할 수 있습니다. Aurora PostgreSQL은 Aurora에 최적화된 여러 DB 인스턴스 클래스를 지원합니다. 크기가 40테라바이트(TB)보다 큰 Aurora 클러스터에는 db.t2 또는 db.t3 인스턴스 클래스를 사용하지 마세요.

> **Note**
>
> T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. T 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

크기 조정은 즉각적으로 수행되지 않습니다. 다른 DB 인스턴스 클래스에 대한 변경을 완료하는 데 15분 이상 걸릴 수 있습니다. 이 접근 방식을 사용하여 DB 인스턴스 클래스를 수정하는 경우, 사용자에게 영향을 주지 않도록 예약된 다음 유지 관리 기간 동안 수정 사항을 적용합니다.

DB 인스턴스 클래스를 직접 수정하는 대신 Amazon Aurora의 고가용성 기능을 사용하여 다운타임을 최소화할 수 있습니다. 먼저 클러스터에 Aurora 복제본을 추가합니다. 복제본을 생성할 때 클러스터에 사용할 DB 인스턴스 클래스 크기를 선택합니다. Aurora 복제본이 클러스터와 동기화되면 새로 추가된 복제본으로 장애 조치됩니다. 자세한 내용은 Aurora 복제본 및 Amazon Aurora PostgreSQL를 사용한 빠른 장애 조치 섹션을 참조하세요.

Aurora PostgreSQL에서 지원하는 DB 인스턴스 클래스의 세부 사양은 DB 인스턴스 클래스에 지원되는 DB 엔진 단원을 참조하십시오.

Aurora PostgreSQL DB 인스턴스에 대한 최대 연결

Aurora PostgreSQL DB 클러스터는 DB 인스턴스 클래스와 사용 가능한 메모리를 기준으로 리소스를 할당합니다. DB 클러스터에 대한 각 연결은 메모리 및 CPU와 같은 이러한 리소스의 증분 양을 소비합니다. 연결당 사용된 메모리는 쿼리 유형, 개수 및 임시 테이블 사용 여부에 따라 달라집니다. 유휴 연결조차도 메모리와 CPU를 소비합니다. 쿼리가 연결에서 실행될 때 각 쿼리에 대해 더 많은 메모리가 할당되고 처리가 중지되더라도 완전히 릴리스되지 않기 때문입니다. 따라서 애플리케이션이 유휴 연결을 유지하지 않도록 하는 것이 좋습니다. 이러한 각 연결은 리소스를 낭비하고 성능에 부정적인 영향을 미칩니다. 자세한 내용은 유휴 PostgreSQL 연결에 의해 소비되는 리소스를 참조하세요.

Aurora PostgreSQL DB 인스턴스에서 허용하는 최대 연결 수는 해당 DB 인스턴스의 파라미터 그룹에 지정된 max_connections 파라미터 값에 따라 결정됩니다. max_connections 파라미터에 대한 이상적인 설정은 사용하지 않는 연결을 초과하지 않고 애플리케이션에 필요한 모든 클라이언트 연결과 AWS 자동화를 지원하기 위한 추가 연결을 3개 이상 지원하는 것입니다. max_connections 파라미터 설정을 수정하기 전에 다음 사항을 고려하는 것이 좋습니다.

- max_connections 값이 너무 낮으면 클라이언트가 연결을 시도할 때 Aurora PostgreSQL DB 인스턴스에 사용 가능한 연결이 충분하지 않을 수 있습니다. 이 경우 psql을 사용하여 연결을 시도하면 다음과 같은 오류 메시지나 나타납니다.

```
psql: FATAL: remaining connection slots are reserved for non-replication superuser
 connections
```

- max_connections 값이 실제로 필요한 연결 수를 초과하면 사용하지 않는 연결로 인해 성능이 저하될 수 있습니다.

max_connections의 기본값은 다음 Aurora PostgreSQL LEAST 함수에서 나옵니다.

LEAST({DBInstanceClassMemory/9531392},5000).

max_connections의 값을 변경하려는 경우 사용자 지정 DB 클러스터 파라미터 그룹을 생성하고 거기에서 값을 변경해야 합니다. 사용자 지정 DB 파라미터 그룹을 클러스터에 적용한 후에는 새 값이 적용되도록 기본 인스턴스를 재부팅해야 합니다. 자세한 내용은 Amazon Aurora PostgreSQL parameters 및 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성(을)를 참조하세요.

> **Tip**
>
> 애플리케이션이 연결을 자주 열고 닫거나, 수명이 긴 연결을 많이 열어 두는 경우, Amazon RDS 프록시를 사용하는 것이 좋습니다. RDS 프록시는 데이터베이스 연결을 효율적으로 안전하게 공유하기 위해 연결 풀링을 사용하는 완전관리형 고가용성 데이터베이스 프록시입니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

Aurora Serverless v2 인스턴스가 이 파라미터를 처리하는 방법에 대한 자세한 내용은 Aurora Serverless v2의 최대 연결 수 섹션을 참조하세요.

Aurora PostgreSQL에 대한 임시 스토리지 한도

Aurora PostgreSQL은 Aurora 스토리지 하위 시스템에 테이블 및 인덱스를 저장합니다. Aurora PostgreSQL은 비영구 임시 파일에 대해 별도의 임시 스토리지를 사용합니다. 여기에는 쿼리 처리 중 대용량 데이터 세트를 정렬하거나 인덱스 빌드 작업을 위해 사용되는 파일이 포함됩니다. 자세한 내용은 Aurora PostgreSQL 호환 인스턴스의 로컬 스토리지 문제를 해결하려면 어떻게 해야 합니까? 문서를 참조하세요. .

이러한 로컬 스토리지 볼륨은 Amazon Elastic Block Store에서 백업하며, 더 큰 D 인스턴스 클래스를 사용하여 확장할 수 있습니다. 스토리지에 대한 자세한 내용은 Amazon Aurora 스토리지 단원을 참조하세요. 또한 NVMe 지원 인스턴스 유형과 Aurora 최적화된 읽기 지원 임시 객체를 사용하여 임시 객체를 위한 로컬 스토리지를 늘릴 수 있습니다. 자세한 내용은 Aurora 최적화된 읽기로 Aurora PostgreSQL 쿼리 성능 개선 섹션을 참조하세요.

> **Note**
>
> db.r5.2xlarge에서 db.r5.4xlarge로 변경하는 등 DB 인스턴스 크기를 조정할 때 `storage-optimization` 이벤트가 표시될 수 있습니다.

다음 표에는 각 Aurora PostgreSQL DB 인스턴스 클래스에 사용할 수 있는 임시 스토리지의 최대 용량이 나와 있습니다. Aurora에 대한 DB 인스턴스 클래스 지원의 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

DB 인스턴스 클래스	사용 가능한 최대 임시 스토리지(GiB)
db.x2g.16xlarge	1829
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67

DB 인스턴스 클래스	사용 가능한 최대 임시 스토리지(GiB)
db.r8g.48xlarge	3072
db.r8g.24xlarge	1536
db.r8g.16xlarge	998
db.r8g.12xlarge	749
db.r8g.8xlarge	499
db.r8g.4xlarge	250
db.r8g.2xlarge	125
db.r8g.xlarge	63
db.r8g.large	31
db.r7g.16xlarge	1008
db.r7g.12xlarge	756
db.r7g.8xlarge	504
db.r7g.4xlarge	252
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r7i.48xlarge	3072
db.r7i.24xlarge	1500
db.r7i.16xlarge	1008
db.r7i.12xlarge	748

DB 인스턴스 클래스	사용 가능한 최대 임시 스토리지(GiB)
db.r7i.8xlarge	504
db.r7i.4xlarge	249
db.r7i.2xlarge	124
db.r7i.xlarge	62
db.r7i.large	31
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62

DB 인스턴스 클래스	사용 가능한 최대 임시 스토리지(GiB)
db.r6i.large	31
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

> **Note**
>
> NVMe 지원 인스턴스 유형은 사용 가능한 임시 공간을 총 NVMe 크기까지 늘릴 수 있습니다. 자세한 내용은 Aurora 최적화된 읽기로 Aurora PostgreSQL 쿼리 성능 개선 섹션을 참조하세요.

Amazon Aurora에 대한 Amazon CloudWatch 지표에 설명된 `FreeLocalStorage` CloudWatch 지표를 사용하여 DB 인스턴스에 사용할 수 있는 임시 스토리지를 모니터링할 수 있습니다. (Aurora Serverless v2에는 적용되지 않습니다.)

일부 워크로드의 경우, 작업을 수행 중인 프로세스에 더 많은 메모리를 할당하여 임시 스토리지의 양을 줄일 수 있습니다. 작업에 사용할 수 있는 메모리를 증가시키려면 work_mem 또는 maintenance_work_mem PostgreSQL 파라미터의 값을 증가시킵니다.

Aurora PostgreSQL의 방대한 페이지

방대한 페이지는 DB 인스턴스가 공유 버퍼에서 사용하는 것과 같은 대규모 연속 메모리 청크로 작업할 때 오버헤드를 줄이는 메모리 관리 기능입니다. 이 PostgreSQL 기능은 현재 사용 가능한 모든 Aurora PostgreSQL 버전에서 지원됩니다.

`Huge_pages` 파라미터는 t3.medium, db.t3.large, db.t4g.medium, db.t4g.large 인스턴스 클래스를 제외한 모든 DB 인스턴스 클래스에서 기본적으로 켜져 있습니다. Aurora PostgreSQL의 지원되는 인스턴스 클래스에서는 `huge_pages` 파라미터 값을 변경하거나 이 기능을 끌 수 없습니다.

방대한 페이지 메모리 기능을 지원하지 않는 Aurora PostgreSQL DB 인스턴스에서는 해당 워크로드 변경 없이 특정 프로세스 메모리 사용량이 증가할 수 있습니다.

시스템은 서버 시작 중에 버퍼 캐시와 같은 공유 메모리 세그먼트를 할당합니다. 방대한 메모리 페이지를 사용할 수 없는 경우 시스템은 이러한 할당을 포스트마스터 프로세스에 청구하지 않습니다. 대신 공유 메모리 세그먼트의 각 4KB 페이지에 처음 액세스한 프로세스의 메모리가 포함됩니다.

> **Note**
>
> 활성 연결은 프로세스 간에 공유 메모리 사용량을 추적하는 방법에 관계없이 필요에 따라 할당된 메모리를 공유합니다.

오류 삽입 쿼리를 사용하여 Amazon Aurora PostgreSQL 테스트

오류 삽입 쿼리를 사용하여 Aurora PostgreSQL DB 클러스터의 내결함성을 테스트할 수 있습니다. 오류 삽입 쿼리는 Amazon Aurora 인스턴스에 SQL 명령으로 발행됩니다. 장애 삽입 쿼리를 사용하면 인스턴스를 중단시켜 장애 조치 및 복구를 테스트할 수 있습니다. 또한 Aurora Replica 장애, 디스크 장애 및 디스크 혼잡을 시뮬레이션할 수 있습니다. 오류 삽입 쿼리는 다음과 같이 사용 가능한 모든 Aurora PostgreSQL 버전에서 지원됩니다.

- Aurora PostgreSQL 버전 12, 13, 14 이상
- Aurora PostgreSQL 버전 11.7 이상
- Aurora PostgreSQL 버전 10.11 이상

주제

오류 삽입 쿼리가 충돌을 지정하면 Aurora PostgreSQL DB 인스턴스가 강제로 충돌합니다. 다른 오류 삽입 쿼리로 인해서도 오류 이벤트 시뮬레이션이 발생하지만 이 경우 이벤트는 발생하지 않습니다. 오류 삽입 쿼리를 제출할 때 실패 이벤트 시뮬레이션이 발생하는 시간 길이를 지정할 수도 있습니다.

Aurora 복제본의 엔드포인트에 연결하여 오류 삽입 쿼리를 Aurora 복제본 중 하나로 제출할 수 있습니다. 자세한 내용은 Amazon Aurora 엔드포인트 연결 섹션을 참조하세요.

인스턴스 충돌 테스트

`aurora_inject_crash()` 오류 삽입 쿼리 함수를 사용하여 Aurora PostgreSQL 인스턴스의 충돌을 강제로 일으킬 수 있습니다.

이 오류 삽입 쿼리의 경우 장애 조치가 발생하지 않습니다. 장애 조치를 테스트하려면 RDS 콘솔에서 DB 클러스터에 대한 장애 조치(Failover) 인스턴스 작업을 선택하거나 failover-db-cluster AWS CLI 명령 또는 FailoverDBCluster RDS API 작업을 사용합니다.

구문

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

옵션

이 오류 삽입 쿼리에는 다음 충돌 유형 중 하나가 사용됩니다. 충돌 유형은 대소문자를 구분하지 않습니다.

'인스턴스'

Amazon Aurora 인스턴스용 PostgreSQL 호환 데이터베이스의 충돌이 시뮬레이션됩니다.

'dispatcher'

Aurora DB 클러스터용 기본 인스턴스에서 디스패처 충돌이 시뮬레이션됩니다. 디스패처가 Amazon Aurora DB 클러스터용 클러스터 볼륨에 업데이트 쓰기 작업을 합니다.

'노드'

PostgreSQL 호환 데이터베이스 및 Amazon Aurora 인스턴스용 디스패처의 충돌이 모두 시뮬레이션됩니다.

Aurora 복제본 실패 테스트

`aurora_inject_replica_failure()` 오류 삽입 쿼리 함수를 사용하여 Aurora 복제본의 실패를 시뮬레이션할 수 있습니다.

Aurora 복제본이 실패하면 Aurora 복제본 또는 DB 클러스터의 모든 Aurora 복제본으로의 복제가 지정된 시간 간격 동안 지정된 백분율만큼 차단됩니다. 시간 간격이 경과되면 영향을 받는 Aurora 복제본이 자동으로 프라이머리 인스턴스와 동기화됩니다.

구문

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

percentage_of_failure

실패 이벤트 도중 차단되는 복제의 비율입니다. 이 값은 0 ~ 100의 실수(Double)입니다. 0을 지정하면 복제가 차단되지 않습니다. 100을 지정하면 모든 복제가 차단됩니다.

time_interval

Aurora Replica 실패 시뮬레이션 시간 길이. 간격은 초 단위입니다. 예를 들어 값이 20이면 시뮬레이션이 20초 동안 실행됩니다.

> **Note**
>
> Aurora 복제본 실패 이벤트에 대한 시간 간격을 지정할 때는 각별히 주의하십시오. 시간 간격을 너무 길게 지정하고 라이터 인스턴스가 실패 이벤트 중에 대량의 데이터를 기록하는 경우 Aurora DB 클러스터에서 Aurora 복제본이 충돌했다고 간주하여 해당 복제본을 교체할 수도 있습니다.

replica_name

실패 시뮬레이션을 삽입할 Aurora Replica 단일 Aurora 복제본의 실패를 시뮬레이션하려면 Aurora 복제본의 이름을 지정합니다. DB 클러스터의 모든 Aurora 복제본에 대한 오류를 시뮬레이션하려면 빈 문자열을 지정합니다.

복제본 이름을 식별하려면 `server_id` 함수의 `aurora_replica_status()` 열을 참조하십시오. 예:

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

디스크 실패 테스트

`aurora_inject_disk_failure()` 오류 삽입 쿼리 함수를 사용하여 Aurora PostgreSQL DB 클러스터에 대한 디스크 실패를 시뮬레이션할 수 있습니다.

디스크 실패 시뮬레이션 중에는 Aurora PostgreSQL DB 클러스터에서 임의로 디스크 세그먼트를 오류 상태로 표시합니다. 시뮬레이션 기간 동안 이러한 세그먼트에 대한 요청이 차단됩니다.

구문

```
SELECT aurora_inject_disk_failure(
```

```
    percentage_of_failure,
    index,
    is_disk,
    time_interval
);
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

percentage_of_failure

실패 이벤트 중에 오류 상태로 표시할 디스크의 비율(%). 이 값은 0 ~ 100의 실수(Double)입니다. 0을 지정하면 디스크의 어떤 부분도 오류 상태로 표시되지 않습니다. 100을 지정하면 전체 디스크가 오류 상태로 표시됩니다.

인덱스를 구축하고 배포할 것입니다

실패 이벤트를 시뮬레이션할 특정 논리 데이터 블록. 가용 논리 블록 또는 스토리지 노드 데이터의 범위를 초과하는 경우 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 이 오류를 방지하려면 Aurora PostgreSQL DB 클러스터를 위한 볼륨 상태 표시 단원을 참조하십시오.

is_disk

삽입 실패가 논리 블록에서 발생했는지 아니면 스토리지 노드에서 발생했는지 여부 표시. true를 지정하면 삽입 실패가 논리 블록에서 발생했음을 의미합니다. false를 지정하면 삽입 실패가 스토리지 노드에서 발생했음을 의미합니다.

time_interval

디스크 장애를 시뮬레이션하는 데 걸리는 시간입니다. 간격은 초 단위입니다. 예를 들어 값이 20이면 시뮬레이션이 20초 동안 실행됩니다.

디스크 정체 테스트

`aurora_inject_disk_congestion()` 오류 삽입 쿼리 함수를 사용하여 Aurora PostgreSQL DB 클러스터에 대한 디스크 정체를 시뮬레이션할 수 있습니다.

디스크 정체 시뮬레이션 중에는 Aurora PostgreSQL DB 클러스터에서 임의로 디스크 세그먼트를 정체 상태로 표시합니다. 시뮬레이션 기간 동안 지정된 최소 지연 시간과 최대 지연 시간 사이에서 이러한 세그먼트에 대한 요청이 지연됩니다.

구문

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

옵션

이 오류 삽입 쿼리에는 다음 파라미터가 사용됩니다.

percentage_of_failure

실패 이벤트 중에 정체 상태로 표시할 디스크의 비율(%). 이 값은 0에서 100 사이의 이중 값입니다. 0을 지정하면 디스크의 어떤 부분도 정체 상태로 표시되지 않습니다. 100을 지정하면 전체 디스크가 정체 상태로 표시됩니다.

인덱스를 구축하고 배포할 것입니다

실패 이벤트 시뮬레이션에 사용할 특정 논리 데이터 블록 또는 스토리지 노드.

가용 논리 블록 또는 스토리지 노드 데이터의 범위를 초과하는 경우 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 이 오류를 방지하려면 Aurora PostgreSQL DB 클러스터를 위한 볼륨 상태 표시 단원을 참조하십시오.

is_disk

삽입 실패가 논리 블록에서 발생했는지 아니면 스토리지 노드에서 발생했는지 여부 표시. true를 지정하면 삽입 실패가 논리 블록에서 발생했음을 의미합니다. false를 지정하면 삽입 실패가 스토리지 노드에서 발생했음을 의미합니다.

time_interval

디스크 정체를 시뮬레이션하는 데 걸리는 시간입니다. 간격은 초 단위입니다. 예를 들어 값이 20이면 시뮬레이션이 20초 동안 실행됩니다.

최소, 최대

최대 및 최소 정체 지연 시간(밀리초). 유효한 값의 범위는 0.0에서 100.0밀리초까지입니다. 정체 상태로 표시된 디스크 세그먼트는 시뮬레이션 기간 동안 최소 및 최대 범위 내에서 임의의 시간 동안 지연됩니다. 최대값은 최소값보다 커야 합니다.

Aurora PostgreSQL DB 클러스터를 위한 볼륨 상태 표시

Amazon Aurora에서 DB 클러스터 볼륨은 논리 블록의 모음으로 구성됩니다. 이 각각은 할당된 스토리지의 10기가바이트를 나타냅니다. 이러한 블록을 보호 그룹이라고 합니다.

각 보호 그룹의 데이터는 스토리지 노드라고 하는 6개의 물리 스토리지 장치에 두루 복제됩니다. 이러한 스토리지 노드는 DB 클러스터가 상주하는 리전의 3개 가용 영역(AZ)에 할당됩니다. 또한 각 스토리지 노드에는 DB 클러스터 볼륨에 대해 1개 이상의 논리 데이터 블록이 포함됩니다. 보호 그룹 및 스토리지 노드에 대한 자세한 내용은 AWS 데이터베이스 블로그의 Aurora 스토리지 엔진 소개를 참조하세요. 일반적인 Aurora 클러스터 볼륨에 대해 자세히 알아보려면 Amazon Aurora 스토리지 섹션에서 참조하세요.

`aurora_show_volume_status()` 함수를 사용하여 다음 서버 상태 변수를 반환합니다.

- `Disks` — DB 클러스터 볼륨에 대한 데이터의 총 논리 블록 수
- `Nodes` - DB 클러스터 볼륨의 총 스토리지 노드 수입니다.

`aurora_show_volume_status()` 함수를 사용하면 `aurora_inject_disk_failure()` 오류 삽입 기능을 사용할 때 오류를 방지할 수 있습니다. `aurora_inject_disk_failure()` 오류 삽입 기능은 전체 스토리지 노드 또는 스토리지 노드 내에 있는 단일 논리 데이터 블록의 실패를 시뮬레이션합니다. 함수에서 특정 논리 데이터 블록 또는 스토리지 노드의 인덱스 값을 지정합니다. 그러나 DB 클러스터 볼륨이 사용하는 논리 데이터 블록 또는 스토리지 노드의 수보다 큰 인덱스 값을 지정하면 이 명령문은 오류를 반환합니다. 오류 삽입 쿼리에 대한 자세한 내용은 오류 삽입 쿼리를 사용하여 Amazon Aurora PostgreSQL 테스트 단원을 참조하십시오.

> **Note**
>
> `aurora_show_volume_status()` 함수는 Aurora PostgreSQL 버전 10.11에서 사용할 수 있습니다. Aurora PostgreSQL 버전에 대한 자세한 내용은 Amazon Aurora PostgreSQL 릴리스 및 엔진 버전 단원을 참조하십시오.

구문

```
SELECT * FROM aurora_show_volume_status();
```

예제

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

stats_temp_directory에 대한 RAM 디스크 지정

Aurora PostgreSQL 파라미터 `rds.pg_stat_ramdisk_size`을(를) 사용하여 PostgreSQL `stats_temp_directory` 저장용 RAM 디스크에 할당되는 시스템 메모리를 지정합니다. RAM 디스크 파라미터는 Aurora PostgreSQL 14 이하 버전에서만 사용할 수 있습니다.

특정 워크로드에서 이 파라미터를 설정하면 성능이 향상되고 IO 요구 사항이 감소될 수 있습니다. `stats_temp_directory`에 대한 자세한 내용은 PostgreSQL 설명서에서 [런타임 통계](https://www.postgresql.org/docs/current/static/runtime-config-statistics.html#GUC-STATS-TEMP-DIRECTORY)를 참조하세요. PostgreSQL 버전 15부터 PostgreSQL 커뮤니티는 동적 공유 메모리를 사용하도록 전환했습니다. 따라서 `stats_temp_directory`를 설정할 필요가 없습니다.

`stats_temp_directory`에 대한 RAM 디스크를 활성화하려면 DB 클러스터에 사용되는 DB 클러스터 파라미터 그룹에서 `rds.pg_stat_ramdisk_size` 파라미터를 0이 아닌 값으로 설정합니다. 이 파라미터는 MB를 나타내므로 정수 값을 사용해야 합니다. 표현식, 공식 및 함수는 `rds.pg_stat_ramdisk_size` 파라미터에 유효하지 않습니다. 변경 사항을 적용하려면 DB 클러스터를 재시작해야 합니다. 파라미터 설정에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹을 참조하세요. DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터 또는 Amazon Aurora DB 인스턴스 재부팅을 참조하세요.

예를 들어 다음 AWS CLI 명령은 RAM 디스크 파라미터를 256MB로 설정합니다.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

DB 클러스터 재시작 후 다음 명령을 실행하여 `stats_temp_directory`의 상태를 확인합니다.

```
postgres=> SHOW stats_temp_directory;
```

명령은 다음을 반환합니다.

```
stats_temp_directory
---------------------------
/rdsdbramdisk/pg_stat_tmp
```

```
(1 row)
```

PostgreSQL을 사용한 임시 파일 관리

PostgreSQL에서 복잡한 쿼리는 여러 가지 정렬 또는 해시 연산을 동시에 수행할 수 있으며, 각 연산은 인스턴스 메모리를 사용하여 work_mem 파라미터에 지정된 값까지 결과를 저장합니다. 인스턴스 메모리가 충분하지 않은 경우 결과를 저장하기 위한 임시 파일이 생성됩니다. 이러한 임시 파일은 쿼리 실행을 완료하기 위해 디스크에 기록됩니다. 나중에 이러한 파일은 쿼리가 완료된 후 자동으로 제거됩니다. Aurora PostgreSQL에서 이러한 파일은 로컬 스토리지를 다른 로그 파일과 함께 공유합니다. FreeLocalStorage용 Amazon CloudWatch 지표를 관찰하면 Aurora PostgreSQL DB 클러스터의 로컬 스토리지 공간을 모니터링할 수 있습니다. 자세한 내용은 로컬 스토리지 문제 해결을 참조하세요.

임시 파일 사용량을 늘리는 다중 동시 쿼리가 포함된 워크로드에서는 Aurora 최적화된 읽기 클러스터를 사용하는 것이 좋습니다. 이러한 클러스터는 SSD(Solid State Drive) 블록 스토리지에 기반한 NVMe(Non-Volatile Memory Express)를 사용하여 임시 파일을 배치할 수 있습니다. 자세한 내용은 Aurora 최적화된 읽기로 Aurora PostgreSQL 쿼리 성능 개선 단원을 참조하십시오.

다음과 같은 파라미터와 함수를 사용하여 인스턴스의 임시 파일을 관리할 수 있습니다.

- **temp_file_limit** - 이 파라미터는 temp_files 크기(KB)를 초과하는 모든 쿼리를 취소합니다. 이러한 한도는 쿼리가 무한으로 실행되어 임시 파일이 디스크 공간을 사용하는 현상을 방지합니다. log_temp_files 파라미터의 결과를 사용하여 값을 추정할 수 있습니다. 가장 좋은 방법은 워크로드 동작을 검사하고 추정치에 따라 한도를 설정하는 것입니다. 다음 예는 한도를 초과했을 때 쿼리가 취소되는 방식을 보여줍니다.

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```

ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** - 이 파라미터는 세션의 임시 파일이 제거될 경우 postgresql.log로 메시지를 보냅니다. 이 파라미터는 쿼리가 성공적으로 완료된 후 로그를 생성합니다. 따라서 활성 상태의 장기 실행 쿼리의 문제를 해결하는 데에는 도움이 되지 않을 수도 있습니다.

  다음 예시에서는 쿼리가 성공적으로 완료되었을 때 임시 파일이 정리되는 동안 항목이 postgresql.log 파일에 기록되는 것을 보여줍니다.

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** - RDS for PostgreSQL 13 이상에서 제공되는 이 함수를 사용하면 현재 임시 파일 사용량을 확인할 수 있습니다. 완료된 쿼리는 함수 결과에 나타나지 않습니다. 다음 예제에서는 이 함수의 결과를 볼 수 있습니다.

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name        |    size    |      modification
------------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
----------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

파일 이름에는 임시 파일을 생성한 세션의 처리 ID(PID)가 포함됩니다. 다음 예와 같은 고급 쿼리는 각 PID에 대한 임시 파일의 합계를 수행합니다.

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** - pg_stat_statements 파라미터를 활성화하면 호출당 평균 임시 파일 사용량을 볼 수 있습니다. 쿼리의 query_id를 식별하고 이를 사용하여 다음 예와 같이 임시 파일 사용량을 검사할 수 있습니다.

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+----------------------------
```

```
 -7170349228837045701 | select a.aid from pgbench |    50 |              239226 |
                        388678
(1 row)
```

- **Performance Insights** - 성능 개선 도우미 대시보드에서 temp_bytes 및 temp_files 메트릭을 켜서 임시 파일 사용량을 확인할 수 있습니다. 그런 다음, 이 두 지표의 평균을 확인하고 두 지표가 쿼리 워크로드에 어떻게 상응하는지 확인할 수 있습니다. 성능 개선 도우미 내부의 보기에는 임시 파일을 생성 중인 쿼리가 구체적으로 표시되지 않습니다. 그러나 성능 개선 도우미를 `pg_ls_tmpdir`에 대해 표시된 쿼리와 결합하면 쿼리 워크로드의 문제를 해결하고, 분석하고, 변경 사항을 확인할 수 있습니다.

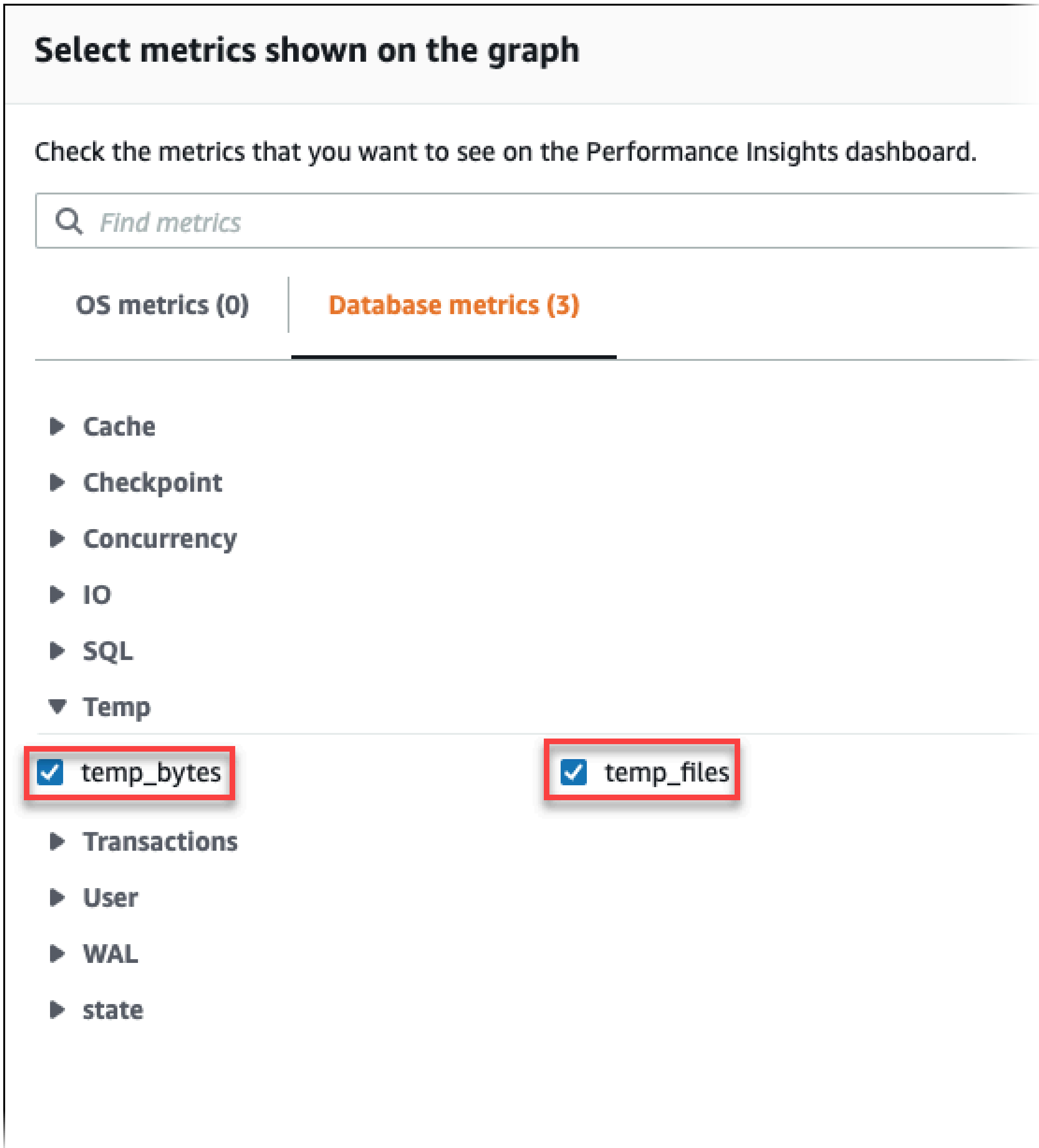
  성능 개선 도우미를 사용하여 지표를 분석하고 쿼리를 분석하는 방법에 대한 자세한 내용은 성능 개선 도우미 대시보드를 사용한 지표 분석 섹션을 참조하세요.

  성능 개선 도우미를 사용하여 임시 파일 사용량을 보는 예제는 성능 개선 도우미를 사용하여 임시 파일 사용량 확인 섹션을 참조하세요.

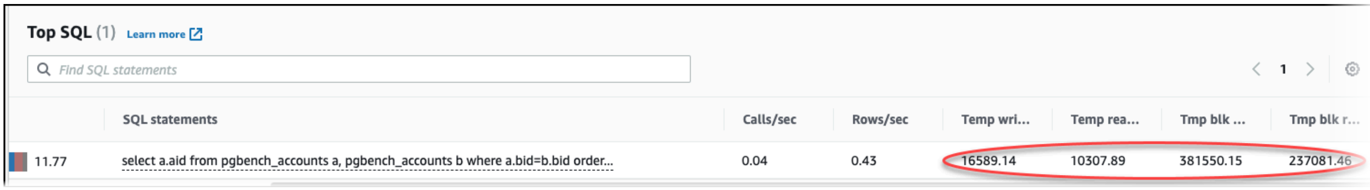
성능 개선 도우미를 사용하여 임시 파일 사용량 확인

성능 개선 도우미를 통해 지표 temp_bytes 및 temp_files를 설정하여 임시 파일 사용량을 볼 수 있습니다. 성능 개선 도우미 보기에는 임시 파일을 생성하는 특정 쿼리가 표시되지 않지만, `pg_ls_tmpdir`에 대해 표시된 쿼리와 성능 개선 도우미를 결합하면 쿼리 워크로드를 분석하고, 변경 사항을 확인하고, 문제를 해결할 수 있습니다.

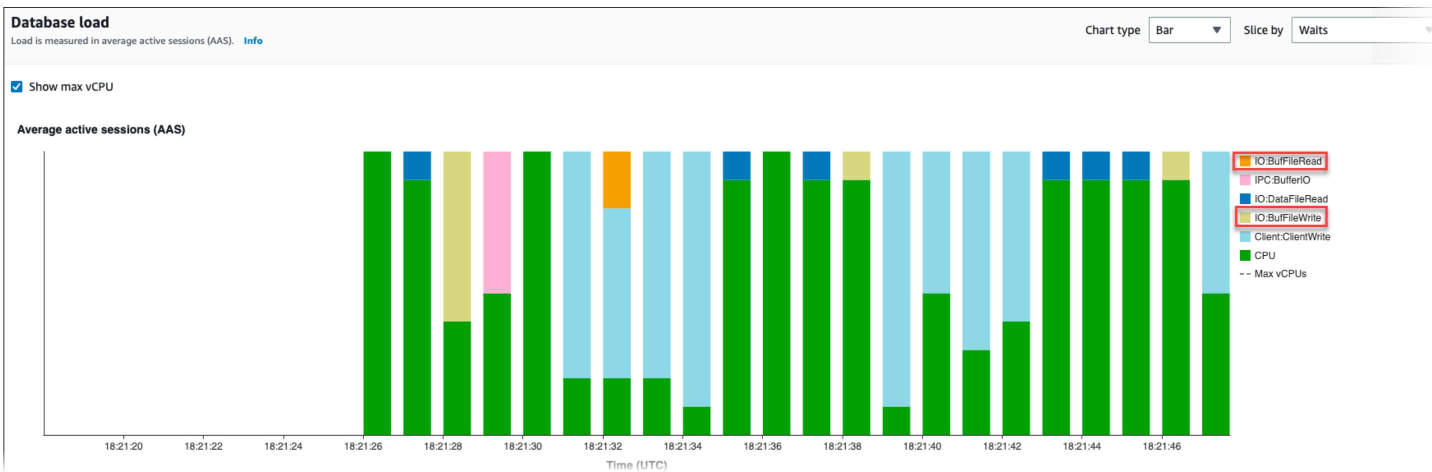
1. 성능 개선 도우미 대시보드에서 지표 관리를 선택합니다.
2. 다음 이미지에 나와 있는 것처럼 데이터베이스 지표를 선택하고, temp_bytes 및 temp_files를 선택합니다.



3. 상위 SQL 탭에서 기본 설정 아이콘을 선택합니다.
4. 기본 설정 창에서 상위 SQL 탭에 표시할 다음 통계를 켜고 계속을 선택합니다.
   - 임시 쓰기/초
   - 임시 읽기/초
   - 임시 대량 쓰기/호출
   - 임시 대량 읽기/호출
5. 다음 예제에 나온 것처럼, 임시 파일은 `pg_ls_tmpdir`에 대해 표시된 쿼리와 함께 통합될 때 분할됩니다.



IO:BufFileRead 및 IO:BufFileWrite 이벤트는 대개 워크로드의 상위 쿼리에서 임시 파일이 생성될 때 발생합니다. 데이터베이스 부하 및 상위 SQL 섹션의 AAS(상위 활성 세션)를 검토하여 IO:BufFileRead 및 IO:BufFileWrite에서 대기 중인 상위 쿼리를 식별하는 데 성능 개선 도우미를 사용할 수 있습니다.



성능 개선 도우미를 사용하여 상위 쿼리와 대기 이벤트에서 부하를 분석하는 방법에 대한 자세한 내용은 상위 SQL(Top SQL) 탭 개요 단원을 참조하세요. 임시 파일 사용량 및 관련 대기 이벤트를 늘리는 쿼리를 식별하고 조정해야 합니다. 이러한 대기 이벤트 및 수정에 대한 자세한 내용은 IO:BufFileRead 및 IO:BufFileWrite를 참조하세요.

> **Note**
>
> 이 `work_mem` 파라미터는 정렬 작업의 메모리가 부족하여 결과가 임시 파일에 기록되는 시점을 제어합니다. 이 파라미터의 설정을 기본값보다 높게 변경하면 모든 데이터베이스 세션에서 더 많은 메모리를 사용할 수 있으므로 변경하지 않는 것이 좋습니다. 또한 복잡한 조인 및 정렬을 수행하는 단일 세션에서는 각 작업이 메모리를 사용하는 병렬 작업을 수행할 수 있습니다. 여러 조인 및 정렬이 포함된 대용량 보고서가 있을 경우 `SET work_mem` 명령을 사용하여 세션 수준에서 이 파라미터를 설정하는 것이 가장 좋습니다. 그러면 변경 사항이 현재 세션에만 적용되고 값이 전체적으로 변경되지 않습니다.

Aurora PostgreSQL의 대기 이벤트를 사용한 튜닝

대기 이벤트는 Aurora PostgreSQL의 중요한 조정 도구입니다. 세션이 리소스를 기다리는 이유와 어떤 작업을 수행하는지 알 수 있다면 병목 현상을 더 잘 줄일 수 있습니다. 이 섹션의 정보를 통해 가능한 원인과 해결 조치를 찾을 수 있습니다. 이 섹션을 자세히 살펴보기 전에 기본적인 Aurora 개념, 특히 다음 주제를 이해하는 것이 좋습니다.

- Amazon Aurora 스토리지
- Aurora DB 클러스터의 성능 및 확장 관리

> **Important**
>
> 이 섹션에서 다루는 대기 이벤트는 Aurora PostgreSQL에만 해당됩니다. 이 섹션의 정보는 RDS for PostgreSQL이 아닌 Amazon Aurora만 튜닝하는 데 사용하세요.
> 이 섹션에서 다루는 일부 대기 이벤트는 이러한 데이터베이스 엔진의 오픈 소스 버전에서는 유사한 것이 없습니다. 다른 대기 이벤트는 오픈 소스 엔진의 이벤트와 이름이 같지만 다르게 동작합니다. 예를 들어, Amazon Aurora 스토리지는 오픈 소스 스토리지와 다르게 작동하므로 스토리지 관련 대기 이벤트는 리소스 상태가 다르다는 것을 나타냅니다.

주제

- Aurora PostgreSQL 튜닝을 위한 필수 개념
- Aurora PostgreSQL 대기 이벤트
- Client:ClientRead
- Client:ClientWrite
- CPU
- IO:BufFileRead 및 IO:BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- IPC:parallel 대기 이벤트
- Lock:advisory
- Lock:extend
- Lock:Relation

- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO(IPC:BufferIO)
- LWLock:lock_manager
- LWLock:MultiXact
- LWLock:pg_stat_statements
- Timeout:PgSleep

Aurora PostgreSQL 튜닝을 위한 필수 개념

Aurora PostgreSQL 데이터베이스를 조정하기 전에 대기 이벤트가 무엇인지, 왜 발생하는지 확인해 보세요. 또한 Aurora PostgreSQL의 기본 메모리 및 디스크 아키텍처도 검토하세요. 유용한 아키텍처 다이어그램은 PostgreSQL 위키북을 참조하세요.

주제

- Aurora PostgreSQL 대기 이벤트
- Aurora PostgreSQL 메모리
- Aurora PostgreSQL 프로세스

Aurora PostgreSQL 대기 이벤트

대기 이벤트는 세션이 대기 중인 리소스를 나타냅니다. 예를 들어 대기 이벤트 `Client:ClientRead`는 Aurora PostgreSQL이 클라이언트로부터 데이터를 수신하기 위해 대기 중일 때 발생합니다. 세션이 기다리는 일반적인 리소스는 다음과 같습니다.

- 버퍼에 대한 단일 스레드 액세스(예: 세션이 버퍼 수정을 시도하는 경우)
- 현재 다른 세션에 의해 잠겨 있는 행
- 데이터 파일 읽기
- 로그 파일 쓰기

예를 들어 쿼리를 충족하기 위해 세션에서 전체 테이블 스캔을 수행할 수 있습니다. 데이터가 아직 메모리에 없는 경우 세션은 디스크 I/O가 완료될 때까지 기다립니다. 버퍼를 메모리로 읽으면 다른 세션

이 동일한 버퍼에 액세스하기 때문에 세션을 기다려야 할 수 있습니다. 데이터베이스는 미리 정의된 대기 이벤트를 사용하여 대기를 기록합니다. 이러한 이벤트는 카테고리로 그룹화됩니다.

대기 이벤트 자체가 성능 문제를 나타내지는 않습니다. 예를 들어 요청된 데이터가 메모리에 없으면 디스크에서 데이터를 읽어야 합니다. 한 세션이 업데이트를 위해 행을 잠그면 다른 세션은 해당 행의 잠금이 해제될 때까지 기다려 업데이트할 수 있습니다. 커밋을 수행하려면 로그 파일에 대한 쓰기가 완료될 때까지 대기해야 합니다. 대기는 데이터베이스의 정상적인 기능에 필수적입니다.

많은 수의 대기 이벤트는 일반적으로 성능 문제를 나타냅니다. 이러한 경우 대기 이벤트 데이터를 사용하여 세션이 시간을 소비하는 위치를 결정할 수 있습니다. 예를 들어 일반적으로 몇 분 안에 실행되는 보고서가 몇 시간 동안 실행되는 경우 총 대기 시간에 가장 많이 기여하는 대기 이벤트를 식별할 수 있습니다. 최상위 대기 이벤트의 원인을 확인할 수 있는 경우 성능을 향상시키는 변경 작업을 수행할 수 있습니다. 예를 들어 세션이 다른 세션에 의해 잠긴 행에서 대기 중인 경우 잠금 세션을 종료할 수 있습니다.

Aurora PostgreSQL 메모리

Aurora PostgreSQL 메모리는 공유 메모리와 로컬 메모리로 구분됩니다.

주제

- [Aurora PostgreSQL의 공유 메모리]
- [Aurora PostgreSQL의 로컬 메모리]

Aurora PostgreSQL의 공유 메모리

Aurora PostgreSQL 인스턴스가 시작될 때 공유 메모리를 할당합니다. 공유 메모리는 여러 하위 영역으로 나뉩니다. 다음에서 가장 중요한 사항에 대한 설명을 찾을 수 있습니다.

주제

- [공유 버퍼]
- [미리 쓰기 로그(WAL) 버퍼]

공유 버퍼

공유 버퍼 풀은 애플리케이션 연결에서 사용 중이거나 사용 중인 모든 페이지를 보관하는 Aurora PostgreSQL 메모리 영역입니다. 페이지는 디스크 블록의 메모리 버전입니다. 공유 버퍼 풀은 디스크에서 읽은 데이터 블록을 캐시합니다. 이 풀은 디스크에서 데이터를 다시 읽어야 할 필요성을 줄여 데이터베이스가 더 효율적으로 연산하게 합니다.

모든 테이블과 인덱스는 고정된 크기의 페이지 배열로 저장됩니다. 각 블록에는 행에 해당하는 여러 튜플이 포함되어 있습니다. 튜플은 모든 페이지에 저장할 수 있습니다.

공유 버퍼 풀에는 유한 메모리가 있습니다. 새 요청에 메모리에 없는 페이지가 필요하고 메모리가 더 이상 존재하지 않는 경우 Aurora PostgreSQL은 요청을 수용하기 위해 자주 사용되지 않는 페이지를 삭제합니다. 제거 정책은 클럭 스윕 알고리즘에 의해 구현됩니다.

`shared_buffers` 파라미터는 서버가 데이터 캐싱에 전념하는 메모리 양을 결정합니다.

미리 쓰기 로그(WAL) 버퍼

미리 쓰기 로그(WAL) 버퍼는Aurora PostgreSQL이 나중에 영구 스토리지에 쓰는 트랜잭션 데이터를 보유합니다. WAL 메커니즘을 사용하여 Aurora PostgreSQL이 다음을 수행할 수 있습니다.

- 장애 발생 후 데이터 복구
- 디스크에 빈번한 쓰기를 방지하여 디스크 I/O 감소

클라이언트가 데이터를 변경하면 Aurora PostgreSQL이 변경 사항을 WAL 버퍼에 기록합니다. 클라이언트가 `COMMIT`을 발행하면 WAL 라이터 프로세스는 트랜잭션 데이터를 WAL 파일에 씁니다.

`wal_level` 파라미터는 WAL에 기록되는 정보의 양을 결정합니다.

Aurora PostgreSQL의 로컬 메모리

모든 백엔드 프로세스는 쿼리 처리를 위해 로컬 메모리를 할당합니다.

주제

- 작업 메모리 영역
- 유지 관리 작업 메모리 영역
- 임시 버퍼 영역

작업 메모리 영역

작업 메모리 영역은 정렬 및 해시를 수행하는 쿼리에 대한 임시 데이터를 보유합니다. 예를 들어, 다음을 포함하는 쿼리 `ORDER BY` 절은 정렬을 수행합니다. 쿼리는 해시 조인 및 집계에서 해시 테이블을 사용합니다.

`work_mem` 파라미터는 임시 디스크 파일에 쓰기 전에 내부 정렬 작업 및 해시 테이블에서 사용할 메모리 양입니다. 기본값은 4MB입니다. 여러 세션을 동시에 실행할 수 있으며 각 세션은 유지 관리 작업을

병렬로 실행할 수 있습니다. 이러한 이유로 사용되는 총 작업 메모리는 `work_mem` 설정의 배수가 될 수 있습니다.

유지 관리 작업 메모리 영역

유지 관리 작업 메모리 영역은 유지 관리 작업을 위해 데이터를 캐시합니다. 이러한 작업에는 베큠, 인덱스 생성 및 외래 키 추가가 포함됩니다.

`maintenance_work_mem` 파라미터는 유지 관리 작업에서 사용할 최대 메모리 양을 지정합니다. 기본값은 64MB입니다. 데이터베이스 세션은 한 번에 하나의 유지 관리 작업만 실행할 수 있습니다.

임시 버퍼 영역

임시 버퍼 영역에서는 각 데이터베이스 세션에 대한 임시 테이블을 캐시합니다.

각 세션은 사용자가 지정한 한도까지 필요에 따라 임시 버퍼를 할당합니다. 세션이 끝나면 서버는 버퍼를 지웁니다.

`temp_buffers` 파라미터는 각 세션에서 사용하는 임시 버퍼의 최대 수를 설정합니다. 세션 내에서 임시 테이블을 처음 사용하기 전에 `temp_buffers` 값을 변경할 수 있습니다.

Aurora PostgreSQL 프로세스

Aurora PostgreSQL 여러 프로세스를 사용합니다.

주제

- Postmaster 프로세스
- 백엔드 프로세스
- 백그라운드 프로세스

Postmaster 프로세스

Postmaster 프로세스는 Aurora PostgreSQL을 시작할 때 시작되는 첫 번째 프로세스입니다. 포스트마스터 프로세스는 다음과 같은 주요 책임이 있습니다.

- 백그라운드 프로세스 포크 및 모니터링
- 클라이언트 프로세스에서 인증 요청을 수신하고 데이터베이스에서 요청을 처리하도록 허용하기 전에 인증 요청을 인증합니다.

백엔드 프로세스

Postmaster가 클라이언트 요청을 인증하면 postmaster는 postgres 프로세스라고도 하는 새 백엔드 프로세스를 생성합니다. 하나의 클라이언트 프로세스가 정확히 하나의 백엔드 프로세스에 연결됩니다. 클라이언트 프로세스와 백엔드 프로세스는 포스트마스터 프로세스의 개입 없이 직접 통신합니다.

백그라운드 프로세스

Postmaster 프로세스는 서로 다른 백엔드 작업을 수행하는 여러 프로세스를 생성합니다. 몇 가지 중요한 사항은 다음과 같습니다.

- WAL 라이터

  Aurora PostgreSQL 미리 쓰기 로깅(WAL) 버퍼의 데이터를 로그 파일에 씁니다. 미리 쓰기 로깅의 원칙은 데이터베이스가 변경 사항을 설명하는 로그 레코드를 디스크에 기록하기 전까지는 데이터베이스가 데이터 파일에 변경 사항을 쓸 수 없다는 것입니다. WAL 메커니즘은 디스크 I/O를 줄이고, Aurora PostgreSQL 장애 후 로그를 사용하여 데이터베이스를 복구할 수 있도록 합니다.
- 백그라운드 라이터

  이 프로세스는 메모리 버퍼에서 데이터 파일로 더티(수정된) 페이지를 주기적으로 씁니다. 백엔드 프로세스가 메모리에서 페이지를 수정하면 페이지가 더티(dirty) 상태가 됩니다.
- Autovacuum 데몬

  데몬은 다음 구성 요소로 이루어져 있습니다.

  - Autovacuum 시작 관리자
  - Autovacuum 작업자 프로세스

  Autovacuum은 삽입되고 업데이트되거나 삭제된 튜플 수가 많은 테이블이 있는지 확인합니다. 데몬에는 다음과 같은 책임이 있습니다.

  - 업데이트되거나 삭제된 행이 차지하는 디스크 공간 복구 또는 재사용
  - 플래너가 사용하는 통계 업데이트
  - 트랜잭션 ID 랩어라운드로 인해 이전 데이터 손실 방지

  Autovacuum 기능은 `VACUUM`과 `ANALYZE` 명령을 자동화합니다. `VACUUM`에는 다음과 같은 표준 및 전체 변형이 있습니다. 스탠다드 베큠은 다른 데이터베이스 연산과 병행하여 실행됩니다. `VACUUM FULL`에는 작업 중인 테이블에 독점적인 잠금이 필요합니다. 따라서 동일한 테이블에 액세스하는 연산과 병렬로 실행할 수 없습니다. `VACUUM`은 상당한 양의 I/O 트래픽을 생성하여 다른 활성 세션의 성능이 저하될 수 있습니다.

Aurora PostgreSQL 대기 이벤트

다음 표에는 성능 문제를 가장 일반적으로 나타내는 Aurora PostgreSQL에 대한 대기 이벤트가 나열되어 있으며 가장 일반적인 원인과 해결 조치가 요약되어 있습니다. 다음 대기 이벤트는 Amazon Aurora PostgreSQL 대기 이벤트 목록의 하위 집합입니다.

대기 이벤트	정의
Client:ClientRead	이 이벤트는 Aurora PostgreSQL이 클라이언트에서 데이터를 수신하기 위해 대기 중일 때 발생합니다.
Client:ClientWrite	이 이벤트는 Aurora PostgreSQL이 클라이언트에 데이터를 쓰기 위해 대기 중일 때 발생합니다.
CPU	이 이벤트는 스레드가 CPU에서 활성 상태이거나 CPU를 대기 중일 때 발생합니다.
IO:BufFileRead 및 IO:BufFileWrite	이러한 이벤트는 Aurora PostgreSQL이 임시 파일을 만들 때 발생합니다.
IO:DataFileRead	이 이벤트는 공유 메모리에서 페이지를 사용할 수 없기 때문에 연결이 백엔드 프로세스에서 저장소에서 필요한 페이지를 읽도록 대기할 때 발생합니다.
IO:XactSync	이 이벤트는 데이터베이스가 Aurora 스토리지 하위 시스템이 일반 트랜잭션의 커밋을 승인하거나 준비된 트랜잭션의 커밋 또는 롤백을 확인할 때까지 대기 중일 때 발생합니다.
IPC:DamRecordTxAck	이 이벤트는 데이터베이스 활동 스트림을 사용하는 세션에서 Aurora PostgreSQL이 활동 스트림 이벤트를 생성한 다음 해당 이벤트가 지속될 때까지 기다리는 경우에 발생합니다.
Lock:advisory	이 이벤트는 PostgreSQL 애플리케이션이 잠금을 사용하여 여러 세션에서 활동을 조정할 때 발생합니다.
Lock:extend	이 이벤트는 백엔드 프로세스가 릴레이션을 확장하기 위해 릴레이션을 잠그기를 기다리는 동안 다른 프

대기 이벤트	정의
	로세스에서 동일한 목적으로 해당 릴레이션에 대한 잠금이 있는 경우에 발생합니다.
Lock:Relation	이 이벤트는 쿼리가 현재 다른 트랜잭션에 의해 잠긴 테이블 또는 뷰에 대한 잠금을 얻기 위해 대기 중일 때 발생합니다.
Lock:transactionid	이 이벤트는 트랜잭션이 행 수준 잠금을 대기 중일 때 발생합니다.
Lock:tuple	이 이벤트는 백엔드 프로세스가 튜플에 대한 잠금 획득을 대기 중일 때 발생합니다.
LWLock:buffer_content (BufferContent)	이 대기 이벤트는 다른 세션에서 특정 데이터 페이지에 대해 쓰기 잠금을 설정한 동안 세션에서 메모리 내 해당 페이지 읽기 또는 쓰기를 대기 중일 때 발생합니다.
LWLock:buffer_mapping	이 이벤트는 세션이 데이터 블록을 공유 버퍼 풀의 버퍼와 연결하기 위해 대기 중일 때 발생합니다.
LWLock:BufferIO(IPC:BufferIO)	이 이벤트는 Aurora PostgreSQL 또는 RDS for PostgreSQL이 페이지에 동시에 액세스하려고 할 때 다른 프로세스가 입출력(I/O) 작업을 완료할 때까지 기다리는 경우에 발생합니다.
LWLock:lock_manager	이 이벤트는 Aurora PostgreSQL 엔진이 빠른 경로 잠금이 불가능할 때 잠금을 할당, 확인 및 할당 해제하기 위해 공유 잠금 메모리 영역을 유지 관리하는 경우에 발생합니다.

대기 이벤트	정의
LWLock:MultiXact	이러한 유형의 이벤트는 테이블의 동일한 행과 관련된 여러 트랜잭션을 완료하기 위해 Aurora PostgreSQL이 세션을 열린 상태로 유지할 때 발생합니다. 대기 이벤트는 다중 트랜잭션 처리의 요소, 즉 LWLock:MultixactOffsetBuffset Buffer, LWlock:MultixactMemberSLRU 또는 LWLock:MultixActMemberBuffer 중 무엇이 대기 이벤트를 생성하는지를 표시합니다.
Timeout:PgSleep	이 이벤트는 서버 프로세스가 `pg_sleep` 함수 및 절전 시간 초과가 만료될 때까지 대기할 때 발생합니다.

Client:ClientRead

`Client:ClientRead` 이벤트는 Aurora PostgreSQL이 클라이언트에서 데이터를 수신하기 위해 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 10 이상에서 지원됩니다.

컨텍스트

Aurora PostgreSQL DB 클러스터가 클라이언트로부터 데이터를 수신하기 위해 대기 중입니다. Aurora PostgreSQL DB 클러스터는 클라이언트로부터 데이터를 수신해야 클라이언트에 더 많은 데이터를 보낼 수 있습니다. 클라이언트에서 데이터를 수신하기 전에 클러스터가 대기하는 시간은 `Client:ClientRead` 이벤트를 트리거합니다.

대기 증가의 가능한 원인

상위 대기에서 나타나는 `Client:ClientRead` 이벤트의 일반적인 원인은 다음을 포함합니다.

네트워크 대기 시간 증가

Aurora PostgreSQL DB 클러스터와 클라이언트 간에 네트워크 지연 시간이 늘어날 수 있습니다. 네트워크 지연 시간이 높을수록 DB 클러스터가 클라이언트에서 데이터를 수신하는 데 필요한 시간이 늘어납니다.

클라이언트에 대한 로드 증가

클라이언트에 CPU 압력 또는 네트워크 포화가 있을 수 있습니다. 클라이언트 로드가 증가하면 클라이언트에서 Aurora PostgreSQL DB 클러스터로의 데이터 전송이 지연될 수 있습니다.

과도한 네트워크 왕복

Aurora PostgreSQL DB 클러스터와 클라이언트 간의 네트워크 왕복 횟수가 많으면 클라이언트에서 Aurora PostgreSQL DB 클러스터로의 데이터 전송을 지연시킬 수 있습니다.

대량 복사 연산

복사 연산 중에 데이터가 클라이언트의 파일 시스템에서 Aurora PostgreSQL DB 클러스터로 전송됩니다. DB 클러스터에 대량의 데이터를 전송하면 클라이언트에서 DB 클러스터로 데이터 전송이 지연될 수 있습니다.

유휴 클라이언트 연결

Aurora PostgreSQL DB 인스턴스에 대한 연결이 트랜잭션 상태에서 유휴 상태이며 클라이언트가 추가 데이터를 전송하거나 명령을 실행하기를 기다리고 있습니다. 이 상태는 `Client:ClientRead` 이벤트 증가로 이어질 수 있습니다.

연결 풀링에 사용되는 PGBouncer

PGBouncer에는 `pkt_buf`와 같은 낮은 수준의 네트워크 구성 설정이 있으며, 기본적으로 4,096으로 설정됩니다. 워크로드가 PGBouncer를 통해 4,096바이트보다 큰 쿼리 패킷을 보내는 경우, `pkt_buf` 설정을 8,192까지 늘리는 것이 좋습니다. 새 설정으로 인해 `Client:ClientRead` 이벤트의 수가 줄어들지 않는 경우 `pkt_buf` 설정을 16,384 또는 32,768과 같이 더 큰 값으로 설정하는 것이 좋습니다. 쿼리 텍스트가 큰 경우 더 큰 설정이 특히 유용할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

클러스터와 동일한 가용 영역 및 VPC 서브넷에 클라이언트를 배치합니다.

네트워크 대기 시간을 줄이고 네트워크 처리량을 늘리려면 Aurora PostgreSQL DB 클러스터와 동일한 가용 영역 및 Virtual Private Cloud(VPC) 서브넷에 클라이언트를 배치합니다. 클라이언트가 가능한 한 DB 클러스터에 지리적으로 가까이 있는지 확인합니다.

클라이언트 확장

Amazon CloudWatch 또는 기타 호스트 지표를 사용하여 클라이언트가 현재 CPU 또는 네트워크 대역폭 또는 둘 다에 의해 제한되어 있는지 확인합니다. 클라이언트가 제한된 경우 그에 따라 클라이언트를 확장합니다.

현재 세대 인스턴스 사용

점보 프레임을 지원하는 DB 인스턴스 클래스를 사용하지 않는 경우도 있습니다. Amazon EC2 애플리케이션을 실행하는 경우 클라이언트에 현재 세대 인스턴스를 사용하는 것이 좋습니다. 또한 클라이언트 운영 체제에서 최대 전송 단위(MTU)를 구성합니다. 이 기술은 네트워크 왕복 수를 줄이고 네트워크 처리량을 늘릴 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 점보 프레임(9001 MTU)을 참조하세요.

DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요. Amazon EC2 인스턴스 유형과 동일한 DB 인스턴스 클래스를 확인하려면 `db.`가 Amazon EC2 인스턴스 유형 이름 앞에 있어야 합니다. 예를 들어, `r5.8xlarge` Amazon EC2 인스턴스는 `db.r5.8xlarge` DB 인스턴스 클래스입니다.

네트워크 대역폭 향상

DB 클러스터의 수신 및 발신 네트워크 트래픽을 모니터링하는 `NetworkReceiveThroughput`과 `NetworkTransmitThroughput` Amazon CloudWatch 지표를 사용합니다. 이러한 지표는 네트워크 대역폭이 워크로드에 충분한지 확인하는 데 도움이 될 수 있습니다.

네트워크 대역폭이 충분하지 않으면 늘리세요. 만약 AWS 클라이언트 또는 DB 인스턴스가 네트워크 대역폭 제한에 도달했다면. 대역폭을 늘리는 유일한 방법은 DB 인스턴스 크기를 늘리는 것입니다.

CloudWatch 지표에 대한 자세한 내용은 [Amazon Aurora에 대한 Amazon CloudWatch 지표](#) 섹션을 참조하세요.

최대 네트워크 성능 모니터링

Amazon EC2 클라이언트를 사용하는 경우 Amazon EC2는 집계 인바운드 및 아웃바운드 네트워크 대역폭을 포함하여 네트워크 성능 지표에 대한 최댓값을 제공합니다. 또한 연결 추적을 제공하여 패킷이 예상대로 반환되고 도메인 이름 시스템(DNS)과 같은 서비스에 대한 링크-로컬 서비스 액세스를 제공합니다. 이러한 최댓값을 모니터링하려면 현재 향상된 네트워킹 드라이버를 사용하고 클라이언트의 네트워크 성능을 모니터링하세요.

자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스의 네트워크 성능 모니터링과 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스의 네트워크 성능 모니터링을 참조하세요.

'트랜잭션에서 유휴' 상태의 트랜잭션 모니터링

`idle in transaction` 연결의 수가 점점 늘어나고 있는지 확인하세요. 이를 수행하려면 `state`열의 `pg_stat_activity`테이블을 모니니터링하세요. 다음과 유사한 쿼리를 실행하여 연결 소스를 식별할 수 있습니다.

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

`Client:ClientWrite` 이벤트는 Aurora PostgreSQL이 클라이언트에 데이터를 쓰기 위해 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 10 이상에서 지원됩니다.

컨텍스트

클라이언트 프로세스는 클러스터가 더 많은 데이터를 보낼 수 있게 되기 전 반드시 Aurora PostgreSQL DB 클러스터로부터 받은 모든 데이터를 읽어야 합니다. 클라이언트에서 데이터를 수신하기 전에 클러스터가 대기하는 시간은 `Client:ClientWrite` 이벤트를 트리거합니다.

Aurora PostgreSQL DB 클러스터와 클라이언트 간의 네트워크 처리량이 감소하면 이 이벤트가 발생할 수 있습니다. 클라이언트에 대한 CPU 압력 및 네트워크 포화 상태시에도 이 이벤트가 발생할 수 있습니다. CPU 압력은 CPU가 완전히 활용되고 CPU 시간을 기다리는 작업이 있을 때입니다. 네트워크 포화는 데이터베이스와 클라이언트 간의 네트워크가 처리할 수 있는 것보다 많은 데이터를 전달하는 경우입니다.

대기 증가의 가능한 원인

상위 대기에서 나타나는 `Client:ClientWrite` 이벤트의 일반적인 원인은 다음을 포함합니다.

네트워크 대기 시간 증가

> Aurora PostgreSQL DB 클러스터와 클라이언트 간에 네트워크 지연 시간이 늘어날 수 있습니다. 네트워크 대기 시간이 높을수록 클라이언트가 데이터를 수신하는 데 필요한 시간이 늘어납니다.

클라이언트에 대한 로드 증가

> 클라이언트에 CPU 압력 또는 네트워크 포화가 있을 수 있습니다. 클라이언트에 대한 로드가 증가하면 Aurora MySQL DB 클러스터에서 데이터 수신이 지연됩니다.

클라이언트에 전송되는 대용량 데이터

> Aurora PostgreSQL DB 클러스터가 많은 양의 데이터를 클라이언트에 전송하고 있을 수 있습니다. 클라이언트는 클러스터가 데이터를 전송하는 만큼 빠르게 데이터를 수신하지 못할 수 있습니다. 큰 테이블 복사와 같은 활동 시 `Client:ClientWrite` 이벤트가 증가할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 클러스터와 동일한 가용 영역 및 VPC 서브넷에 클라이언트를 배치합니다.
- 현재 세대 인스턴스 사용
- 클라이언트에 전송해야 하는 데이터 양 감소
- 클라이언트 확장

클러스터와 동일한 가용 영역 및 VPC 서브넷에 클라이언트를 배치합니다.

네트워크 대기 시간을 줄이고 네트워크 처리량을 늘리려면 Aurora PostgreSQL DB 클러스터와 동일한 가용 영역 및 Virtual Private Cloud(VPC) 서브넷에 클라이언트를 배치합니다.

현재 세대 인스턴스 사용

점보 프레임을 지원하는 DB 인스턴스 클래스를 사용하지 않는 경우도 있습니다. Amazon EC2 애플리케이션을 실행하는 경우 클라이언트에 현재 세대 인스턴스를 사용하는 것이 좋습니다. 또한 클라이언트 운영 체제에서 최대 전송 단위(MTU)를 구성합니다. 이 기술은 네트워크 왕복 수를 줄이고 네트워크 처리량을 늘릴 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 점보 프레임(9001 MTU)을 참조하세요.

DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요. Amazon EC2 인스턴스 유형과 동일한 DB 인스턴스 클래스를 확인하려면 `db.`가 Amazon EC2 인스턴스 유형 이름 앞에 있어야 합니다. 예를 들어, `r5.8xlarge` Amazon EC2 인스턴스는 `db.r5.8xlarge` DB 인스턴스 클래스입니다.

클라이언트에 전송해야 하는 데이터 양 감소

가능하면 애플리케이션을 조정하여 Aurora PostgreSQL DB 클러스터가 클라이언트에 보내는 데이터의 양을 줄이세요. 이러한 조정을 수행하면 클라이언트에서 CPU 및 네트워크 경합을 줄일 수 있습니다.

클라이언트 확장

Amazon CloudWatch 또는 기타 호스트 지표를 사용하여 클라이언트가 현재 CPU 또는 네트워크 대역폭 또는 둘 다에 의해 제한되어 있는지 확인합니다. 클라이언트가 제한된 경우 그에 따라 클라이언트를 확장합니다.

CPU

이 이벤트는 스레드가 CPU에서 활성 상태이거나 CPU를 대기 중일 때 발생합니다.

주제

지원되는 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 9.6 이상에서 지원됩니다.

컨텍스트

중앙 처리 장치는 명령을 실행하는 컴퓨터의 구성 요소입니다. 예를 들어 CPU 명령은 연산 연산을 수행하고 메모리에서 데이터를 교환합니다. 쿼리가 데이터베이스 엔진을 통해 수행하는 명령 수를 늘리면 쿼리 실행 시간이 늘어납니다. CPU 스케줄링은 프로세스에 CPU 시간을 부여합니다. 스케줄링은 운영 체제 커널에 의해 조정됩니다.

주제

이 대기 시간이 언제 발생하는지 확인하는 방법

CPU 대기 이벤트는 백엔드 프로세스가 CPU에서 활성 상태이거나 CPU를 기다리고 있음을 나타냅니다. 쿼리에 다음 정보가 표시될 때 발생한다는 것을 알고 있습니다.

- `pg_stat_activity.state` 열이 `active` 값을 갖고 있습니다.
- `pg_stat_activity`의 `wait_event_type`과 `wait_event` 열은 모두 `null`입니다.

CPU를 사용하거나 대기 중인 백엔드 프로세스를 보려면 다음 쿼리를 실행하세요.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
```

```
AND    wait_event IS NULL;
```

DBloadCPU 지표

CPU 성능 개선 도우미의 지표는 DBLoadCPU입니다. DBLoadCPU의 값은 Amazon CloudWatch 지표 CPUUtilization의 값과 다를 수 있습니다. 후자의 지표는 데이터베이스 인스턴스에 대한 HyperVisor에서 수집됩니다.

os.cpuUtilization 지표

성능 개선 도우미 운영 시스템 지표는 CPU 사용률에 대한 자세한 정보를 제공합니다. 예를 들어 다음 지표가 표시될 수 있습니다.

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

성능 개선 도우미는 데이터베이스 엔진의 CPU 사용량을 `os.cpuUtilization.nice.avg`와 같이 보고합니다.

CPU 스케줄링의 원인

운영 체제 관점에서 볼 때 CPU는 유휴 스레드를 실행하지 않을 때 활성화됩니다. CPU는 계산을 수행하는 동안 활성 상태이지만 메모리 I/O에서 대기할 때도 활성화됩니다. 이러한 유형의 I/O는 일반적인 데이터베이스 워크로드를 지배합니다.

다음 조건이 충족되면 프로세스가 CPU에 스케줄링될 때까지 대기할 수 있습니다.

- CloudWatch `CPUUtilization` 지표가 100%에 가깝습니다.
- 평균 로드가 vCPU 수보다 크므로 부하가 많음을 나타냅니다. 성능 개선 도우미의 OS 지표 섹션에서 `loadAverageMinute` 지표를 찾을 수 있습니다.

대기 증가의 가능한 원인

이 이벤트가 정상 상태보다 많이 발생하여 성능 문제를 나타낼 수 있는 경우 일반적인 원인은 다음과 같습니다.

주제

- [갑작스러운 스파이크의 원인]
- [장기 고주파의 원인]
- [코너 케이스]

갑작스러운 스파이크의 원인

갑작스런 스파이크의 가장 큰 원인은 다음과 같습니다.

- 애플리케이션이 데이터베이스에 대한 동시 연결을 너무 많이 열었습니다. 이 시나리오를 '연결 폭풍'이라고 합니다.
- 다음 방법 중 하나로 애플리케이션 워크로드가 변경되었습니다.
  - 새 쿼리
  - 데이터 집합 크기 증가
  - 인덱스 유지 관리 또는 생성
  - 새로운 함수
  - 새로운 연산자
  - 병렬 쿼리 실행의 증가
- 쿼리 실행 계획이 변경되었습니다. 경우에 따라 변경으로 인해 버퍼가 증가할 수 있습니다. 예를 들어, 쿼리는 이전에 인덱스를 사용했을 때의 순차적 스캔을 사용하고 있습니다. 이 경우 쿼리는 동일한 목표를 달성하기 위해 더 많은 CPU가 필요합니다.

장기 고주파의 원인

장기간에 걸쳐 재발하는 이벤트의 가장 큰 원인은 다음과 같습니다.

- 너무 많은 백엔드 프로세스가 CPU에서 동시에 실행되고 있습니다. 이러한 프로세스는 병렬 작업자가 될 수 있습니다.
- 쿼리는 많은 수의 버퍼가 필요하기 때문에 차선적으로 수행됩니다.

코너 케이스

실제 원인으로 판명될 가능성이 있는 원인이 없다면 다음과 같은 상황이 발생할 수 있습니다.

- CPU가 프로세스를 안팎으로 교환하고 있습니다.
- CPU 컨텍스트 전환이 증가했습니다.

- Aurora PostgreSQL 코드에 대기 이벤트가 누락되었습니다.

작업

CPU 대기 이벤트가 데이터베이스 활동을 지배하는 경우 반드시 성능 문제를 나타내지는 않습니다. 성능이 저하되는 경우에만 이 이벤트에 응답하세요.

주제

- 데이터베이스로 인해 CPU 증가가 발생하는지 조사합니다.
- 연결 수 증가 여부 확인
- 워크로드 변경에 대응

데이터베이스로 인해 CPU 증가가 발생하는지 조사합니다.

성능 개선 도우미의 `os.cpuUtilization.nice.avg` 지표를 검토합니다. 이 값이 CPU 사용량보다 훨씬 작다면 데이터베이스가 아닌 프로세스가 CPU의 주요 기여자입니다.

연결 수 증가 여부 확인

Amazon CloudWatch의 `DatabaseConnections` 지표를 검토합니다. 작업은 CPU 대기 이벤트 증가 시기 동안 증가 또는 감소 여부에 따라 달라집니다.

연결이 증가했습니다.

연결 수가 증가했다면, CPU를 사용하는 백엔드 프로세스 수를 vCPU 수와 비교하세요. 가능한 시나리오는 다음과 같습니다.

- CPU를 사용하는 백엔드 프로세스 수가 vCPU 수보다 적습니다.

  이 경우 연결 수는 문제가 되지 않습니다. 그러나 계속해서 CPU 사용률을 줄이려고 할 수도 있습니다.
- CPU를 사용하는 백엔드 프로세스 수가 vCPU 수보다 적습니다.

  이 경우에는 다음 옵션을 고려하세요.
  - 데이터베이스에 연결된 백엔드 프로세스 수를 줄입니다. 예를 들어 RDS 프록시와 같은 연결 풀링 솔루션을 구현합니다. 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.
  - 인스턴스 크기를 업그레이드하여 vCPU 수를 늘립니다.
  - 해당하는 경우 일부 읽기 전용 워크로드를 리더 노드로 리디렉션하세요.

연결이 증가하지 않았습니다.

성능 개선 도우미의 blks_hit 지표를 검토합니다. blks_hit의 증가와 CPU 사용량 사이의 상관관계를 찾습니다. 가능한 시나리오는 다음과 같습니다.

- CPU 사용량 및 blks_hit이 상관관계가 있습니다.

  이 경우 CPU 사용량에 연결된 최상위 SQL 문을 찾아 계획 변경을 찾습니다. 다음과 같은 기술을 사용할 수 있습니다.

  - 계획을 수동으로 설명하고 예상 실행 계획과 비교합니다.
  - 초당 블록 히트와 초당 로컬 블록 히트가 증가하는지 확인합니다. 성능 개선 도우미의 대시보드의 상위 SQL 섹션에서 환경설정을 선택합니다.
- CPU 사용량 및 blks_hit이 상관관계가 없습니다.

  이 경우 다음 중 한 가지라도 발생하는지 확인합니다.

  - 애플리케이션이 데이터베이스에 급속하게 연결되고 데이터베이스와의 연결이 끊어지고 있습니다.

    log_connections와 log_disconnections를 켜 이 동작을 진단하고, 그런 다음 PostgreSQL 로그를 분석합니다. pgbadger 로그 분석기 사용을 고려하세요. 자세한 내용은 https://github.com/darold/pgbadger 섹션을 참조하세요.
  - OS에 과부하가 걸렸습니다.

    이 경우 성능 개선 도우미는 백엔드 프로세스가 평소보다 오랜 시간 동안 CPU를 사용하고 있음을 보여줍니다. 성능 개선 도우미 os.cpuUtilization 지표 또는 CloudWatch CPUUtilization 지표에서 증거를 찾습니다. 운영 체제에 과부하가 걸린 경우 향상된 모니터링 지표를 살펴보고 더 자세히 진단하세요. 특히 프로세스 목록과 각 프로세스에서 소비되는 CPU 비율을 살펴보세요.
  - 상위 SQL 문이 너무 많은 CPU를 소비하고 있습니다.

    CPU 사용량에 연결된 문을 검사하여 CPU를 더 적게 사용할 수 있는지 확인합니다. EXPLAIN 명령을 실행하고 가장 큰 영향을 미치는 계획 노드에 중점을 둡니다. PostgreSQL 실행 계획 비주얼라이저를 사용하는 것이 좋습니다. http://explain.dalibo.com/ 섹션을 참조해 이 도구를 사용해 보세요.

워크로드 변경에 대응

워크로드가 변경된 경우 다음 변경 유형을 찾습니다.

새 쿼리

새 쿼리가 예상되는지 확인합니다. 그렇다면 실행 계획과 초당 실행 횟수가 예상되는지 확인합니다.

데이터 집합 크기 증가

파티셔닝이 아직 구현되지 않은 경우 파티셔닝이 도움이 될지 확인합니다. 이 전략은 쿼리가 검색해야 하는 페이지 수를 줄일 수 있습니다.

인덱스 유지 관리 또는 생성

유지 관리 일정이 예상되는지 확인합니다. 모범 사례는 피크 활동 이외의 유지 관리 활동을 예약하는 것입니다.

새로운 함수

테스트 중에 이러한 함수가 예상대로 작동하는지 확인합니다. 특히 초당 실행 횟수가 예상되는지 확인합니다.

새로운 연산자

테스트 중에 이러한 함수가 예상대로 작동하는지 확인합니다.

병렬 쿼리 실행 증가

다음 상황 중 한 가지라도 발생했는지 확인합니다.

- 포함된 관계식 또는 인덱스의 크기가 갑자기 커져 `min_parallel_table_scan_size` 또는 `min_parallel_index_scan_size`와 크게 다릅니다.
- `parallel_setup_cost` 또는 `parallel_tuple_cost`에 최근 변경 사항이 적용되었습니다.
- `max_parallel_workers` 또는 `max_parallel_workers_per_gather`에 최근 변경 사항이 적용되었습니다.

IO:BufFileRead 및 IO:BufFileWrite

`IO:BufFileRead`와 `IO:BufFileWrite` 이벤트는 Aurora PostgreSQL이 임시 파일을 만들 때 발생합니다. 연산에 현재 정의된 작업 메모리 파라미터보다 많은 메모리가 필요한 경우 임시 데이터를 영구 스토리지에 씁니다. 이 작업을 '디스크로 유출'이라고도 합니다.

주제

- [지원되는 엔진 버전]

- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

`IO:BufFileRead`와 `IO:BufFileWrite`는 작업 메모리 영역 및 유지 관리 작업 메모리 영역과 관련이 있습니다. 이러한 로컬 메모리 영역에 대한 자세한 내용은 작업 메모리 영역 및 유지 관리 작업 메모리 영역 섹션을 참조하세요.

기본값은 `work_mem` 또는 4MB입니다. 한 세션이 병렬로 연산을 수행하는 경우 병렬 처리를 처리하는 각 작업자는 4MB의 메모리를 사용합니다. 이런 이유로, `work_mem`을 신중하게 설정하세요. 값을 너무 많이 늘리면 많은 세션을 실행하는 데이터베이스가 너무 많은 메모리를 소비할 수 있습니다. 값을 너무 낮게 설정하면 Aurora PostgreSQL이 로컬 스토리지에 임시 파일을 만듭니다. 이러한 임시 파일의 디스크 I/O는 성능을 저하시킬 수 있습니다.

다음과 같은 일련의 이벤트를 관찰하면 데이터베이스에서 임시 파일이 생성될 수 있습니다.

1. 갑작스럽고 급격한 가용성 감소
2. 여유 공간을 위한 신속한 복구

'체인톱' 패턴도 볼 수 있습니다. 이 패턴은 데이터베이스에서 지속적으로 작은 파일을 생성한다는 것을 의미할 수 있습니다.

대기 증가의 가능한 원인

일반적으로 이러한 대기 이벤트는 `work_mem` 또는 `maintenance_work_mem` 파라미터가 할당하는 것보다 더 많은 메모리를 소모하는 연산에 의해 발생합니다. 보정을 위해 연산은 임시 파일에 기록합니다. `IO:BufFileRead`와 `IO:BufFileWrite` 이벤트의 일반적인 원인에는 다음이 포함됩니다.

작업 메모리 영역에 존재하는 것보다 많은 메모리가 필요한 쿼리

다음 특성을 가진 쿼리는 작업 메모리 영역을 사용합니다.

- 해시 조인.

- `ORDER BY` 절
- `GROUP BY` 절
- `DISTINCT`
- 윈도 함수
- `CREATE TABLE AS SELECT`
- 구체화 뷰 새로고침

작업 메모리 영역에 존재하는 것보다 많은 메모리가 필요한 명령문

다음 명령문에서는 유지 관리 작업 메모리 영역을 사용합니다.

- `CREATE INDEX`
- `CLUSTER`

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 문제 식별
- 조인 쿼리 검토
- ORDER BY와 GROUP BY 쿼리를 검토합니다.
- DISTINCT 연산 사용을 피하세요.
- GROUP BY 함수 대신 윈도 함수를 사용하는 것이 좋습니다.
- 구체화된 뷰 및 CTAS 명령문 조사
- 인덱스를 만들 때 pg_repack 사용
- 테이블을 클러스터링할 때 maintenance_work_mem 증가
- 메모리를 조정하여 IO:BufFileRead 및 IO:BufFileWrite 방지

문제 식별

성능 개선 도우미가 켜져 있지 않고 `IO:BufFileRead`와 `IO:BufFileWrite`가 정상보다 더 자주 발생한다고 의심되는 상황을 가정합니다. 다음을 따릅니다.

1. Amazon CloudWatch의 `FreeLocalStorage` 지표를 검토합니다.
2. 톱니 모양의 스파이크 시리즈인 체인톱 패턴을 찾으세요.

체인톱 패턴은 임시 파일과 관련된 스토리지의 빠른 사용 및 릴리스를 나타냅니다. 이 패턴이 발견되면 성능 개선 도우미를 켜세요. 성능 개선 도우미를 사용하면 대기 이벤트가 발생하는 시기와 연관된 쿼리를 식별할 수 있습니다. 솔루션은 이벤트를 유발하는 특정 쿼리에 따라 달라집니다.

또는 `log_temp_files` 파라미터를 설정합니다. 이 파라미터는 임시 파일의 임계값 KB 이상을 생성하는 모든 쿼리를 기록합니다. 값이 `0`이라면, Aurora PostgreSQL이 모든 임시 파일을 기록합니다. 값이 `1024`라면, Aurora PostgreSQL이 1MB보다 큰 임시 파일을 생성하는 모든 쿼리를 기록합니다. `log_temp_files`에 관한 자세한 내용은 PostgreSQL 문서의 [오류 보고 및 로깅](#)을 참조하세요.

조인 쿼리 검토

애플리케이션에서 조인을 사용할 수 있습니다. 예를 들어 다음 쿼리는 네 개의 테이블을 조인합니다.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

임시 파일 사용량 스파이크의 원인은 쿼리 자체의 문제입니다. 예를 들어, 끊어진 절은 조인을 제대로 필터링하지 않을 수 있습니다. 다음 예에서 두 번째 내부 조인을 고려해 보겠습니다.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

위의 쿼리가 실수로 `customer.id`를 `customer.id`에 조인하여 모든 고객과 모든 주문 사이에 데카르트 프로덕트를 생성합니다. 이러한 유형의 우발적인 조인은 큰 임시 파일을 생성합니다. 테이블의 크기에 따라 테카르트 쿼리(Cartesian query)는 스토리지를 채울 수도 있습니다. 다음 조건이 충족되면 애플리케이션에 데카르트 조인(Cartesian join)이 있을 수 있습니다.

- 스토리지 가용성이 급격히, 크게 줄어들고 복구가 빨라집니다.
- 인덱스가 생성되지 않습니다.
- `CREATE TABLE FROM SELECT` 문이 발행되고 있지 않습니다.
- 구체화 뷰가 새로 고침이 되지 않습니다.

적절한 키를 사용하여 테이블이 조인되고 있는지 확인하려면 쿼리 및 객체 관계형 매핑 지시어를 검사합니다. 애플리케이션의 특정 쿼리가 항상 호출되는 것은 아니며 일부 쿼리는 동적으로 생성됩니다.

ORDER BY와 GROUP BY 쿼리를 검토합니다.

경우에 따라 `ORDER BY` 절에 임시 파일이 과도하게 발생할 수 있습니다. 다음 지침을 참고하세요.

- 정렬이 필요할 때 `ORDER BY` 절의 열만 포함합니다. 이 지침은 `ORDER BY` 절에서 수천 개의 행을 반환하고 많은 열을 지정하는 쿼리에 특히 중요합니다.
- 오름차순 또는 내림차순이 동일한 열과 일치할 때 `ORDER BY` 절을 가속하기 위해 인덱스를 생성하는 것이 좋습니다. 부분 인덱스는 작기 때문에 선호됩니다. 작은 인덱스를 더 빠르게 읽고 탐색할 수 있습니다.
- null 값을 허용할 수 있는 열에 대한 인덱스를 만드는 경우 null 값을 인덱스의 끝에 저장할지 아니면 인덱스의 시작 부분에 저장할지 고려해야 합니다.

  가능한 경우 결과 집합을 필터링하여 정렬해야 하는 행 수를 줄입니다. `WITH` 절의 명령문 또는 하위 쿼리를 사용할 경우, 내부 쿼리가 결과 집합을 생성하여 외부 쿼리로 전달한다는 것을 기억하세요. 쿼리가 필터링할 수 있는 행이 많을수록 쿼리 정렬이 할 일이 줄어듭니다.
- 전체 결과 집합을 가져올 필요가 없는 경우 `LIMIT` 절을 사용하세요. 예를 들어 상위 5개 행만 원하는 경우 `LIMIT` 절을 사용하는 쿼리는 결과를 계속 생성하지 않습니다. 이렇게 하면 쿼리에 메모리와 임시 파일이 더 적게 필요합니다.

`GROUP BY` 절을 사용하는 쿼리는 임시 파일이 필요할 수도 있습니다. `GROUP BY` 쿼리는 다음과 같은 함수를 사용하여 값을 요약합니다.

- `COUNT`
- `AVG`

- MIN
- MAX
- SUM
- STDDEV

GROUP BY 쿼리를 튜닝하려면 ORDER BY 쿼리의 권장 사항을 따르세요.

DISTINCT 연산 사용을 피하세요.

가능하면 중복된 행을 제거하기 위해 DISTINCT 연산을 사용하지 마세요. 쿼리가 반환하는 불필요하고 중복된 행이 많을수록 DISTINCT 연산에 시간이 오래 걸리게 됩니다. 가능하면 WHERE 절에 필터를 추가합니다. 다른 테이블에 동일한 필터를 사용하는 경우에도 마찬가지입니다. 쿼리를 필터링하고 조인하면 성능이 향상되고 리소스 사용이 줄어듭니다. 또한 잘못된 보고와 결과를 방지할 수 있습니다.

DISTINCT를 동일 테이블의 여러 행에서 사용해야 하는 경우 복합 인덱스를 만드는 것이 좋습니다. 인덱스에서 여러 열을 그룹화하면 고유한 행을 평가하는 시간을 단축할 수 있습니다. 또한 Amazon Aurora PostgreSQL 버전 10 이상을 사용하는 경우, CREATE STATISTICS 명령을 통해 다중 열의 통계를 상호 연관시킬 수 있습니다.

GROUP BY 함수 대신 윈도 함수를 사용하는 것이 좋습니다.

GROUP BY를 사용하여 결과 집합을 변경한 다음 집계된 결과를 검색합니다. 윈도 함수를 사용하면 결과 집합을 변경하지 않고 데이터를 집계할 수 있습니다. 윈도 함수는 OVER 절을 통해 다른 행과 상호 연관시키는 쿼리에 의해 정의된 집합에서 계산을 수행합니다. 윈도 함수의 모든 GROUP BY 함수를 사용할 수 있으며, 다음과 같은 함수도 사용할 수 있습니다.

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

윈도 함수에 의해 생성된 임시 파일 수를 최소화하려면 두 개의 별개의 집계가 필요할 때 동일한 결과 집합에 대한 중복을 제거합니다. 다음과 같은 쿼리를 가정하겠습니다.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
```

```
  FROM empsalary;
```

WINDOW 절을 사용하여 다음과 같이 쿼리를 다시 작성할 수 있습니다.

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

기본값으로, Aurora PostgreSQL 실행 플래너는 유사한 노드를 통합하여 연산을 복제하지 않습니다. 그러나 윈도 블록에 대한 명시적 선언을 사용하면 쿼리를 보다 쉽게 유지할 수 있습니다. 또한 중복을 방지하여 성능을 향상시킬 수 있습니다.

구체화된 뷰 및 CTAS 명령문 조사

구체화된 뷰가 새로 고쳐지면 쿼리가 실행됩니다. 이 쿼리에는 GROUP BY, ORDER BY, 또는 DISTINCT 같은 연산이 포함될 수 있습니다. 새로 고침을 하는 동안 많은 수의 임시 파일과 IO:BufFileWrite 및 IO:BufFileRead 대기 이벤트를 관찰할 수 있습니다. 마찬가지로, SELECT 문을 기반으로 테이블을 생성할 때도 마찬가지입니다. CREATE TABLE 문은 쿼리를 실행합니다. 필요한 임시 파일을 줄이려면 쿼리를 최적화하세요.

인덱스를 만들 때 pg_repack 사용

인덱스를 생성하면 엔진이 결과 세트를 정렬합니다. 테이블의 크기가 커지고 인덱싱된 열의 값이 다양해짐에 따라 임시 파일에 더 많은 공간이 필요합니다. 대부분의 경우 유지 관리 작업 메모리 영역을 수정하지 않으면 큰 테이블에 대한 임시 파일이 생성되지 않도록 할 수 없습니다. 자세한 내용은 유지 관리 작업 메모리 영역 단원을 참조하십시오.

큰 인덱스를 다시 만들 때 가능한 해결 방법은 pg_repack 도구를 사용하는 것입니다. 자세한 내용은 pg_repack 문서의 최소한의 잠금으로 PostgreSQL 데이터베이스의 테이블 재구성을 참조하세요.

테이블을 클러스터링할 때 maintenance_work_mem 증가

CLUSTER 명령은 index_name에 의해 지정된 기존 인덱스를 기반으로 하는 table_name에 의해 지정된 테이블을 클러스터링합니다. Aurora PostgreSQL은 지정된 인덱스의 순서와 일치하도록 테이블을 물리적으로 다시 만듭니다.

마그네틱 스토리지가 널리 보급되었을 때 스토리지 처리량이 제한적이었기 때문에 클러스터링이 일반적이었습니다. SSD 기반 스토리지가 일반적이므로 클러스터링은 인기가 덜합니다. 그러나 테이블을 클러스터링하는 경우에도 테이블 크기, 인덱스, 쿼리 등에 따라 성능이 약간 향상될 수 있습니다.

CLUSTER 명령을 실행하거나 IO:BufFileWrite 및 IO:BufFileRead 대기 이벤트를 관찰할 수 있는 경우, maintenance_work_mem을 조정합니다. 메모리 크기를 상당히 크게 늘립니다. 값이 높으면 엔진이 클러스터링 작업에 더 많은 메모리를 사용할 수 있음을 의미합니다.

메모리를 조정하여 IO:BufFileRead 및 IO:BufFileWrite 방지

어떤 상황에서는 메모리를 조정해야 합니다. 목표는 다음 요구 사항의 균형을 맞추는 것입니다.

- work_mem 값(작업 메모리 영역 참조)
- shared_buffers 값 디스카운트 후 남은 메모리(버퍼 풀 참조)
- 열리고 사용 중인 최대 연결은 max_connections로 제한됩니다.

작업 메모리 영역 크기 증가

경우에 따라 세션에서 사용하는 메모리를 늘리는 것이 유일한 방법입니다. 쿼리가 올바르게 작성되고 조인에 올바른 키를 사용하는 경우 work_mem 값을 증가시키는 것이 좋습니다. 자세한 내용은 작업 메모리 영역 단원을 참조하십시오.

쿼리가 생성하는 임시 파일 수를 확인하려면 log_temp_files를 0에 설정하세요. work_mem 값을 로그에서 식별된 최대값으로 늘리면 쿼리가 임시 파일을 생성하지 못하도록 합니다. 하지만 work_mem은 각 연결 또는 병렬 워커에 대해 계획 노드당 최대값을 설정합니다. 데이터베이스에 5,000개의 연결이 있고 각 연결이 256MiB 메모리를 사용하는 경우 엔진에 1.2TiB의 RAM이 필요합니다. 따라서 인스턴스의 메모리가 부족할 수 있습니다.

공유 버퍼 풀에 충분한 메모리 예약

데이터베이스는 작업 메모리 영역뿐만 아니라 공유 버퍼 풀과 같은 메모리 영역을 사용합니다. work_mem을 늘리기 전에 이러한 추가 메모리 영역의 요구 사항을 고려하는 것이 좋습니다. 버퍼 풀에 대한 자세한 내용은 버퍼 풀 섹션을 참조하세요.

예를 들어, Aurora PostgreSQL 인스턴스 클래스가 db.r5.2xlarge라고 가정합니다. 이 클래스에는 64GiB의 메모리가 있습니다. 기본적으로 메모리의 75% 가 공유 버퍼 풀에 예약되어 있습니다. 공유 메모리 영역에 할당된 양을 빼면 16,384MB가 남아 있습니다. 운영 체제와 엔진에도 메모리가 필요하므로 나머지 메모리를 작업 메모리 영역에만 할당하지 마세요.

work_mem에 할당할 수 있는 메모리는 인스턴스 클래스에 따라 달라집니다. 더 큰 인스턴스 클래스를 사용하는 경우 더 많은 메모리를 사용할 수 있습니다. 하지만 앞의 예에서는 16GiB를 초과하여 사용할 수 없습니다. 그렇지 않으면 메모리가 부족할 때 인스턴스를 사용할 수 없게 됩니다. 인스턴스를 사용할 수 없는 상태에서 복구하기 위해 Aurora PostgreSQL 자동화 서비스가 자동으로 다시 시작됩니다.

연결 수 관리

데이터베이스 인스턴스에 5,000개의 동시 연결이 있다고 가정합니다. 각 연결은 최소 4MiB의 work_mem을 사용합니다. 연결의 메모리 소비량이 많으면 성능이 저하될 수 있습니다. 다음과 같은 옵션이 있습니다.

- vCPU가 더 많은 대규모 인스턴스 클래스로 업그레이드하세요.
- 연결 프록시 또는 풀러를 사용하여 동시 데이터베이스 연결 수를 줄이세요.

프록시의 경우 Amazon RDS 프록시, PGBouncer 또는 애플리케이션에 기반한 연결 풀러를 고려하세요. 이 솔루션은 CPU 부하를 완화합니다. 또한 모든 연결에 작업 메모리 영역이 필요할 때 위험을 줄일 수 있습니다. 데이터베이스 연결 수가 적으면 work_mem 값을 늘릴 수 있습니다. 이런 방법으로 IO:BufFileRead와 IO:BufFileWrite 대기 이벤트의 발생을 줄일 수 있습니다. 또한 작업 메모리 영역을 기다리는 쿼리의 속도가 크게 향상됩니다.

IO:DataFileRead

IO:DataFileRead 이벤트는 공유 메모리에서 페이지를 사용할 수 없기 때문에 저장소에서 필요한 페이지를 읽기 위해 백엔드 프로세스에서 연결이 대기할 때 발생합니다.

주제

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

모든 쿼리 및 데이터 조작(DML) 작업은 버퍼 풀의 페이지에 액세스합니다. 읽기를 유도할 수 있는 명령문은 SELECT,UPDATE 및 DELETE가 있습니다. 예를 들어, 하나의 UPDATE는 테이블 또는 인덱스에서 페이지를 읽을 수 있습니다. 요청하거나 업데이트되는 페이지가 공유 버퍼 풀에 없는 경우 이 읽기는 IO:DataFileRead 이벤트를 생성합니다.

공유 버퍼 풀은 유한하기 때문에 채워질 수 있습니다. 이 경우 메모리에 없는 페이지에 대한 요청은 데이터베이스가 디스크에서 블록을 읽도록 강제로 합니다. 만약 IO:DataFileRead 이벤트가 자주 발생한다면 공유 버퍼 풀이 너무 작아서 워크로드를 수용할 수 없는 것일 수도 있습니다. 이 문제는 버퍼 풀에 맞지 않는 많은 수의 행을 읽는 SELECT 쿼리에서 극심하게 발생합니다. 버퍼 풀에 대한 자세한 내용은 버퍼 풀 섹션을 참조하세요.

대기 증가의 가능한 원인

IO:DataFileRead 이벤트의 일반적인 원인에는 다음이 포함됩니다.

연결 스파이크

동일한 수의 Io:DataFileRead 대기 이벤트를 생성하는 여러 연결을 찾을 수 있습니다. 이 경우 IO:DataFileRead 이벤트의 스파이크 갑작스럽고 큰 증가)가 발생할 수 있습니다.

순차 검사를 수행하는 SELECT 및 DML 문

애플리케이션에서 새 작업을 수행하고 있을 수 있습니다. 또는 새 실행 계획으로 인해 기존 작업이 변경될 수 있습니다. 이러한 경우 테이블(특히 큰 테이블)이 더 큰 seq_scan 값을 가진 테이블을 찾으세요. pg_stat_user_tables를 쿼리하여 탐색 더 많은 읽기 작업을 생성하는 쿼리를 추적하려면 pg_stat_statements 확장 프로그램을 사용하세요.

대용량 데이터 세트를 위한 CTAS 및 CREATE 인덱스

CTAS는 CREATE TABLE AS SELECT 문입니다. 대용량 데이터 세트를 소스로 사용하여 CTAS를 실행하거나 큰 테이블에 인덱스를 만드는 경우 IO:DataFileRead 이벤트가 발생할 수 있습니다. 인덱스를 만들 때 데이터베이스는 순차 스캔을 사용하여 전체 객체를 읽어야 할 수 있습니다. CTAS는 페이지가 메모리에 없을 때 IO:DataFile 리드를 생성합니다.

여러 베큠 작업자가 동시에 실행

베큠 작업자는 수동 또는 자동으로 트리거될 수 있습니다. 공격적인 베큠 전략을 채택하는 것이 좋습니다. 그러나 테이블에 업데이트되거나 삭제된 행이 많으면 IO:DataFileRead 대기가 늘어납니다. 공간을 회수한 후 IO:DataFileRead에 소비되는 베큠 시간이 감소합니다.

대용량 데이터 수집

애플리케이션이 대용량 데이터를 수집할 때 ANALYZE 연산이 더 자주 발생할 수 있습니다. ANALYZE 프로세스는 autovacuum 시작 관리자에 의해 트리거되거나 수동으로 호출될 수 있습니다.

ANALYZE 연산은 테이블의 하위 집합을 읽습니다. 스캔해야 하는 페이지 수는 default_statistics_target 값에 30을 곱하여 계산합니다. 자세한 내용은 PostgreSQL 설명

서를 참조하세요. default_statistics_target 파라미터 1에서 10,000 사이의 값을 허용합니다. 여기서 기본값은 100입니다.

리소스 부족

인스턴스 네트워크 대역폭 또는 CPU가 사용되는 경우 IO:DataFileRead 이벤트가 더 자주 발생할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

대기를 생성하는 쿼리에 대한 술어 필터 확인

IO:DataFileRead 대기 이벤트를 생성 중인 특정 쿼리를 식별한다고 가정합니다. 다음 기법을 사용하여 식별할 수 있습니다.

- 성능 개선 도우미
- pg_stat_statements 확장에서 제공하는 것과 같은 카탈로그 뷰
- 카탈로그 뷰 pg_stat_all_tables, 주기적으로 증가하는 물리적 읽기 횟수가 표시되는 경우
- pg_statio_all_tables 뷰, _read 카운터의 증가를 보여주는 경우

이 쿼리의 술어(WHERE 절)에 사용되는 필터를 결정하는 것이 좋습니다. 아래 지침을 따르세요.

- EXPLAIN 명령을 실행합니다. 출력에서 어떤 유형의 스캔이 사용되는지 식별합니다. 순차 스캔이 반드시 문제를 의미하지는 않습니다. 순차 스캔을 사용하는 쿼리는 필터를 사용하는 쿼리와 비교할 때 자연스럽게 더 많은 IO:DataFileRead 이벤트를 생성합니다.

  WHERE 절에 열거된 열이 인덱싱되었는지 확인합니다. 그렇지 않은 경우 이 열에 대한 인덱스를 만드는 것이 좋습니다. 이 접근법은 순차적 스캔을 피하고 IO:DataFileRead 이벤트를 줄입니다. 쿼리에 제한적인 필터가 있지만 순차적 스캔을 생성하는 경우 적절한 인덱스가 사용되고 있는지 평가합니다.

- 쿼리가 매우 큰 테이블에 액세스하고 있는지 확인합니다. 경우에 따라 테이블을 분할하면 성능이 향상되어 쿼리가 필요한 파티션만 읽을 수 있습니다.
- 조인 작업에서 카디널리티(총 행 수)를 조사합니다. WHERE 절을 위해 필터에 전달하는 값이 얼마나 제한적인지 확인하세요. 가능한 경우 쿼리를 조정하여 계획의 각 단계에서 전달되는 행 수를 줄입니다.

유지 보수 작업의 영향 최소화

VACUUM과 ANALYZE 같은 유지 관리 작업이 중요합니다. 이러한 유지 관리 작업과 관련된 IO:DataFileRead 대기 이벤트를 찾을 수 있으므로 끄지 않는 것이 좋습니다. 다음 방법을 사용하면 이러한 작업의 효과를 최소화할 수 있습니다.

- 사용량이 적은 시간대에 수동으로 유지 관리 작업을 실행합니다. 이 기술은 데이터베이스가 자동 연산의 임계값에 도달하지 못하도록 합니다.
- 매우 큰 테이블의 경우 테이블을 분할하는 것이 좋습니다. 이 기술은 유지보수 작업의 오버헤드를 줄여줍니다. 데이터베이스는 유지 관리가 필요한 파티션에만 액세스합니다.
- 대량의 데이터를 수집할 때는 자동 분석 기능을 사용하지 않도록 설정하는 것이 좋습니다.

다음 공식이 true이면 테이블에 대해 autovacuum 기능이 자동으로 트리거됩니다.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

뷰 pg_stat_user_tables와 카탈로그 pg_class에는 여러 개의 행이 있습니다. 한 행은 테이블의 한 행에 대응할 수 있습니다. 이 공식은 reltuples가 특정 테이블을 위한 것이라고 가정합니다. 파라미터 autovacuum_vacuum_scale_factor(기본적으로 0.20)와 autovacuum_vacuum_threshold(기본적으로 50개의 튜플)는 일반적으로 전체 인스턴스에 대해 전역으로 설정됩니다. 그러나 특정 테이블에 대해 다른 값을 설정할 수 있습니다.

주제

- 불필요한 공간을 소비하는 테이블 찾기
- 불필요한 공간을 차지하는 테이블 찾기
- Autovacuum이 가능한 테이블 찾기

불필요한 공간을 소비하는 테이블 찾기

필요 이상의 공간을 소비하는 테이블을 찾으려면 다음 쿼리를 실행합니다. `rds_superuser` 역할이 없는 데이터베이스 사용자 역할이 이 쿼리를 실행하면, 해당 사용자 역할이 읽기 권한이 있는 테이블에 관한 정보만 반환됩니다. 이 쿼리는 PostgreSQL 버전 12 이상에서 지원됩니다.

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
           SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                  ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                  ,tblpages
                  ,fillfactor
                  ,block_size
                  ,tblid
                  ,schemaname
                  ,tblname
                  ,n_dead_tup
                  ,n_live_tup
                  ,heappages
                  ,toastpages
                  ,is_na
             FROM (
                    SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                               - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
 tpl_hdr_size%ma END
```

```
                              - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
 ceil(tpl_data_size)::int%ma END
                          ) AS tpl_size
                          ,block_size - page_hdr AS size_per_block
                          ,(heappages + toastpages) AS tblpages
                          ,heappages
                          ,toastpages
                          ,reltuples
                          ,toasttuples
                          ,block_size
                          ,page_hdr
                          ,tblid
                          ,schemaname
                          ,tblname
                          ,fillfactor
                          ,is_na
                          ,n_dead_tup
                          ,n_live_tup
                         FROM (
                               SELECT  tbl.oid                       AS tblid
                                      ,ns.nspname                    AS schemaname
                                      ,tbl.relname                   AS tblname
                                      ,tbl.reltuples                 AS reltuples
                                      ,tbl.relpages                  AS heappages
                                      ,coalesce(toast.relpages, 0)   AS toastpages
                                      ,coalesce(toast.reltuples, 0)  AS toasttuples
                                      ,psat.n_dead_tup               AS n_dead_tup
                                      ,psat.n_live_tup               AS n_live_tup
                                      ,24                            AS page_hdr
                                      ,current_setting('block_size')::numeric AS
 block_size

 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                      ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                      ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END             AS tpl_hdr_size
                                      ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                   AS tpl_data_size
                                      ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)         AS is_na
                                 FROM  pg_attribute       AS att
```

```
                                 JOIN   pg_class          AS tbl    ON (att.attrelid =
 tbl.oid)
                                 JOIN   pg_stat_all_tables AS psat  ON (tbl.oid =
 psat.relid)
                                 JOIN   pg_namespace      AS ns     ON (ns.oid =
 tbl.relnamespace)
                            LEFT JOIN   pg_stats          AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                            LEFT JOIN   pg_class          AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                WHERE   att.attnum > 0
                                  AND   NOT att.attisdropped
                                  AND   tbl.relkind = 'r'
                             GROUP BY   tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                             ORDER BY   schemaname, tblname
                          ) AS s
                ) AS s2
      ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

애플리케이션에서 테이블 및 인덱스 팽창을 검사할 수 있습니다. 자세한 내용은 테이블 및 인덱스 팽창 진단 단원을 참조하십시오.

불필요한 공간을 차지하는 테이블 찾기

불필요한 공간을 차지하는 테이블을 찾으려면 다음 쿼리를 실행합니다.

```
-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
```

```
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
```

```
      SELECT
        i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
 AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
          WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
 THEN 8
          ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
          THEN 2 -- IndexTupleData size
          ELSE 2 + (( 32 + 8 - 1 ) / 8)
          -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
 index + 8 - 1 /8)
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from
 stats */
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
 nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
 AS is_na
      FROM pg_attribute AS a
        JOIN (
          SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
            idx.reltuples, idx.relpages, idx.relam,
            indrelid, indexrelid, indkey::smallint[] AS attnum,
            coalesce(substring(
              array_to_string(idx.reloptions, ' ')
               from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
          FROM pg_index
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
```

```
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

Autovacuum이 가능한 테이블 찾기

Autovacuum이 가능한 테이블을 찾으려면 다음 쿼리를 실행합니다.

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
        coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
        as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
```

```
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

많은 연결 수에 대응

Amazon CloudWatch를 모니터링할 때 `DatabaseConnections` 지표 스파이크를 확인할 수도 있습니다. 이 증가는 데이터베이스에 대한 연결 수가 증가했음을 나타냅니다. 다음과 같이 하는 것이 좋습니다.

- 애플리케이션이 각 인스턴스에서 열 수 있는 연결 수를 제한합니다. 애플리케이션에 내장된 연결 풀 기능이 있는 경우 적절한 연결 수를 설정합니다. 인스턴스의 vCPU가 효과적으로 병렬화할 수 있는 항목에 따라 숫자를 기준으로 합니다.

  애플리케이션에서 연결 풀 기능을 사용하지 않는 경우 Amazon RDS 프록시 또는 대안을 사용하는 것이 좋습니다. 이 접근 방식을 사용하면 애플리케이션이 로드 밸런서와 여러 연결을 열 수 있습니다. 그런 다음 밸런서는 데이터베이스와의 제한된 수의 연결을 열 수 있습니다. 병렬로 실행되는 연결 수가 적을수록 DB 인스턴스는 커널에서 컨텍스트 전환을 덜 수행합니다. 쿼리는 더 빠르게 진행되므로 대기 이벤트 수가 줄어듭니다. 자세한 내용은 Amazon RDS Proxy for Aurora 단원을 참조하십시오.
- 가능하면 Aurora PostgreSQL용 리더 노드와 RDS for PostgreSQL용 읽기 전용 복제본을 활용할 수 있습니다. 애플리케이션이 읽기 전용 작업을 실행하면 이러한 요청을 읽기 전용 엔드포인트로 보냅니다. 이 기술은 애플리케이션 요청을 모든 리더 노드에 분산시켜 작성기 노드의 I/O 부담을 줄입니다.
- DB 인스턴스를 확장하는 것이 좋습니다. 대용량 인스턴스 클래스는 더 많은 메모리를 제공하므로 Aurora PostgreSQL에 페이지를 저장할 수 있는 더 큰 공유 버퍼 풀이 제공됩니다. 크기가 클수록 DB

인스턴스에 연결을 처리할 수 있는 vCPU가 늘어납니다. 더 많은 vCPU가 IO:DataFileRead 대기 이벤트를 생성 중인 연산이 쓰기를 하고 있을 때 특히 유용합니다.

IO:XactSync

IO:XactSync 이벤트는 데이터베이스가 Aurora 스토리지 하위 시스템이 일반 트랜잭션의 커밋을 승인하거나 준비된 트랜잭션의 커밋 또는 롤백을 확인할 때까지 대기 중일 때 발생합니다. 준비된 트랜잭션은 PostgreSQL의 2단계 커밋 지원의 일부입니다. 또한 이 이벤트는 쿼리가 다른 트랜잭션의 커밋을 기다리고 있을 때, 특히 자동 커밋이 꺼져 있는 경우 발생할 수 있습니다. 이러한 시나리오에서는 업데이트가 아직 커밋되지 않았더라도 XactSync에서 대기 중인 것으로 보일 수 있습니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

이벤트 IO:XactSync는 인스턴스가 Aurora 스토리지 하위 시스템이 트랜잭션 데이터가 처리되었음을 확인하는 것을 기다리는 데 시간을 소비하고 있음을 나타냅니다.

대기 증가의 가능한 원인

IO:XactSync 이벤트가 정상보다 많이 나타나 성능 문제를 나타내는 경우 일반적인 원인은 다음과 같습니다.

네트워크 포화

클라이언트와 DB 인스턴스 간 트래픽 또는 스토리지 하위 시스템에 대한 트래픽은 네트워크 대역폭에 비해 너무 무거울 수 있습니다.

CPU 압력

워크로드가 많으면 Aurora 스토리지 데몬이 충분한 CPU 시간을 얻지 못하게 될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 리소스 모니터링
- CPU 확장
- 네트워크 대역폭 향상
- 커밋 수 저감

리소스 모니터링

`IO:XactSync` 이벤트가 증가한 원인을 파악하려면, 다음 지표를 확인하세요.

- `WriteThroughput`과 `CommitThroughput` - 쓰기 처리량 또는 커밋 처리량이 변경되면 워크로드가 증가할 수 있습니다.
- `WriteLatency`와 `CommitLatency` - 쓰기 지연 시간 또는 커밋 대기 시간이 변경되면 스토리지 하위 시스템이 더 많은 작업을 수행해야 한다는 메시지가 표시될 수 있습니다.
- `CPUUtilization` - 인스턴스의 CPU 사용률이 90%를 초과하면 Aurora 스토리지 데몬이 CPU에서 충분한 시간을 확보하지 못할 수 있습니다. 이 경우 I/O 성능이 저하됩니다.

지표에 대한 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 섹션을 참조하세요.

CPU 확장

CPU 부족 문제를 해결하려면 CPU 용량이 더 많은 인스턴스 유형으로 변경하는 것이 좋습니다. DB 인스턴스 클래스의 CPU 용량에 대한 자세한 내용은 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양 섹션을 참조하세요.

네트워크 대역폭 향상

인스턴스가 네트워크 대역폭 제한에 도달하는지 확인하려면 다음과 같은 다른 대기 이벤트를 확인합니다.

- `IO:DataFileRead`,`IO:BufferRead`,`IO:BufferWrite`, 및 `IO:XactWrite` - 대량의 I/O를 사용하는 쿼리는 이러한 대기 이벤트를 더 많이 생성할 수 있습니다.
- `Client:ClientRead`와 `Client:ClientWrite` - 클라이언트 통신이 많은 쿼리가 이러한 대기 이벤트를 더 많이 생성할 수 있습니다.

네트워크 대역폭이 문제인 경우 네트워크 대역폭이 더 많은 인스턴스 유형으로 변경하는 것이 좋습니다. DB 인스턴스 클래스의 네트워크 성능에 관한 자세한 내용은 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양 섹션을 참조하세요.

커밋 수 저감

커밋 수를 줄이려면 명령문을 트랜잭션 블록으로 결합하세요.

IPC:DamRecordTxAck

`IPC:DamRecordTxAck` 이벤트는 데이터베이스 활동 스트림을 사용하는 세션에서 Aurora PostgreSQL이 활동 스트림 이벤트를 생성한 다음 해당 이벤트가 지속될 때까지 기다리는 경우에 발생합니다.

주제

- 관련 엔진 버전
- 컨텍스트
- 원인
- 작업

관련 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 10.7 이상 10 버전, 11.4 이상 11 버전 및 모든 12 및 13 버전과 관련이 있습니다.

컨텍스트

동기 모드에서는 활동 스트림 이벤트의 내구성이 데이터베이스 성능보다 선호됩니다. 이벤트에 대한 내구성 쓰기를 기다리는 동안 세션은 다른 데이터베이스 작업을 차단하여 `IPC:DamRecordTxAck` 대기 이벤트를 발생시킵니다.

원인

`IPC:DamRecordTxAck` 이벤트가 최상위 대기에 나타나는 것의 가장 흔한 원인은 데이터베이스 작업 스트림(DAS) 기능이 홀리스틱 감사라는 것입니다. SQL 활동이 높을수록 기록해야 하는 활동 스트림 이벤트가 생성됩니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- SQL 문 수를 줄이거나 데이터베이스 작업 스트림을 끕니다. 이렇게 하면 내구성 있는 쓰기가 필요한 이벤트의 수가 줄어듭니다.
- 비동기 모드로 변경합니다. 이렇게 하면 `IPC:DamRecordTxAck` 대기 이벤트의 경합을 줄일 수 있습니다.

  그러나 DAS 기능은 비동기 모드에서 모든 이벤트의 내구성을 보장할 수 없습니다.

IPC:parallel 대기 이벤트

다음 `IPC:parallel wait events`는 세션이 병렬 쿼리 실행 작업과 관련된 프로세스 간 통신을 기다리고 있음을 나타냅니다.

- `IPC:BgWorkerStartup` - 병렬 워커 프로세스가 시작 시퀀스를 완료할 때까지 프로세스가 기다리고 있습니다. 병렬 쿼리 실행을 위해 워커를 초기화할 때 발생합니다.
- `IPC:BgWorkerShutdown` - 병렬 워커 프로세스가 종료 시퀀스를 완료할 때까지 프로세스가 기다리고 있습니다. 병렬 쿼리 실행의 정리 단계에서 발생합니다.
- `IPC:ExecuteGather` - 프로세스가 쿼리 실행 중에 병렬 워커 프로세스로부터 데이터 수신을 기다리고 있습니다. 리더 프로세스가 워커로부터 결과를 수집해야 할 때 발생합니다.
- `IPC:ParallelFinish` - 병렬 워커가 실행을 완료하고 최종 결과를 보고할 때까지 프로세스가 기다리고 있습니다. 병렬 쿼리 실행의 완료 단계에서 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

PostgreSQL에서 병렬 쿼리를 실행할 때는 단일 쿼리를 처리하기 위해 여러 프로세스가 함께 작동합니다. 쿼리가 병렬화에 적합한 것으로 확인되면 리더 프로세스는

`max_parallel_workers_per_gather` 파라미터 설정에 따라 하나 이상의 병렬 워커 프로세스와 조정합니다. 리더 프로세스는 워커들을 대상으로 작업을 나누고, 각 워커는 각자가 맡은 데이터를 독립적으로 처리하며, 결과는 리더 프로세스로 다시 수집됩니다.

> **Note**
> 각 병렬 워커는 전체 사용자 세션과 유사한 리소스 요구 사항이 있는 별도의 프로세스로 작동합니다. 즉, 워커가 4개인 병렬 쿼리는 비병렬 쿼리에 비해 리소스(CPU, 메모리, I/O 대역폭)를 최대 5배까지 소비할 수 있습니다. 리더 프로세스와 각 워커 프로세스 모두 자체 리소스 할당을 유지하기 때문입니다. 예를 들어, `work_mem`과 같은 설정은 각 워커에게 개별적으로 적용되어 모든 프로세스에서 총 메모리 사용량을 크게 증가시킬 수 있습니다.

병렬 쿼리 아키텍처는 세 가지 주요 구성 요소로 구성됩니다.

- 리더 프로세스: 병렬 작업을 시작하고, 워크로드를 나누고, 워커 프로세스와 조정하는 주요 프로세스입니다.
- 워커 프로세스: 쿼리의 일부를 병렬로 실행하는 백그라운드 프로세스입니다.
- 수집/수집 병합: 여러 워커 프로세스의 결과를 리더에게 다시 결합하는 작업입니다.

병렬 실행 중에 프로세스는 프로세스 간 통신(IPC) 메커니즘을 통해 서로 통신해야 합니다. 이러한 IPC 대기 이벤트는 여러 단계에서 발생합니다.

- 워커 시작: 병렬 워커가 초기화될 때
- 데이터 교환: 워커가 데이터를 처리하고 리더에게 결과를 보낼 때
- 워커 종료: 병렬 실행이 완료되고 워커가 종료될 때
- 동기화 포인트: 프로세스가 다른 프로세스와 조정하거나 다른 프로세스가 작업을 완료하기를 기다려야 할 때

이러한 대기 이벤트를 이해하는 것은 병렬 쿼리 실행과 관련된 성능 문제를 진단하는 데 매우 중요합니다. 특히 여러 병렬 쿼리가 동시에 실행될 수 있는 동시성이 높은 환경에서는 더욱 그렇습니다.

대기 증가의 가능한 원인

병렬 관련 IPC 대기 이벤트 증가에 여러 요인이 기여할 수 있습니다.

병렬 쿼리의 높은 동시성

많은 병렬 쿼리가 동시에 실행되면 리소스 경합이 발생하고 IPC 작업 대기 시간이 늘어날 수 있습니다. 특히 트랜잭션 볼륨 또는 분석 워크로드가 많은 시스템에서 흔히 발생합니다.

최적화되지 않은 병렬 쿼리 계획

쿼리 플래너가 비효율적인 병렬 계획을 선택하면 불필요한 병렬화 또는 워커 간의 작업 분산 부실이 발생할 수 있습니다. 이로 인해 특히 IPC:ExecuteGather 및 IPC:ParallelFinish 이벤트의 IPC 대기 시간이 늘어날 수 있습니다. 이러한 계획 문제는 오래된 통계 및 테이블/인덱스 팽창으로 인해 발생하는 경우가 많습니다.

병렬 워커의 빈번한 시작 및 종료

병렬 워커를 자주 시작하고 종료하는 수명이 짧은 쿼리는 `IPC:BgWorkerStartup` 및 `IPC:BgWorkerShutdown` 이벤트를 증가시킬 수 있습니다. 이는 작고 병렬화 가능한 쿼리가 많은 OLTP 워크로드에서 흔히 볼 수 있습니다.

리소스 제약 조건

CPU, 메모리 또는 I/O 용량이 제한되면 병렬 실행에서 병목 현상이 발생하여 모든 IPC 이벤트에서 대기 시간이 늘어날 수 있습니다. 예를 들어 CPU가 포화 상태인 경우 워커 프로세스를 시작하거나 할당된 작업을 처리하는 데 시간이 더 오래 걸릴 수 있습니다.

복잡한 쿼리 구조

여러 수준의 병렬 처리(예: 병렬 조인 후 병렬 집계)가 있는 쿼리는 더 복잡한 IPC 패턴을 초래하고 특히 `IPC:ExecuteGather` 이벤트의 경우 대기 시간이 증가할 수 있습니다.

대규모 결과 세트

대규모 결과 세트를 생성하는 쿼리는 리더 프로세스가 워커 프로세스로부터 결과를 수집하고 처리하는 데 더 많은 시간을 소비하므로 `IPC:ExecuteGather` 대기 시간이 늘어날 수 있습니다.

이러한 요인을 이해하면 Aurora PostgreSQL에서 병렬 쿼리 실행과 관련된 성능 문제를 진단하고 해결하는 데 도움이 될 수 있습니다.

작업

병렬 쿼리와 관련하여 대기가 발생하면 일반적으로 백엔드 프로세스가 병렬 워커 프로세스를 조정하거나 대기 중임을 의미합니다. 이러한 대기는 병렬 계획을 실행하는 동안 흔히 발생합니다. 병렬 워커 사용량을 모니터링하고, 파라미터 설정을 검토하고, 쿼리 실행 및 리소스 할당을 조정하여 이러한 대기의 영향을 조사하고 완화할 수 있습니다.

주제

- 비효율적인 병렬 처리가 있는지 쿼리 계획 분석
- 병렬 쿼리 사용량 모니터링
- 병렬 쿼리 설정 검토 및 조정
- 리소스 할당 최적화
- 연결 관리 조사
- 유지 관리 작업 검토 및 최적화
- 쿼리 계획 관리(QPM) 활용

비효율적인 병렬 처리가 있는지 쿼리 계획 분석

병렬 쿼리 실행은 종종 시스템 불안정, CPU 급증 및 예측할 수 없는 쿼리 성능 변동으로 이어질 수 있습니다. 병렬 처리가 실제로 특정 워크로드를 개선하는지를 철저히 분석하는 것이 중요합니다. EXPLAIN ANALYZE를 사용하여 병렬 쿼리 실행 계획을 검토합니다.

세션 수준에서 병렬 처리를 일시적으로 비활성화하여 계획 효율성을 비교합니다.

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

병렬 처리를 다시 활성화하고 비교합니다.

```
RESET max_parallel_workers_per_gather;
EXPLAIN ANALYZE <your_query>;
```

병렬 처리를 비활성화했을 때 결과가 더 좋거나 더 일관된 경우 SET 명령을 사용하여 세션 수준에서 특정 쿼리에 대해 병렬 처리를 비활성화하는 것이 좋습니다. 더 광범위한 영향을 미치려면 클러스터 또는 인스턴스 파라미터 그룹에서 관련 파라미터를 조정하여 인스턴스 수준에서 병렬 처리를 비활성화할 수 있습니다. 자세한 내용은 Amazon Aurora PostgreSQL parameters 섹션을 참조하세요.

병렬 쿼리 사용량 모니터링

다음 쿼리를 사용하여 병렬 쿼리 활동 및 용량에 대한 가시성을 확보합니다.

활성 병렬 워커 프로세스를 확인합니다.

```
SELECT
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

이 쿼리는 활성 병렬 워커 프로세스의 수를 보여줍니다. 값이 크면 `max_parallel_workers`가 큰 값으로 구성되어 있음을 나타낼 수 있으며 이 값을 줄이는 것이 좋습니다.

동시 병렬 쿼리를 확인합니다.

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
    leader_pid IS NOT NULL;
```

이 쿼리는 병렬 쿼리를 시작한 고유한 리더 프로세스의 수를 반환합니다. 여기서 숫자가 크면 여러 세션이 병렬 쿼리를 동시에 실행하고 있다는 뜻이며 CPU와 메모리에 대한 수요가 증가할 수 있습니다.

병렬 쿼리 설정 검토 및 조정

다음 파라미터를 검토하여 워크로드에 적절한지 확인합니다.

- `max_parallel_workers`: 모든 세션의 총 병렬 워커 수입니다.
- `max_parallel_workers_per_gather`: 쿼리당 최대 워커 수입니다.

OLAP 워크로드의 경우 이러한 값을 늘리면 성능이 향상될 수 있습니다. OLTP 워크로드의 경우 일반적으로 낮은 값이 선호됩니다.

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

리소스 할당 최적화

CPU 사용률을 모니터링하고, 사용률이 지속적으로 높고 애플리케이션이 병렬 쿼리의 이점을 누릴 수 있는 경우 vCPU 수를 조정하는 것이 좋습니다. 병렬 작업에 적절한 메모리를 사용할 수 있는지 확인합니다.

- 성능 개선 도우미 지표를 사용하여 시스템이 CPU 바운드인지 확인합니다.
- 각 병렬 워커는 자체 `work_mem`을 사용합니다. 총 메모리 사용량이 인스턴스 한도 내에 있는지 확인합니다.

병렬 쿼리는 비병렬 쿼리보다 훨씬 많은 리소스를 사용할 수 있습니다. 각 워커 프로세스는 시스템에 추가 사용자 세션과 거의 동일한 영향을 미치는 완전히 별개의 프로세스이기 때문입니다. 이 설정에 대한 값을 선택할 때, 그리고 `work_mem`과 같이 리소스 사용률을 제어하는 다른 설정을 구성할 때 이 점을 고려해야 합니다. 자세한 내용은 PostgreSQL 설명서를 참조하세요. `work_mem`과 같은 리소스 한도는 각 워커에 개별적으로 적용됩니다. 즉, 전체 사용률이 단일 프로세스보다 모든 프로세스에서 훨씬 높을 수 있습니다.

워크로드가 심하게 병렬화된 경우 vCPU를 늘리거나 메모리 파라미터를 조정하는 것이 좋습니다.

연결 관리 조사

연결 소진이 발생하는 경우 애플리케이션 연결 풀링 전략을 검토합니다. 아직 사용하지 않는 경우 애플리케이션 수준에서 연결 풀링을 구현하는 것이 좋습니다.

유지 관리 작업 검토 및 최적화

인덱스 생성 및 기타 유지 관리 작업을 조정하여 리소스 경합을 방지합니다. 이러한 작업은 사용량이 적은 시간으로 예약하는 것이 좋습니다. 사용자 쿼리 로드가 많은 기간에는 과도한 유지 관리(예: 병렬 인덱스 빌드)를 예약하지 마세요. 이러한 작업은 병렬 워커를 소비하고 정규 쿼리의 성능에 영향을 미칠 수 있습니다.

쿼리 계획 관리(QPM) 활용

Aurora PostgreSQL에서 쿼리 계획 관리(QPM) 기능은 쿼리 계획 회귀를 유발할 수 있는 데이터베이스 환경 변경과 관계없이 계획 적응성과 안정성을 보장하도록 설계되었습니다. 자세한 내용은 Aurora PostgreSQL 쿼리 계획 관리 개요 섹션을 참조하세요. QPM에서 옵티마이저에 대한 몇 가지 제어 기능을 제공합니다. QPM에서 승인된 계획을 검토하여 현재 병렬 처리 설정과 일치하는지 확인하세요. 최적화되지 않은 병렬 실행을 강제 적용할 수 있는 오래된 계획을 업데이트하거나 제거하세요.

pg_hint_plan을 사용하여 계획을 수정할 수도 있습니다. 자세한 내용은 pg_hint_plan을 사용하여 계획 수정 섹션을 참조하세요. `Parallel`이라는 힌트를 사용하여 병렬 실행을 강제 적용할 수 있습니다. 자세한 내용은 Hints for parallel plans을 참조하세요.

Lock:advisory

`Lock:advisory` 이벤트는 PostgreSQL 애플리케이션이 잠금을 사용하여 여러 세션에서 활동을 조정할 때 발생합니다.

주제

- 관련 엔진 버전
- 컨텍스트
- 원인
- 작업

관련 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 9.6 이상과 관련이 있습니다.

컨텍스트

PostgreSQL 권고 잠금은 사용자의 애플리케이션 코드에 의해 명시적으로 잠기거나 잠금 해제된 애플리케이션 수준의 협력 잠금입니다. 애플리케이션은 PostgreSQL 권고 잠금을 사용하여 여러 세션에서 활동을 조정할 수 있습니다. 일반, 객체 또는 행 레벨 잠금과는 달리 애플리케이션은 잠금 수명을 완벽하게 제어할 수 있습니다. 자세한 내용은 PostgreSQL 설명서의 권고 잠금을 참조하세요.

권고 잠금은 트랜잭션이 종료되기 전에 해제되거나 트랜잭션 간에 세션에 의해 유지될 수 있습니다. `CREATE INDEX` 문에 의해 획득한 테이블에 대한 액세스 독점 잠금과 같은 암시적 시스템 적용 잠금에는 해당되지 않습니다.

권고 잠금을 획득(잠금) 및 릴리즈(잠금 해제) 하는 데 사용되는 함수에 대한 설명은 PostgreSQL 설명서의 권고 잠금 함수을 참조하세요.

명시적인 잠금은 일반 PostgreSQL 잠금 시스템에 구현되며 `pg_locks` 시스템 뷰에서 볼 수 있습니다.

원인

이 잠금 유형은 명시적으로 사용하는 애플리케이션에 의해서만 제어됩니다. 쿼리의 일부로 각 행에 대해 획득된 권고 사항 잠금은 잠금이 급증하거나 장기 축적을 일으킬 수 있습니다.

이러한 효과는 쿼리에서 반환되는 것보다 많은 행에 대한 잠금을 획득하는 방식으로 쿼리가 실행될 때 발생합니다. 애플리케이션은 결국 모든 잠금을 해제해야 하지만 반환되지 않은 행에 잠금을 획득하면 애플리케이션에서 모든 잠금을 찾을 수 없습니다.

다음 예는 PostgreSQL 설명서의 권고 잠금에 자세히 설명되어 있습니다.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

이 예에서, LIMIT 절은 행이 이미 내부적으로 선택되고 ID 값이 잠긴 후에만 쿼리의 출력을 중지할 수 있습니다. 이는 데이터 볼륨이 증가함에 따라 플래너가 개발 중에 테스트되지 않은 다른 실행 계획을 선택하게 되면 갑자기 발생할 수 있습니다. 이 경우 빌드업은 애플리케이션이 잠긴 모든 ID 값에 대해 pg_advisory_unlock을 명시적으로 호출하기 때문에 발생합니다. 그러나 이 경우 반환되지 않은 행에서 획득한 잠금 집합을 찾을 수 없습니다. 잠금은 세션 수준에서 획득되기 때문에 트랜잭션이 끝날 때 자동으로 해제되지 않습니다.

차단된 잠금 시도에서 스파이크가 발생할 수 있는 또 다른 원인은 의도하지 않은 충돌입니다. 이러한 충돌에서 애플리케이션의 관련 없는 부분은 실수로 동일한 잠금 ID 공간을 공유합니다.

작업

권고 잠금의 애플리리케이션 사용을 검토하고 애플리케이션 흐름에서 각 유형의 권고 잠금을 획득하고 해제하는 위치와 시기를 자세히 검토합니다.

세션이 너무 많은 잠금을 획득하고 있는지 아니면 장기 실행 세션에서 잠금을 조기에 해제하지 않아 잠금이 느리게 축적되는지 확인합니다. pg_terminate_backend(pid)를 사용하는 세션을 종료하여 세션 수준 잠금의 느린 축적을 수정할 수 있습니다.

권고 잠금을 대기하는 클라이언트는 pg_stat_activity에서 wait_event_type=Lock 및 wait_event=advisory와 표시됩니다. locktype=advisory 및 granted=f를 탐색하며 동일한 pid를 위해 pg_locks 시스템 뷰를 쿼리하여 특정 잠금 값을 얻을 수 있습니다.

그런 다음, 예에 표시된 대로 granted=t를 가진 동일한 명시적인 잠금을 위해 pg_locks를 쿼리하여 차단 세션을 식별할 수 있습니다.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
```

```
        concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
blocked_wait_event,
        concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
blocking_wait_event,
        blocked_activity.state AS blocked_state,
        blocking_activity.state AS blocking_state,
        blocked_locks.locktype AS blocked_locktype,
        blocking_locks.locktype AS blocking_locktype,
        blocked_activity.query AS blocked_statement,
        blocking_activity.query AS blocking_statement
   FROM pg_catalog.pg_locks blocked_locks
   JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
   JOIN pg_catalog.pg_locks blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
   JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

모든 명시적인 잠금 API 함수에는 두 개의 인수 집합이 있으며, 하나의 `bigint` 인수 또는 두 개의 `integer` 인수입니다.

- 하나의 API 함수의 경우 `bigint` 인수, 상위 32비트는`pg_locks.classid` 인수, 더 낮은 32비트는 `pg_locks.objid` 인수입니다.
- 두 개가 포함된 API 함수의 경우 `integer` 인수, 첫 번째 인수는 `pg_locks.classid`, 두 번째 인수는 `pg_locks.objid`입니다.

`pg_locks.objsubid` 값은 어떤 API 양식을 사용했는지를 나타냅니다. 1은 하나의 `bigint` 인수를 의미하며, 2는 두 개의 `integer` 인수를 의미합니다.

Lock:extend

Lock:extend 이벤트는 백엔드 프로세스가 릴레이션을 확장하기 위해 릴레이션을 잠그기를 기다리는 동안 다른 프로세스에서 동일한 목적으로 해당 릴레이션에 대한 잠금이 있는 경우에 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

이벤트 Lock:extend는 백엔드 프로세스가 관계를 확장하는 동안 다른 백엔드 프로세스가 잠금을 유지하는 관계를 확장하기 위해 대기 중임을 나타냅니다. 한 번에 하나의 프로세스만 관계를 확장할 수 있기 때문에 시스템은 Lock:extend 대기 이벤트를 생성합니다. INSERT,COPY, 및 UPDATE 연산은 이 이벤트를 생성할 수 있습니다.

대기 증가의 가능한 원인

Lock:extend 이벤트가 정상보다 많이 나타나 성능 문제를 나타내는 경우 일반적인 원인은 다음과 같습니다.

동일한 테이블에 대한 동시 삽입 또는 업데이트 서지

동일한 테이블에 삽입하거나 업데이트하는 쿼리의 동시 세션 수가 증가할 수 있습니다.

네트워크 대역폭 부족

DB 인스턴스의 네트워크 대역폭이 현재 워크로드의 스토리지 통신 요구 사항에 따라 충분하지 않을 수 있습니다. 이로 인해 Lock:extend 이벤트에서 스토리지 지연 시간이 증가할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 동일한 관계식으로 동시 삽입 및 업데이트 감소
- 네트워크 대역폭 향상

동일한 관계식으로 동시 삽입 및 업데이트 감소

먼저, 증가가 있는지 확인합니다. tup_inserted와 tup_updated 지표 및 이 대기 이벤트의 증가가 수반됩니다. 그렇다면 삽입 및 업데이트 작업에 대한 경합이 높은 관계식을 확인합니다. 이를 확인하려면 n_tup_ins 및 n_tup_upd 필드의 값을 위한 pg_stat_all_tables 뷰를 쿼리하세요. pg_stat_all_tables 뷰에 대한 자세한 내용은 PostgreSQL 설명서의 pg_stat_statements를 참조하세요.

차단 및 차단된 쿼리에 대한 자세한 정보를 위해서 다음 예와 같이 pg_stat_activity를 쿼리하세요.

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |             query            | blocking_id |
             blocking_query                           | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

Lock:extend 이벤트 증가에 기여하는 관계를 파악한 후 다음 기술을 사용하여 경합을 줄입니다.

- 파티셔닝을 사용하여 동일한 테이블에 대한 경합을 줄일 수 있는지 확인합니다. 삽입되거나 업데이트된 튜플을 다른 파티션으로 분리하면 경합을 줄일 수 있습니다. 파티셔닝에 대한 자세한 내용은 pg_partman 확장자를 사용하여 PostgreSQL 파티션 관리하기 섹션을 참조하세요.
- 대기 이벤트가 주로 업데이트 활동으로 인한 경우 관계식의 채우기 요소 값을 줄이는 것이 좋습니다. 이렇게 하면 업데이트 중에 새 블록에 대한 요청을 줄일 수 있습니다. 필팩터는 테이블 페이지를 포장하기 위한 최대 공간을 결정하는 테이블의 저장 파라미터입니다. 이 값은 페이지의 전체 공간의 백분율로 표시됩니다. 필팩터 파라미터에 대한 자세한 내용은 PostgreSQL 설명서의 테이블 생성을 참조하세요.

> **Important**
>
> 필팩터를 변경하는 경우 이 값을 변경하면 워크로드에 따라 성능에 부정적인 영향을 줄 수 있으므로 시스템을 테스트하는 것이 좋습니다.

네트워크 대역폭 향상

쓰기 지연 시간이 증가하는지 확인하려면 CloudWatch의 `WriteLatency` 지표를 확인하세요. 그렇다면, `WriteThroughput`과 `ReadThroughput` DB Amazon CloudWatch 지표를 사용하여 DB 클러스터의 스토리지 관련 트래픽을 모니터링하세요. 이러한 지표는 네트워크 대역폭이 워크로드의 스토리지 활동에 충분한지 확인하는 데 도움이 될 수 있습니다.

네트워크 대역폭이 충분하지 않으면 늘리세요. 만약 클라이언트 또는 DB 인스턴스가 네트워크 대역폭 제한에 도달했다면. 대역폭을 늘리는 유일한 방법은 DB 인스턴스 크기를 늘리는 것입니다.

CloudWatch 지표에 대한 자세한 내용은 Amazon Aurora에 대한 Amazon CloudWatch 지표 섹션을 참조하세요. 각 DB 인스턴스 클래스에 대한 네트워크 성능에 관한 자세한 내용은 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양 섹션을 참조하세요.

Lock:Relation

`Lock:Relation` 이벤트는 쿼리가 현재 다른 트랜잭션에 의해 잠긴 테이블 또는 뷰(관계식)에 대한 잠금을 얻기 위해 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인

- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

대부분의 PostgreSQL 명령은 암시적으로 잠금을 사용하여 테이블의 데이터에 대한 동시 액세스를 제어합니다. 애플리케이션 코드에서 `LOCK` 명령과 함께 이러한 잠금을 명시적으로 사용할 수도 있습니다. 많은 잠금 모드는 서로 호환되지 않으며 동일한 객체에 액세스하려고 할 때 트랜잭션을 차단할 수 있습니다. 이 경우 Aurora PostgreSQL은 `Lock:Relation` 이벤트를 생성합니다. 다음은 몇 가지 일반적인 예입니다.

- `ACCESS EXCLUSIVE`와 같은 독점 잠금은 모든 동시 액세스를 차단할 수 있습니다. 데이터 정의 언어(DDL) 작업(예: `DROP TABLE`,`TRUNCATE`,`VACUUM FULL`, 및`CLUSTER`)은 명시적으로 `ACCESS EXCLUSIVE` 잠금을 획득합니다. 또한 `ACCESS EXCLUSIVE`는 모드를 명시적으로 식별하지 않는 `LOCK TABLE` 문을 위한 기본 잠금 모드입니다.
- 데이터 조작 언어(DML) 명령문 `UPDATE`,`DELETE`, 및 `INSERT`와 충돌하는 테이블에 `ROW EXCLUSIVE` 잠금을 획득하는 `CREATE INDEX (without CONCURRENT)`를 사용하세요.

테이블 수준 잠금 및 충돌하는 잠금 모드에 대한 자세한 내용은 PostgreSQL 설명서의 명시적 잠금을 참조하세요.

쿼리 및 트랜잭션 차단은 일반적으로 다음 중 한 가지 방법으로 잠금 해제합니다.

- 쿼리 차단 - 애플리케이션이 쿼리를 취소하거나 사용자가 프로세스를 종료할 수 있습니다. 세션의 명령문 시간 초과 또는 교착 상태 감지 메커니즘으로 인해 엔진이 쿼리를 강제로 종료할 수도 있습니다.
- 트랜잭션 차단 - 트랜잭션이 `ROLLBACK`이나 `COMMIT` 문을 실행할 때 차단을 중지합니다. 롤백은 클라이언트 또는 네트워크 문제로 세션의 연결이 끊어지거나 종료된 경우에도 자동으로 수행됩니다. 세션은 데이터베이스 엔진이 종료될 때, 시스템의 메모리가 부족할 때 종료될 수 있습니다.

대기 증가의 가능한 원인

`Lock:Relation` 이벤트가 평소보다 더 자주 발생하면 성능 문제를 나타낼 수 있습니다. 일반적인 원인은 다음과 같습니다.

테이블 잠금 충돌로 인한 동시 세션 증가

잠금 모드가 충돌하는 동일한 테이블을 하거나 잠그는 쿼리의 동시 세션 수가 증가할 수 있습니다.

유지 관리 작업

VACUUM과 ANALYZE 같은 상태 유지 관리 작업은 충돌하는 잠금 수를 크게 늘릴 수 있습니다. VACUUM FULL은 하나의 ACCESS EXCLUSIVE 잠금을, ANALYZE는 하나의 SHARE UPDATE EXCLUSIVE 잠금을 획득합니다. 두 가지 유형의 잠금은 모두 Lock:Relation 대기 이벤트를 발생시킬 수 있습니다. 구체화된 뷰 새로 고침과 같은 애플리케이션 데이터 유지 관리 작업은 차단된 질의 및 트랜잭션을 증가시킬 수도 있습니다.

리더 인스턴스 잠금

라이터와 리더가 보유한 관계 잠금 간에 충돌이 있을 수 있습니다. 현재 ACCESS EXCLUSIVE 관계 잠금만 리더 인스턴스에 복제됩니다. 그러나 ACCESS EXCLUSIVE 관계 잠금은 리더가 보유한 모든 ACCESS SHARE 관계 잠금과 충돌합니다. 이로 인해 리더에서 잠금 관계 대기 이벤트가 증가할 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- SQL 문 차단의 영향 감소
- 유지 보수 작업의 영향 최소화
- 리더 잠금 확인

SQL 문 차단의 영향 감소

SQL 문 차단의 영향을 줄이려면 가능한 경우 애플리케이션 코드를 수정하세요. 다음은 블록을 줄이기 위한 두 가지 일반적인 기술입니다.

- NOWAIT 옵션 사용 - SELECT나 LOCK 문 같은 일부 SQL 명령은 이 옵션을 지원합니다. NOWAIT 지시어는 잠금을 즉시 획득할 수 없는 경우 잠금 요청 쿼리를 취소합니다. 이 기술은 차단 세션이 그 뒤에 차단된 세션이 쌓이지 않게 하는 데 도움이 될 수 있습니다.

  예: 트랜잭션 A가 트랜잭션 B가 보유한 잠금을 기다리고 있다고 가정합니다. 이제 B가 트랜잭션 C에 의해 잠긴 테이블에 대한 잠금을 요청하면 트랜잭션 C가 완료될 때까지 트랜잭션 A가 차단될 수 있

습니다. 그러나 트랜잭션 B가 NOWAIT를 사용하는 경우 C에서 잠금을 요청하면 빠르게 실패하고 트랜잭션 A가 계속해서 기다릴 필요가 없도록 할 수 있습니다.

- SET lock_timeout 사용 - SQL 문이 관계식 잠금을 획득하기 위해 대기하는 시간을 제한하는 lock_timeout 값을 설정하세요. 지정된 제한 시간 내에 잠금을 획득하지 않으면 잠금을 요청하는 트랜잭션이 취소됩니다. 세션 수준에서 이 값을 설정합니다.

유지 보수 작업의 영향 최소화

VACUUM과 ANALYZE 같은 유지 관리 작업이 중요합니다. 이러한 유지 관리 작업과 관련된 Lock:Relation 대기 이벤트를 찾을 수 있으므로 끄지 않는 것이 좋습니다. 다음 방법을 사용하면 이러한 작업의 효과를 최소화할 수 있습니다.

- 사용량이 적은 시간대에 수동으로 유지 관리 작업을 실행합니다.
- Autovacuum 작업으로 인한 Lock:Relation 대기를 줄이려면 필요한 autovacuum 튜닝을 수행하세요. Autovacuum 튜닝에 대한 자세한 내용은 Amazon RDS 사용 설명서의 Amazon RDS에서 PostgreSQL Autovacuum 사용을 참조하세요.

리더 잠금 확인

라이터 및 리더의 동시 세션이 서로를 차단하는 잠금을 유지하는 방법을 확인할 수 있습니다. 이를 수행하는 한 가지 방법은 잠금 유형 및 관계를 반환하는 쿼리를 실행하는 것입니다. 테이블에서 두 개의 동시 세션, 즉 라이터 세션과 리더 세션에 대한 쿼리 시퀀스를 찾을 수 있습니다.

재생 프로세스는 리더 쿼리를 취소하기 전에 max_standby_streaming_delay의 기간 동안 기다립니다. 예에서 볼 수 있듯이 100ms의 잠금 시간 제한은 기본값 max_standby_streaming_delay인 30초보다 훨씬 낮습니다. 문제가 발생하기 전에 잠금 시간이 초과됩니다.

시퀀스 이벤트	세션	명령 또는 출력

지정된 값을 사용하여 READER라는 환경 변수를 설정하고 이 엔드포인트를 사용하여 DB 인스턴스에 연결을 시도합니다.	리더 세션	CLI 명령:
		`export READER=aurorapg2.12345678910.us-west-1.rds.amazonaws.com`
		
		`psql -h $READER`

시퀀스 이벤트	세션	명령 또는 출력

		출력:
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

WRITER라는 환경 변수를 설정하고 이 엔드포인트 를 사용하여 DB 인스턴스에 연결을 시도합니다.	라이터 세션	CLI 명령:
		`export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $WRITER`
		출력:
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

라이터 세션은 라이터 인스턴스에서 table t1을 생성합니다.	라이터 세션	PostgreSQL 쿼리:
		`postgres=>  CREATE TABLE t1(b integer);`
		`CREATE TABLE`

라이터에 충돌하는 쿼리가 없는 경우 라이터에서 ACCESS EXCLUSIVE 잠금을 즉시 획득합니다.	라이터 세션	`ACCESS EXCLUSIVE` 잠금 활성화됨

리더 세션은 잠금 시간 초과 간격을 100밀리초로 설정합니다.	리더 세션	PostgreSQL 쿼리:
		`postgres=>  SET lock_timeout=100;`
		`SET`

<table>
<tr><th>시퀀스 이벤트</th><th>세션</th><th>명령 또는 출력</th></tr>
<tr><td>리더 세션이 리더 인스턴스의 table t1에서 데이터를 읽으려고 시도합니다.</td><td>리더 세션</td><td>PostgreSQL 쿼리:
<pre>postgres=> SELECT *
 FROM t1;</pre>샘플 출력:
<pre>b

(0 rows)</pre></td></tr>
<tr><td>라이터 세션이 t1을 삭제합니다.</td><td>라이터 세션</td><td>PostgreSQL 쿼리:
<pre>postgres=> BEGIN;
BEGIN
postgres=> DROP TABLE
 t1;
DROP TABLE
postgres=></pre></td></tr>
<tr><td>쿼리 시간이 초과되고 리더에서 취소됩니다.</td><td>리더 세션</td><td>PostgreSQL 쿼리:
<pre>postgres=> SELECT *
 FROM t1;</pre>샘플 출력:
<pre>ERROR: canceling
 statement due to lock
 timeout
LINE 1: SELECT * FROM
 t1;
                      ^</pre></td></tr>
</table>

시퀀스 이벤트	세션	명령 또는 출력

리더 세션이 pg_locks와 pg_stat_activity 를 쿼리하여 오류의 원인을 확인합니다.	리더 세션	PostgreSQL 쿼리:
		`postgres=> SELECT locktype, relation, mode, backend_type`
		`postgres=> FROM pg_locks l, pg_stat_activity t1`
		`postgres=> WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid;`

결과는 aurora wal replay 프로세스가 table t1의 ACCESS EXCLUSIVE 잠금을 유지하고 있습니다.	리더 세션	쿼리 결과:
		`locktype | relation | mode | backend_type`
		`----------+----------+---------------------+-------------------`
		`relation | 68628525 | AccessExclusiveLock | aurora wal replay`
		`(1 row)`

Lock:transactionid

Lock:transactionid 이벤트는 트랜잭션이 행 수준 잠금을 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

이벤트 `Lock:transactionid` 트랜잭션이 동시에 실행 중인 트랜잭션에 이미 부여된 행 수준 잠금을 획득하려고 할 때 발생합니다. `Lock:transactionid` 대기 이벤트를 보여주는 세션이 이 잠금으로 인해 차단되었습니다. 차단 트랜잭션이 종료된 후 `COMMIT` 또는`ROLLBACK` 문, 차단된 트랜잭션을 진행할 수 있습니다.

Aurora PostgreSQL의 다중 버전 동시성 제어 의미는 리더가 라이터를 차단하지 않으며 라이터는 리더를 차단하지 않는다는 것을 보장합니다. 행 수준 충돌이 발생하려면 차단 및 차단된 트랜잭션이 다음 유형의 충돌하는 명령문을 실행해야 합니다.

- `UPDATE`
- `SELECT … FOR UPDATE`
- `SELECT … FOR KEY SHARE`

명령문 `SELECT … FOR KEY SHARE`는 특별한 경우입니다. 데이터베이스가 `FOR KEY SHARE` 절을 사용해 참조 무결성의 성능을 최적화합니다. 행의 행 수준 잠금은 행을 참조하는 다른 테이블에서 `INSERT`, `UPDATE`, `DELETE` 명령을 차단할 수 있습니다.

대기 증가의 가능한 원인

이 이벤트가 정상보다 많이 나타나는 경우 일반적으로 원인은 다음 조건과 결합된 `UPDATE`, `SELECT … FOR UPDATE`, 또는 `SELECT … FOR KEY SHARE` 문입니다.

주제

- 높은 동시성
- 트랜잭션의 유휴 상태
- 장기 실행 트랜잭션

높은 동시성

Aurora PostgreSQL 세분화된 행 수준 잠금 의미를 사용할 수 있습니다. 다음 조건이 충족되면 행 수준 충돌 확률이 높아집니다.

- 동일한 행에 대해 높은 동시 워크로드가 경합됩니다.
- 동시성이 증가합니다.

트랜잭션의 유휴 상태

가끔 pg_stat_activity.state 열에 idle in transaction 값이 표시됩니다. 이 값은 트랜잭션을 시작했지만 아직 COMMIT 또는 ROLLBACK을 실행하지 않은 세션에 대해 표시됩니다. 만약 pg_stat_activity.state 값이 active가 아니라면, pg_stat_activity에 표시된 쿼리는 실행을 마친 가장 최근의 쿼리입니다. 차단 세션은 열려 있는 트랜잭션이 잠금을 유지하고 있기 때문에 쿼리를 능동적으로 처리하지 않습니다.

유휴 트랜잭션이 행 수준 잠금을 획득한 경우 다른 세션에서 잠금을 획득하지 못할 수 있습니다. 이 조건으로 인해 Lock:transactionid 대기 이벤트가 자주 발생합니다. 문제를 진단하려면 pg_stat_activity와 pg_locks의 출력을 검사하세요.

장기 실행 트랜잭션

오랫동안 실행되는 트랜잭션은 오랜 시간 동안 잠금을 얻습니다. 이러한 장기 잠금은 다른 트랜잭션의 실행을 차단할 수 있습니다.

작업

행 잠금은 UPDATE, SELECT … FOR UPDATE, 또는 SELECT … FOR KEY SHARE 문 사이의 충돌입니다. 솔루션을 시도하기 전에 이러한 명령문이 동일한 행에서 실행되는 시점을 확인하세요. 다음 섹션에 설명된 전략을 선택하려면 이 정보를 사용하세요.

주제

- [동시성에 대응](#)
- [유휴 트랜잭션에 대응](#)
- [장기 실행 트랜잭션에 대응](#)

동시성에 대응

동시성이 문제가 되는 경우 다음 기술 중 하나를 시도해 보세요.

- 애플리케이션의 동시성을 낮춥니다. 예를 들어 활성 세션 수를 줄일 수 있습니다.
- 연결 풀을 구현합니다. RDS 프록시로 연결을 풀링하는 방법에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 섹션을 참조하세요.

- UPDATE 문과 SELECT … FOR UPDATE 문의 경쟁 방지를 위한 애플리케이션 또는 데이터 모델을 설계하세요. SELECT … FOR KEY SHARE 문으로 액세스하는 외래 키 수를 줄일 수도 있습니다.

유휴 트랜잭션에 대응

pg_stat_activity.state가 idle in transaction을 나타낸다면, 다음 전략을 시도해 보세요.

- 가능하면 자동 커밋을 켭니다. 이 접근 방식은 트랜잭션이 COMMIT이나 ROLLBACK을 대기하는 동안 다른 트랜잭션을 차단하는 것을 방지합니다.
- COMMIT, ROLLBACK, 또는 END가 누락된 코드 경로 검색
- 애플리케이션의 예외 처리 논리에 항상 유효한 end of transaction으로 향하는 경로가 있는지 확인하세요.
- COMMIT 및 ROLLBACK과의 트랜잭션을 종료한 후 애플리케이션이 쿼리 결과를 처리하는지 확인하세요.

장기 실행 트랜잭션에 대응

장기 실행 트랜잭션으로 인해 Lock:transactionid가 자주 발생하는 경우 다음 전략을 시도해 보세요.

- 장기 실행 트랜잭션에서 행 잠금을 차단합니다.
- 가능하면 자동 커밋을 구현하여 쿼리 길이를 제한합니다.

Lock:tuple

Lock:tuple 이벤트는 백엔드 프로세스가 튜플에 대한 잠금 획득을 대기 중일 때 발생합니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

이벤트 Lock:tuple은 다른 백엔드가 동일한 튜플에서 충돌하는 잠금을 보유하는 동안 백엔드가 튜플에 대한 잠금을 얻기 위해 대기 중임을 나타냅니다. 다음 테이블에서는 세션이 Lock:tuple 이벤트를 생성하는 시나리오를 가정합니다.

Time	세션 1	세션 2	세션 3
t1	트랜잭션을 시작합니다.		
t2	1행을 업데이트합니다.		
t3		1행을 업데이트합니다. 세션은 튜플에 대한 배타적 잠금을 획득한 다음 세션 1이 커밋하거나 롤백하여 잠금을 해제할 때까지 기다립니다.	
t4			1행을 업데이트합니다. 세션은 세션 2가 튜플에서 배타적 잠금을 해제할 때까지 기다립니다.

또는 벤치마킹 도구 pgbench를 사용하여 이 대기 이벤트를 시뮬레이션할 수 있습니다. 테이블의 동일한 행을 사용자 정의 SQL 파일로 업데이트하도록 많은 수의 동시 세션을 구성합니다.

충돌하는 잠금 모드에 대한 자세한 내용은 PostgreSQL 설명서의 [명시적 잠금](#)을 참조하세요. pgbench에 관한 더 자세한 내용은 PostgreSQL 설명서의 [pgbench](#) 섹션을 참조하세요.

대기 증가의 가능한 원인

이 이벤트가 정상보다 많이 발생해 성능 문제를 일으킬 수 있는 경우 일반적인 원인은 다음과 같습니다.

- 많은 수의 동시 세션이 UPDATE 또는 DELETE 문을 실행하여 동일한 튜플에 대해 충돌하는 잠금을 얻으려고 시도합니다.
- 높은 동시 세션이 FOR UPDATE 또는 FOR NO KEY UPDATE 잠금 모드를 통해 SELECT 문을 실행하고 있습니다.

- 다양한 요인으로 인해 애플리케이션이나 연결 풀이 더 많은 세션을 열어 동일한 작업을 실행할 수 있습니다. 새 세션이 동일한 행을 수정하려고 하면 DB 로드가 급증할 수 있으며, `Lock:tuple`이 표시될 수 있습니다.

자세한 내용은 PostgreSQL 설명서의 [행 수준 잠금]을 참조하세요.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 애플리케이션 로직 조사
- 차단 세션 찾기
- 높은 동시성 저감
- 병목 현상 해결

애플리케이션 로직 조사

차단 세션이 오랜 시간 동안 `idle in transaction` 상태인지 확인하세요. 그렇다면 차단 세션을 단기 솔루션으로 종료하는 것이 좋습니다. `pg_terminate_backend` 함수를 사용할 수 있습니다. 이 함수에 대한 자세한 내용은 PostgreSQL 설명서의 서버 신호 전송 함수를 참조하세요.

장기 솔루션의 경우 다음을 수행합니다.

- 애플리케이션 로직을 조정합니다.
- `idle_in_transaction_session_timeout` 파라미터를 사용합니다. 이 파라미터는 지정된 시간보다 오랫동안 유휴 상태인 열린 트랜잭션으로 세션을 종료합니다. 자세한 내용은 PostgreSQL 설명서의 클라이언트 인증을 참조하세요.
- autocommit을 최대한 많이 사용합니다. 자세한 내용은 PostgreSQL 설명서의 AUTOCOMMIT 설정을 참조하세요.

차단 세션 찾기

`Lock:tuple` 대기 이벤트가 발생하는 동안 어떤 잠금이 서로 의존하는지 파악하여 차단 및 차단된 세션을 식별합니다. 자세한 내용은 PostgreSQL 위키의 잠금 종속성 정보를 참조하세요. 과거의 `Lock:tuple` 이벤트를 분석하려면, Aurora 함수 `aurora_stat_backend_waits`를 사용하세요.

다음 예에서는 `tuple` 필터링과 `wait_time` 정렬을 통해 모든 세션을 쿼리합니다.

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
-------+---------+----------+------------------------------------------------
+-------------------+--------------------+---------------+-----------+------------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

높은 동시성 저감

`Lock:tuple` 이벤트는 지속적으로 발생할 수 있으며, 특히 작업 부하가 많은 시간에 발생할 수 있습니다. 이 경우 매우 바쁜 행에 대해 높은 동시성을 줄이는 것이 좋습니다. 종종 몇 개의 행만 대기열이나 불리언 로직을 제어하므로 이러한 행을 매우 바쁘게 만듭니다.

비즈니스 요구 사항, 애플리케이션 로직 및 워크로드 유형에 따라 다양한 접근 방식을 사용하여 동시성을 줄일 수 있습니다. 예를 들면, 다음을 수행할 수 있습니다.

- 테이블 및 데이터 로직을 재설계하여 높은 동시성을 줄입니다.
- 행 수준에서 높은 동시성을 줄이기 위해 애플리케이션 로직을 변경합니다.
- 행 수준 잠금으로 쿼리를 활용하고 재설계합니다.
- `NOWAIT` 절을 사용해 연산을 재시도합니다.
- 낙관적 및 하이브리드 잠금 로직 동시성 제어를 사용하는 것이 좋습니다.
- 데이터베이스 격리 수준을 변경하는 것이 좋습니다.

병목 현상 해결

`Lock:tuple`은 CPU 부족 또는 Amazon EBS 대역폭의 최대 사용량과 같은 병목 현상이 발생시킬 수 있습니다. 병목 현상을 줄이려면 다음 방법을 고려하세요.

- 인스턴스 클래스 유형을 확장하세요.
- 리소스 집약적인 쿼리를 최적화하세요.
- 애플리케이션 로직을 변경하세요.
- 거의 액세스하지 않는 데이터를 아카이브하세요.

LWLock:buffer_content (BufferContent)

`LWLock:buffer_content` 이벤트는 다른 세션에서 특정 데이터 페이지에 대해 쓰기 잠금을 설정한 동안 세션에서 메모리 내 해당 페이지 읽기 또는 쓰기를 대기 중일 때 발생합니다. Aurora PostgreSQL 13 이상에서는 `BufferContent` 대기 이벤트가 호출됩니다.

주제

- [지원되는 엔진 버전]
- [컨텍스트]
- [대기 증가의 가능한 원인]

- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

데이터를 읽거나 조작하기 위해 PostgreSQL 공유 메모리 버퍼를 통해 데이터에 액세스합니다. 버퍼에서 읽기 위해 프로세스는 공유 모드에서 버퍼 콘텐츠에 대한 경량 잠금(LWLock) 을 가져옵니다. 버퍼에 쓰려면 배타적 모드에서 해당 잠금을 가져옵니다. 공유 잠금을 사용하면 다른 프로세스가 동시에 해당 콘텐츠에 대한 공유 잠금을 획득할 수 있습니다. 배타적 잠금은 다른 프로세스에서 모든 유형의 잠금을 얻지 못하게 합니다.

`LWLock:buffer_content(BufferContent)` 이벤트는 여러 프로세스가 특정 버퍼의 내용을 잠그려고 시도하고 있음을 나타냅니다.

대기 증가의 가능한 원인

`LWLock:buffer_content(BufferContent)` 이벤트가 정상보다 많이 나타나 성능 문제를 나타내는 경우 일반적인 원인은 다음과 같습니다.

동일한 데이터에 대한 동시 업데이트 증가

동일한 테이블에 삽입하거나 업데이트하는 쿼리의 동시 세션 수가 증가할 수 있습니다. 이 경합은 인덱스가 많은 테이블에서 더 두드러질 수 있습니다.

워크로드 데이터가 메모리에 없습니다.

활성 워크로드에서 처리 중인 데이터가 메모리에 없으면 이러한 대기 이벤트가 증가할 수 있습니다. 이 효과는 잠금을 유지하는 프로세스가 디스크 I/O 작업을 수행하는 동안 더 오래 유지할 수 있기 때문입니다.

외래 키 제약 조건을 과도하게 사용

외래 키 제약 조건은 프로세스가 버퍼 콘텐츠 잠금에 보관하는 시간을 늘릴 수 있습니다. 이 효과는 해당 키가 업데이트되는 동안 읽기 작업에서 참조된 키에 대해 공유 버퍼 콘텐츠 잠금이 필요하기 때문입니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다. Amazon RDS 성능 개선 도우미를 사용하거나 `pg_stat_activity` 뷰를 쿼리하여 `LWLock:buffer_content`(`BufferContent`) 이벤트를 식별할 수 있습니다.

주제

- [인메모리 효율성 향상]
- [외래 키 제약 조건 사용 감소]
- [사용되지 않는 인덱스 삭제]

인메모리 효율성 향상

활성 워크로드 데이터가 메모리에 있을 확률을 높이려면 테이블을 분할하거나 인스턴스 클래스를 확장하세요. DB 인스턴스 클래스에 대한 자세한 내용은 [Amazon Aurora DB 인스턴스 클래스] 섹션을 참조하세요.

외래 키 제약 조건 사용 감소

외래 키 제약 조건을 사용을 위해 많은 수의 `LWLock:buffer_content`(`BufferContent`) 대기 이벤트를 경험하는 워크로드를 조사합니다. 불필요한 외래 키 제약 조건을 제거합니다.

사용되지 않는 인덱스 삭제

많은 수의 `LWLock:buffer_content`(`BufferContent`) 대기 이벤트를 경험하는 워크로드를 위해 사용하지 않는 인덱스를 식별하고 제거합니다.

LWLock:buffer_mapping

이 이벤트는 세션이 데이터 블록을 공유 버퍼 풀의 버퍼와 연결하기 위해 대기 중일 때 발생합니다.

> **Note**
>
> 이 이벤트는 Aurora PostgreSQL 버전 12 이하에서는 `LWLock:buffer_mapping`으로 나타나며, 13 버전 이상에서는 `LWLock:BufferMapping`으로 표시됩니다.

주제

- [지원되는 엔진 버전]

- 컨텍스트
- 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 9.6 이상에서 지원됩니다.

컨텍스트

공유 버퍼 풀은 애플리케이션에서 사용 중이거나 사용 중인 모든 페이지를 보관하는 Aurora PostgreSQL 메모리 영역입니다. 프로세스에 페이지가 필요한 경우 페이지를 공유 버퍼 풀로 읽습니다. `shared_buffers` 파라미터는 공유 버퍼 크기를 설정하고 테이블 및 인덱스 페이지를 저장할 메모리 영역을 예약합니다. 이 파라미터를 변경하는 경우 데이터베이스를 다시 시작해야 합니다. 자세한 내용은 공유 버퍼 단원을 참조하십시오.

`LWLock:buffer_mapping` 대기 이벤트는 다음 시나리오에서 발생합니다.

- 프로세스가 버퍼 테이블에서 페이지를 검색하고 공유 버퍼 매핑 잠금을 획득합니다.
- 프로세스가 페이지를 버퍼 풀로 로드하고 배타적 버퍼 매핑 잠금을 획득합니다.
- 프로세스가 페이지를 버퍼 풀로 로드하고 배타적 버퍼 매핑 잠금을 획득합니다.

원인

이 이벤트가 정상보다 많이 발생하여 성능 문제가 발생할 수 있는 경우 데이터베이스가 공유 버퍼 풀에 페이징 및 페이징 중입니다. 일반적인 원인은 다음과 같습니다.

- 대규모 쿼리
- 부풀린 인덱스 및 테이블
- 전체 테이블 스캔
- 작업 세트보다 작은 공유 풀 크기

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 버퍼 관련 지표 모니터링
- 인덱싱 전략 평가
- 신속하게 할당해야 하는 버퍼 수 저감

버퍼 관련 지표 모니터링

`LWLock:buffer_mapping`이 스파이크를 기다렸다가 버퍼 적중률을 조사합니다. 이러한 지표를 사용하여 버퍼 캐시에서 발생하는 상황을 더 잘 이해할 수 있습니다. 다음 지표를 검토합니다.

`BufferCacheHitRatio`

이 Amazon CloudWatch 지표는 DB 클러스터에서 DB 인스턴스의 버퍼 캐시가 제공하는 요청 백분율을 측정합니다. 이 지표가 `LWLock:buffer_mapping` 대기 이벤트의 리드 업에서 감소하는 것을 볼 수 있습니다.

`blks_hit`

이 성능 개선 도우미 카운터 지표는 공유 버퍼 풀에서 검색된 블록 수를 나타냅니다. `LWLock:buffer_mapping` 대기 이벤트가 발생하면 `blks_hit`에서 스파이크가 발생할 수 있습니다.

`blks_read`

이 성능 개선 도우미 카운터 지표는 공유 버퍼 풀에서 I/O를 읽어야 하는 블록 수를 나타냅니다. `LWLock:buffer_mapping` 대기 이벤트의 리드업에서 `blks_read`의 스파이크가 발생할 수 있습니다.

인덱싱 전략 평가

인덱싱 전략의 성능 저하가 아닌지 확인하려면 다음을 알아보세요.

인덱스 팽창

인덱스와 테이블 팽창으로 인해 불필요한 페이지가 공유 버퍼로 읽히지 않는지 확인합니다. 테이블에 사용되지 않는 행이 포함된 경우 데이터를 보관하고 테이블에서 행을 제거하는 것이 좋습니다. 그런 다음 크기가 조정된 테이블의 인덱스를 다시 작성할 수 있습니다.

자주 사용하는 쿼리의 인덱스

최적의 인덱스가 있는지 확인하려면 성능 개선 도우미에서 DB 엔진 지표를 모니터링하세요. `tup_returned` 지표는 읽은 행의 수를 보여줍니다. `tup_fetched` 지표는 클라이언트에게 반환

되는 행 수를 보여줍니다 tup_returned가 tup_fetched보다 훨씬 큰 경우, 데이터가 제대로 인덱스화되지 않을 수 있습니다. 또한 테이블 통계가 최신 상태가 아닐 수도 있습니다.

신속하게 할당해야 하는 버퍼 수 저감

LWLock:buffer_mapping 대기 이벤트를 줄이려면, 신속하게 할당해야 하는 버퍼 수를 줄이세요. 한 가지 전략은 소규모 배치 작업을 수행하는 것입니다. 테이블을 분할하여 더 작은 배치를 달성할 수 있습니다.

LWLock:BufferIO(IPC:BufferIO)

LWLock:BufferIO 이벤트는 Aurora PostgreSQL 또는 RDS for PostgreSQL이 페이지에 동시에 액세스하려고 할 때 다른 프로세스가 입출력 (I/O) 작업을 완료할 때까지 기다리는 경우에 발생합니다. 그 목적은 동일한 페이지를 공유 버퍼로 읽는 것입니다.

주제

- 관련 엔진 버전
- 컨텍스트
- 원인
- 작업

관련 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전과 관련이 있습니다. Aurora PostgreSQL 12 이하 버전의 경우 이 대기 이벤트의 이름이 lwlock:buffer_io인 반면, Aurora PostgreSQL 13 버전에서는 lwlock:bufferio로 이름이 지정됩니다. Aurora PostgreSQL 14 버전부터는 BufferIO 대기 이벤트가 LWLock에서 IPC 대기 이벤트 유형(IPC:BufferIO)으로 변경되었습니다.

컨텍스트

각 공유 버퍼에는 매번 블록(또는 페이지)이 공유 버퍼 풀 외부에서 회수되야 하는 LWLock:BufferIO 대기 이벤트와 연결된 I/O 잠금이 있습니다.

이 잠금은 모두 동일한 블록에 액세스해야 하는 여러 세션을 처리하는 데 사용됩니다. 이 블록은 shared_buffers 파라미터로 정의된 공유 버퍼 풀 외부에서 읽어야 합니다.

공유 버퍼 풀 내에서 페이지를 읽는 즉시 LWLock:BufferIO 잠금은 해제됩니다.

> **Note**
>
> LWLock:BufferIO 대기 이벤트는 IO:DataFileRead 대기 이벤트에 선행됩니다. IO:DataFileRead 대기 이벤트는 스토리지에서 데이터를 읽는 동안 발생합니다.

경량 잠금에 대한 자세한 내용은 잠금 개요를 참조하세요.

원인

상위 대기에서 나타나는 LWLock:BufferIO 이벤트의 일반적인 원인은 다음을 포함합니다.

- I/O 작업이 보류 중인 동일한 페이지에 액세스하려고 시도하는 여러 백엔드 또는 연결
- 공유 버퍼 풀의 크기 간의 비율(shared_buffers 파라미터로 정의됨) 및 현재 워크로드에 필요한 버퍼 수
- 공유 버퍼 풀의 크기가 다른 작업에서 제거되는 페이지 수와 균형이 맞지 않습니다.
- 엔진이 공유 버퍼 풀에 필요한 것보다 많은 페이지를 읽어야 하는 크거나 부풀린 인덱스
- DB 엔진이 테이블에서 필요한 것보다 더 많은 페이지를 읽도록 하는 인덱스의 부족
- 같은 페이지에서 작업을 수행하려고 시도하는 데이터베이스 연결의 갑작스러운 급증

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- BufferCacheHitRatio 및 LWLock:BufferIO 대기 이벤트의 급격한 감소 간의 상관관계에 대한 Amazon CloudWatch 지표가 관찰됩니다. 이 효과는 공유 버퍼 설정이 작음을 의미할 수 있습니다. DB 인스턴스 클래스를 늘리거나 확장해야 할 수 있습니다. 워크로드를 더 많은 리더 노드로 분할할 수 있습니다.
- 사용되지 않는 인덱스가 있는지 확인한 다음 제거합니다.
- 분할된 테이블(분할된 인덱스도 있음)을 사용합니다. 이렇게 하면 인덱스 재정렬을 낮게 유지하고 영향을 줄일 수 있습니다.
- 불필요하게 열을 인덱싱하지 마세요.
- 연결 풀을 사용하여 갑작스러운 데이터베이스 연결 스파이크를 방지합니다.
- 데이터베이스에 대한 최대 연결 수를 모범 사례로 제한합니다.

LWLock:lock_manager

이 이벤트는 Aurora PostgreSQL 엔진이 빠른 경로 잠금이 불가능할 때 잠금을 할당, 확인 및 할당 해제하기 위해 공유 잠금 메모리 영역을 유지 관리하는 경우에 발생합니다.

주제

지원되는 엔진 버전

이 대기 이벤트 정보는 Aurora PostgreSQL 버전 9.6 이상에서 지원됩니다.

컨텍스트

SQL 문을 실행하면 Aurora PostgreSQL 동시 작업 중에 데이터베이스의 구조, 데이터 및 무결성을 보호하기 위해 잠금을 기록합니다. 엔진은 빠른 경로 잠금 또는 빠르지 않은 경로 잠금을 사용하여 이 목표를 달성할 수 있습니다. 빠르지 않은 경로 잠금은 빠른 경로 잠금보다 비용이 많이 들고 오버헤드가 더 많이 발생합니다.

빠른 경로 잠금

자주 수행되고 해제되지만 충돌이 거의 발생하지 않는 잠금의 오버헤드를 줄이기 위해 백엔드 프로세스에서 빠른 경로 잠금을 사용할 수 있습니다. 데이터베이스는 다음 기준을 충족하는 잠금에 대해 이 메커니즘을 사용합니다.

- DEFAULT 잠금 방법을 사용합니다.
- 공유 관계가 아닌 데이터베이스 관계에 대한 잠금을 나타냅니다.
- 그들은 충돌할 가능성이 없는 약한 잠금입니다.
- 엔진은 충돌하는 잠금이 존재하지 않을 수 있는지 신속하게 확인할 수 있습니다.

다음 조건 중 하나에 부합할 때 엔진은 빠른 경로 잠금을 사용할 수 없습니다.

- 이 잠금이 이전 기준을 충족하지 않습니다.
- 백엔드 프로세스에 사용할 수 있는 슬롯이 더 이상 없습니다.

고속 경로 잠금에 대한 자세한 내용은 잠금 관리자 README의 빠른 경로와 PostgreSQL 설명서의 pg-locks를 참조하세요.

잠금 관리자의 배율 조정 문제 예

이 예에서는 이름이 `purchases`인 테이블이 매일로 분할된 5년짜리 데이터를 저장합니다. 각 파티션에는 두 개의 인덱스가 있습니다. 다음과 같은 일련의 이벤트가 발생합니다.

1. 며칠 분량의 데이터를 쿼리하면 데이터베이스가 많은 파티션을 읽어야 합니다.
2. 데이터베이스는 각 파티션에 대한 잠금 항목을 만듭니다. 파티션 인덱스가 최적기 액세스 경로의 일부인 경우 데이터베이스도 해당 인덱스에 대한 잠금 항목을 만듭니다.
3. 동일한 백엔드 프로세스에 대해 요청된 잠금 항목 수가 `FP_LOCK_SLOTS_PER_BACKEND`의 값인 16보다 높은 경우 잠금 관리자는 비고속 경로 잠금 방법을 사용합니다.

최신 애플리케이션에는 수백 개의 세션이 있을 수 있습니다. 동시 세션이 적절한 파티션 정리 없이 부모를 쿼리하는 경우 데이터베이스는 수백 또는 수천 개의 고속 경로 잠금을 생성할 수 있습니다. 일반적으로 이 동시성이 vCPU 수보다 높으면 `LWLock:lock_manager` 대기 이벤트가 표시됩니다.

> **Note**
>
> `LWLock:lock_manager` 대기 이벤트는 데이터베이스 스키마의 파티션 또는 인덱스 수와 관련이 없습니다. 대신 데이터베이스가 제어해야 하는 비고속 경로 잠금 수와 관련이 있습니다.

대기 증가의 가능한 원인

`LWLock:lock_manager` 대기 이벤트가 정상보다 더 발생하여 성능 문제를 나타낼 수 있으며 갑작스런 스파이크의 가장 큰 원인은 다음과 같습니다.

- 동시 활성 세션은 빠른 경로 잠금을 사용하지 않는 쿼리를 실행하고 있습니다. 이러한 세션도 최대 vCPU를 초과합니다.
- 많은 수의 동시 활성 세션이 심하게 분할된 테이블에 액세스하고 있습니다. 각 파티션에는 여러 개의 인덱스가 있습니다.
- 데이터베이스에 연결 폭풍이 발생합니다. 기본적으로 일부 애플리케이션 및 연결 풀 소프트웨어는 데이터베이스 속도가 느릴 때 더 많은 연결을 만듭니다. 이 관행은 문제를 악화시킵니다. 연결 스톰이 발생하지 않도록 연결 풀 소프트웨어를 튜닝합니다.
- 많은 수의 세션이 파티션을 정리하지 않고 상위 테이블을 쿼리합니다.

- 데이터 정의 언어 (DDL), 유지 관리 명령은 자주 액세스하거나 수정되는 사용 중인 관계식 또는 튜플을 독점적으로 잠급니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

주제

- 파티션 정리 사용
- 불필요한 인덱스 삭제
- 빠른 경로 잠금을 위한 쿼리 조정
- 다른 대기 이벤트 조정
- 하드웨어 병목 현상 저감
- 연결 풀러 사용
- Aurora PostgreSQL 버전 업그레이드

파티션 정리 사용

파티션 정리는 테이블 검색에서 불필요한 파티션을 제외하여 성능을 향상시키는 쿼리 최적화 전략입니다. 파티션 정리는 기본적으로 활성화되어 있습니다. 꺼져 있으면 다음과 같이 켭니다.

```
SET enable_partition_pruning = on;
```

WHERE 절에 파티셔닝에 사용되는 열이 들어 있으면 쿼리는 파티션 정리를 활용할 수 있습니다. 더 자세한 내용은 PostgreSQL 설명서의 파티션 정리를 참조하세요.

불필요한 인덱스 삭제

데이터베이스에 사용되지 않거나 거의 사용되지 않는 인덱스가 포함되어 있을 수 있습니다. 이 경우 삭제하는 것이 좋습니다. 다음 중 하나를 수행하세요.

- 불필요한 인덱스를 찾아내려면, PostgreSQL 위키의 사용되지 않는 인덱스를 읽어보세요.
- PG 컬렉터를 실행합니다. 이 SQL 스크립트는 데이터베이스 정보를 수집하여 통합 HTML 보고서로 표시합니다. '사용되지 않은 인덱스' 섹션을 확인하세요. 자세한 내용은 AWS Labs GitHub 리포지토리의 pg-collector를 참조하세요.

빠른 경로 잠금을 위한 쿼리 조정

쿼리에서 빠른 경로 잠금을 사용하는지 확인하려면 pg_locks 테이블의 fastpath 열을 쿼리하세요. 쿼리에서 빠른 경로 잠금을 사용하지 않는 경우 쿼리당 관계 수를 16 미만으로 줄이세요.

다른 대기 이벤트 조정

LWLock:lock_manager가 최상위 대기 목록에서 첫 번째 또는 두 번째일 경우, 다음 대기 이벤트도 목록에 나타나는지 확인합니다.

- Lock:Relation
- Lock:transactionid
- Lock:tuple

위의 이벤트가 목록에서 높게 표시되는 경우 이러한 대기 이벤트를 먼저 조정하는 것이 좋습니다. 이러한 이벤트는 LWLock:lock_manager 드라이버가 될 수 있습니다.

하드웨어 병목 현상 저감

CPU 부족 또는 Amazon EBS 대역폭의 최대 사용량과 같은 하드웨어 병목 현상이 발생할 수 있습니다. 이 경우에는 하드웨어 병목 현상을 줄이는 것이 좋습니다. 다음 조치를 고려해 보세요.

- 인스턴스 클래스를 확장하세요.
- 대량의 CPU와 메모리를 사용하는 쿼리를 최적화하세요.
- 애플리케이션 로직을 변경하세요.
- 데이터를 아카이빙하세요.

CPU, 메모리 및 EBS 네트워크 대역폭에 대한 자세한 내용은 Amazon RDS 인스턴스 유형을 참조하세요.

연결 풀러 사용

총 활성 연결 수가 최대 vCPU를 초과하는 경우 인스턴스 유형이 지원할 수 있는 것보다 많은 OS 프로세스에 CPU가 필요합니다. 이 경우에는 연결 풀을 사용하거나 튜닝하는 것이 좋습니다. 인스턴스 유형별 vCPU 수에 대한 자세한 내용은 Amazon RDS 인스턴스 유형을 참조하세요.

연결 풀링에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Amazon RDS Proxy for Aurora
- pgbouncer
- PostgreSQL 설명서의 연결 풀 및 데이터 원본

Aurora PostgreSQL 버전 업그레이드

현재 버전의 Aurora PostgreSQL이 12보다 낮은 경우 버전 12 이상으로 업그레이드하세요. PostgreSQL 버전 12 및 13에는 향상된 파티션 메커니즘이 있습니다. 버전 12의 더 자세한 정보는 PostgreSQL 릴리스 12.0을 참조하세요. Aurora PostgreSQL 업그레이드에 대한 자세한 내용은 Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 섹션을 참조하세요.

LWLock:MultiXact

`LWLock:MultiXactMemberBuffer`, `LWLock:MultiXactOffsetBuffer`, `LWLock:MultiXactMemberSLRU` 및 `LWLock:MultiXactOffsetSLRU` 대기 이벤트는 세션이 지정된 테이블에 있는 동일한 행을 수정하는 트랜잭션 목록을 검색하기 위해 대기 중임을 나타냅니다.

- `LWLock:MultiXactMemberBuffer` – 프로세스가 multixact 멤버를 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.
- `LWLock:MultiXactMemberSLRU` – 프로세스가 multixact 멤버를 위해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.
- `LWLock:MultiXactOffsetBuffer` – 프로세스가 multixact 오프셋을 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.
- `LWLock:MultiXactOffsetSLRU` – 프로세스가 multixact 오프셋을 위해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

multixact는 동일한 테이블 행을 수정하는 트랜잭션 ID(XID) 목록을 저장하는 데이터 구조입니다. 단일 트랜잭션이 특정 테이블의 행을 참조하는 경우 트랜잭션 ID가 해당 테이블의 헤더 행에 저장됩니다. 여러 트랜잭션이 특정 테이블의 동일한 행을 참조하는 경우 트랜잭션 ID 목록이 multixact 데이터 구조에 저장됩니다. multixact 대기 이벤트는 세션이 데이터 구조에서 테이블의 지정된 행을 참조하는 트랜잭션 목록을 검색하고 있음을 나타냅니다.

대기 증가의 가능한 원인

multixact를 사용하는 세 가지 일반적인 원인은 다음과 같습니다.

- 명시적 세이브 포인트의 하위 트랜잭션 – 트랜잭션에서 명시적으로 저장점을 생성하면 동일한 행에 대한 새 트랜잭션이 생성됩니다. 예를 들자면 `SELECT FOR UPDATE`를 사용한 다음 `SAVEPOINT`와 `UPDATE`를 차례대로 사용하는 식입니다.

  일부 드라이버, ORM(객체 관계 매퍼) 및 추상화 계층에는 모든 작업을 저장점으로 자동 래핑하는 구성 옵션이 있습니다. 이 옵션 때문에 일부 워크로드에서 multixact 대기 이벤트가 많이 생성될 수 있습니다. PostgreSQL JDBC 드라이버의 `autosave` 옵션이 대표적인 예입니다. 자세한 내용은 PostgreSQL JDBC 설명서의 [pgJDBC](https://jdbc.postgresql.org/documentation/use/)를 참조하세요. 또 다른 예는 PostgreSQL ODBC 드라이버와 이 드라이버의 `protocol` 옵션입니다. 자세한 내용은 PostgreSQL ODBC 드라이버 설명서의 [psqlODBC 구성 옵션](https://odbc.postgresql.org/docs/config.html)을 참조하세요.
- PL/pgSQL EXCEPTION 조항의 하위 트랜잭션 - PL/pgSQL 함수 또는 프로시저에서 작성한 각 `EXCEPTION` 절은 `SAVEPOINT`를 내부에서 생성합니다.
- 외래 키 - 여러 트랜잭션이 상위 행에 대한 공유 잠금을 획득합니다.

지정된 행이 다중 트랜잭션 작업에 포함된 경우, 행을 처리하려면 `multixact` 목록에서 트랜잭션 ID를 검색해야 합니다. 조회 시 메모리 캐시에서 multixact를 가져올 수 없는 경우, Aurora 스토리지 계층에서 데이터 구조를 읽어야 합니다. 스토리지의 이 I/O에서는 SQL 쿼리가 더 오래 걸릴 수 있습니다. 다중 트랜잭션이 너무 많아 사용량이 많아지만 메모리 캐시 누락이 발생할 수 있습니다. 이러한 모든 요인은 이 대기 이벤트의 증가를 유발합니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다. 이러한 작업 중 일부는 대기 이벤트를 즉시 줄이는 데 도움이 될 수 있습니다. 하지만 워크로드의 규모를 조정하기 위해 조사 및 수정이 필요한 경우도 있습니다.

주제

- [이 대기 이벤트가 발생한 테이블에서 vacuum freeze 수행]
- [이 대기 이벤트가 있는 테이블의 autovacuum 빈도 상향]
- [메모리 파라미터 상향]
- [장기 실행 트랜잭션 저감]
- [장기적인 조치]

이 대기 이벤트가 발생한 테이블에서 vacuum freeze 수행

이 대기 이벤트가 급증하여 프로덕션 환경에 영향을 미치는 경우, 다음과 같은 임시 방법 중 하나를 사용하여 이벤트 수를 줄일 수 있습니다.

- 영향을 받는 테이블 또는 테이블 파티션에서 VACUUM FREEZE를 사용하여 문제를 즉시 해결합니다. 자세한 내용은 VACUUM을 참조하세요.
- VACUUM(FREEZE, INDEX_CLEANUP FALSE) 절을 사용하여 인덱스를 건너뛰고 빠르게 vacuum을 수행합니다. 자세한 내용은 테이블에 최대한 신속하게 vacuum 실행을 참조하세요.

이 대기 이벤트가 있는 테이블의 autovacuum 빈도 상향

모든 데이터베이스의 모든 테이블을 스캔한 후 VACUUM은 multixact를 제거하며, 가장 오래된 multixact 값이 승격됩니다. 자세한 내용은 Multixacts 및 랩어라운드를 참조하세요. LWLock:MultiXact 대기 이벤트를 최소화하려면 VACUUM을 필요한 만큼 자주 실행해야 합니다. 이렇게 하려면 Aurora PostgreSQL DB 클러스터의 VACUUM이 최적으로 구성되어 있어야 합니다.

영향을 받는 테이블 또는 테이블 파티션에서 VACUUM FREEZE를 사용하여 대기 이벤트 문제가 해결되면 인스턴스 수준에서 autovacuum을 조정하는 대신 VACUUM을 수행할 수 있도록 `pg_cron` 등의 스케줄러를 사용하는 것이 좋습니다.

autovacuum이 더 자주 실행되도록 하려면 영향을 받는 테이블의 스토리지 파라미터 `autovacuum_multixact_freeze_max_age`의 값을 줄일 수 있습니다. 자세한 내용은 autovacuum_multixact_freeze_max_age를 참조하세요.

메모리 파라미터 상향

다음 파라미터를 조정하여 multixact 캐시의 메모리 사용량을 최적화할 수 있습니다. 이러한 설정은 이러한 캐시에 예약되는 메모리의 양을 제어하므로 워크로드의 multixact 대기 이벤트를 줄이는 데 도움이 될 수 있습니다. 다음 값으로 시작하는 것이 좋습니다.

Aurora PostgreSQL 17 이상의 경우:

- `multixact_offset_buffers` = 128
- `multixact_member_buffers` = 256

Aurora PostgreSQL 16 이하의 경우:

- `multixact_offsets_cache_size` = 128
- `multixact_members_cache_size` = 256

> **Note**
>
> Aurora PostgreSQL 17에서는 커뮤니티 PostgreSQL 17에 맞게 파라미터 이름이 `multixact_offsets_cache_size`에서 `multixact_offset_buffers`로, `multixact_members_cache_size`에서 `multixact_member_buffers`로 변경되었습니다.

클러스터의 모든 인스턴스가 일관되게 유지되도록 클러스터 수준에서 이러한 파라미터를 설정할 수 있습니다. 특정 워크로드 요구 사항 및 인스턴스 클래스에 가장 적합하도록 값을 테스트하고 조정하는 것이 좋습니다. 파라미터 변경 사항을 적용하려면 인스턴스를 재부팅해야 합니다.

파라미터는 multixact 캐시 항목으로 표시됩니다. 각 캐시 항목은 `8 KB`의 메모리를 사용합니다. 예약된 총 메모리를 계산하려면 각 파라미터 값에 `8 KB`를 곱합니다. 예를 들어 파라미터를 128로 설정하면 총 예약 메모리는 `128 * 8 KB = 1 MB`입니다.

장기 실행 트랜잭션 저감

장기 실행 트랜잭션은 트랜잭션이 커밋되거나 읽기 전용 트랜잭션이 종료될 때까지 vacuum이 트랜잭션 정보를 유지합니다. 장기 실행 트랜잭션을 사전에 모니터링하고 관리하는 것이 좋습니다. 자세한 내용은 [데이터베이스가 트랜잭션 연결 시 오랫동안 유휴 상태로 실행됨](#) 섹션을 참조하세요. 장기 실행 트랜잭션의 사용을 방지하거나 최소화하도록 애플리케이션을 수정해 보세요.

장기적인 조치

워크로드를 검사하여 multixact 스필오버의 원인을 찾아내세요. 워크로드의 규모를 조정하고 대기 이벤트를 줄이려면 문제를 해결해야 합니다.

- 테이블을 만드는 데 사용한 DDL(데이터 정의 언어)을 분석해야 합니다. 테이블 구조와 인덱스가 제대로 설계되었는지 확인하세요.

- 영향을 받는 테이블에 외래 키가 있는 경우, 외래 키가 필요한지 또는 참조 무결성을 적용할 다른 방법이 있는지 파악하세요.
- 테이블에 사용되지 않은 대용량 인덱스가 있으면 autovacuum이 워크로드에 맞지 않아 실행이 차단될 수 있습니다. 이를 방지하려면 사용되지 않은 인덱스가 있는지 확인하고, 있는 경우 이를 완전히 제거하세요. 자세한 내용은 [대용량 인덱스가 있는 autovacuum 관리]를 참조하세요.
- 트랜잭션에서 세이브 포인트 사용을 줄이세요.

LWLock:pg_stat_statements

LWLock:pg_stat_statements 대기 이벤트는 `pg_stat_statements` 확장이 SQL 문을 추적하는 해시 테이블에서 배타적 잠금을 수행할 때 발생합니다. 다음 시나리오에서 발생할 수 있습니다.

- 추적된 문 수가 구성된 `pg_stat_statements.max` 파라미터 값에 도달하고 더 많은 항목을 위한 공간을 확보해야 하는 경우, 확장은 직접 호출 수를 기준으로 정렬을 수행하고 가장 적게 실행된 문 중 5%를 제거한 다음 해시를 나머지 항목으로 다시 채웁니다.
- `pg_stat_statements`가 디스크의 `pgss_query_texts.stat` 파일에 대해 `garbage collection` 작업을 수행하고 파일을 다시 쓰는 경우

주제

- 지원되는 엔진 버전
- 컨텍스트
- 대기 증가의 가능한 원인
- 작업

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 Aurora PostgreSQL 버전에서 지원됩니다.

컨텍스트

pg_stat_statements 확장 이해 - pg_stat_statements 확장은 해시 테이블에서 SQL 문 실행 통계를 추적합니다. 이 확장은 `pg_stat_statements.max` 파라미터에서 정의된 한도까지 SQL 문을 추적합니다. 이 파라미터는 추적할 수 있는 문의 최대 수를 결정하며, 이는 pg_stat_statements 뷰의 최대 행 수에 해당합니다.

문 통계 지속성 - 이 확장은 다음을 통해 인스턴스 재시작 시 문 통계를 유지합니다.

- pg_stat_statements.stat라는 파일에 데이터 쓰기
- pg_stat_statements.save 파라미터를 사용하여 지속성 동작 제어

pg_stat_statements.save가 다음과 같이 설정된 경우:

- on(기본값): 통계는 종료 시 저장되고 서버 시작 시 다시 로드됩니다.
- off: 통계는 종료 시 저장되지 않으며 서버 시작 시 다시 로드되지 않습니다.

쿼리 텍스트 저장소 - 이 확장은 추적된 쿼리의 텍스트를 `pgss_query_texts.stat`라는 파일에 저장합니다. 이 파일은 폐영역 회수가 발생하기 전에 추적된 모든 SQL 문의 평균 크기의 두 배까지 늘어날 수 있습니다. 이 확장에는 정리 작업 및 `pgss_query_texts.stat` 파일 재작성 중에 해시 테이블에 대한 배타적 잠금이 필요합니다.

문 할당 취소 프로세스 - 추적된 문 수가 `pg_stat_statements.max` 한도에 도달하고 새 문을 추적해야 하는 경우 확장은 다음을 수행합니다.

- 해시 테이블에서 배타적 잠금(LWLock:pg_stat_statements)을 수행합니다.
- 기존 데이터를 로컬 메모리에 로드합니다.
- 직접 호출 수를 기반으로 빠른 정렬을 수행합니다.
- 가장 적게 호출된 문(하위 5%)을 제거합니다.
- 해시 테이블을 나머지 항목으로 다시 채웁니다.

문 할당 취소 모니터링 - PostgreSQL 14 이상에서는 pg_stat_statements_info 뷰를 사용하여 문 할당 취소를 모니터링할 수 있습니다. 이 뷰에는 새 문을 위한 공간을 마련하기 위해 문을 할당 취소한 횟수를 보여주는 dealloc 열이 포함되어 있습니다.

문의 할당 취소가 자주 발생하면 디스크에서 `pgss_query_texts.stat` 파일의 폐영역 회수가 더 자주 발생합니다.

대기 증가의 가능한 원인

`LWLock:pg_stat_statements` 대기 시간 증가의 일반적인 원인은 다음과 같습니다.

- 애플리케이션에서 사용하는 고유 쿼리 수가 증가함
- 사용되는 고유 쿼리 수 대비 `pg_stat_statements.max` 파라미터 값이 작음

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다. Amazon RDS Performance Insights를 사용하거나 `pg_stat_activity` 뷰를 쿼리하여 `LWLock:pg_stat_statements` 이벤트를 식별할 수 있습니다.

다음 `pg_stat_statements` 파라미터를 조정하여 추적 동작을 제어하고 LWLock:pg_stat_ 문 대기 이벤트를 줄입니다.

주제

- pg_stat_statements.track 파라미터 비활성화
- pg_stat_statements.max 파라미터 증가
- pg_stat_statements.track_utility 파라미터 비활성화

pg_stat_statements.track 파라미터 비활성화

LWLock:pg_stat_statements 대기 이벤트가 데이터베이스 성능에 부정적인 영향을 미치고 근본 원인을 식별하기 위해 `pg_stat_statements` 뷰를 추가로 분석하기 전에 신속한 솔루션이 필요한 경우 `pg_stat_statements.track` 파라미터를 none으로 설정하여 파라미터를 비활성화할 수 있습니다. 그러면 문 통계 수집이 비활성화됩니다.

pg_stat_statements.max 파라미터 증가

할당 취소를 줄이고 디스크에서 `pgss_query_texts.stat` 파일의 폐영역 회수를 최소화하려면 `pg_stat_statements.max` 파라미터 값을 늘립니다. 기본값은 `5,000`입니다.

> **Note**
>
> `pg_stat_statements.max` 파라미터는 정적입니다. 이 파라미터에 변경 사항을 적용하려면 DB 인스턴스를 다시 시작해야 합니다.

pg_stat_statements.track_utility 파라미터 비활성화

pg_stat_statements 뷰를 분석하여 `pg_stat_statements`에서 추적하는 리소스를 가장 많이 소비하는 유틸리티 명령을 확인할 수 있습니다.

pg_stat_statements.track_utility 파라미터는 모듈이 SELECT, INSERT, UPDATE, DELETE 및 MERGE를 제외한 모든 명령을 포함하는 유틸리티 명령을 추적하는지를 제어합니다. 이 파라미터는 기본적으로 on로 설정되어 있습니다.

예를 들어 애플리케이션이 본질적으로 고유한 많은 저장점 쿼리를 사용하는 경우 문 할당 취소가 증가할 수 있습니다. 이를 해결하기 위해 pg_stat_statements.track_utility 파라미터를 비활성화하여 pg_stat_statements가 저장점 쿼리를 추적하지 않도록 할 수 있습니다.

> Note
>
> pg_stat_statements.track_utility 파라미터는 동적 파라미터입니다. 데이터베이스 인스턴스를 다시 시작하지 않고도 값을 변경할 수 있습니다.

Example pg_stat_statements의 고유한 저장점 쿼리 예시

```
                     query                       |       queryid
-------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                 | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                   | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                  |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016      | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                  | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                  | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                   |  6388797791882029332
```

PostgreSQL 17에는 유틸리티 명령 추적을 위한 몇 가지 향상된 기능이 도입되었습니다.

- 이제 Savepoint 이름이 상수로 표시됩니다.
- 이제 두 단계 커밋 명령의 글로벌 트랜잭션 ID(GID)가 상수로 표시됩니다.
- DEALLOCATE 문의 이름이 상수로 표시됩니다.
- 이제 CALL 파라미터가 상수로 표시됩니다.

Timeout:PgSleep

Timeout:PgSleep 이벤트는 서버 프로세스가 pg_sleep 함수를 가리키고 및 절전 시간 초과가 만료될 때까지 대기할 때 발생합니다.

주제

- [지원되는 엔진 버전]
- [대기 증가의 가능한 원인]
- [작업]

지원되는 엔진 버전

이 대기 이벤트 정보는 모든 버전의 Aurora PostgreSQL에서 지원됩니다.

대기 증가의 가능한 원인

이 대기 이벤트는 애플리케이션, 저장된 함수 또는 사용자가 다음 함수 중 하나를 호출하는 SQL 문을 실행할 때 발생합니다.

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

위의 함수는 지정된 시간(초)이 경과할 때까지 실행을 지연시킵니다. 예를 들어, `SELECT pg_sleep(1)`은 1초 동안 일시 중지됩니다. 자세한 내용은 PostgreSQL 설명서의 [실행 지연]을 참조하세요.

작업

`pg_sleep` 함수를 실행 중인 명령문을 식별합니다. 함수의 사용이 적절한지 확인합니다.

Amazon DevOps Guru의 사전 예방 인사이트를 활용하여 Aurora PostgreSQL 튜닝

DevOps Guru의 사전 예방 인사이트는 Aurora PostgreSQL DB 클러스터에서 문제를 일으킬 수 있는 조건을 감지하여 문제가 발생하기 전에 이를 알려줍니다. 사전 예방적 인사이트를 통해 트랜잭션 연결의 장기 유휴 상태를 경고할 수 있습니다. 트랜잭션 연결에서 유휴 상태로 오래 실행되는 문제 해결에 대한 자세한 내용은 [데이터베이스가 트랜잭션 연결 시 오랫동안 유휴 상태로 실행됨] 섹션을 참조하세요.

DevOps Guru는 다음과 같은 작업을 수행할 수 있습니다.

- 데이터베이스 구성을 일반적인 권장 설정과 교차 검사하여 여러 가지 일반적인 데이터베이스 문제를 방지합니다.
- 확인하지 않은 상태로 두면 나중에 더 큰 문제로 이어질 수 있는 플릿의 심각한 문제를 알려줍니다.
- 새로 발견된 문제를 알려줍니다.

모든 사전 예방 인사이트에는 문제의 원인에 대한 분석과 수정 조치를 위한 권장 사항이 포함됩니다.

Amazon DevOps Guru for Amazon RDS에 대한 자세한 내용은 Amazon DevOps Guru for Amazon RDS로 Aurora 성능 이상 분석 섹션을 참조하세요.

데이터베이스가 트랜잭션 연결 시 오랫동안 유휴 상태로 실행됨

데이터베이스에 대한 연결이 1,800초 이상 `idle in transaction` 상태로 지속되었습니다.

주제

지원되는 엔진 버전

이 인사이트 정보는 Aurora PostgreSQL의 모든 버전에서 지원됩니다.

컨텍스트

해당 `idle in transaction` 상태의 트랜잭션은 다른 쿼리를 차단하는 잠금을 유지할 수 있습니다. 또한 VACUUM(autovacuum 포함)이 잘못된 행을 정리하여 인덱스 또는 테이블 팽창 또는 트랜잭션 ID 랩어라운드로 이어지는 것을 방지할 수 있습니다.

이 문제의 잠재적 원인

BEGIN 또는 START TRANSACTION을 사용하여 대화형 세션에서 시작된 트랜잭션이 COMMIT, ROLLBACK 또는 END 명령을 사용하여 종료되지 않았습니다. 이 경우 트랜잭션이 `idle in transaction` 상태로 이동합니다.

작업

pg_stat_activity 쿼리를 통해 유휴 트랜잭션을 찾을 수 있습니다.

SQL 클라이언트에서 다음 쿼리를 실행하여 idle in transaction 상태의 모든 연결을 나열하고 기간별로 정렬합니다.

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

인사이트의 원인에 따라 다른 조치를 취할 것을 권장합니다.

주제

- 트랜잭션 종료
- 연결 종료
- idle_in_transaction_session_timeout 파라미터 구성
- AUTOCOMMIT 상태 확인
- 애플리케이션 코드에서 트랜잭션 로직 확인

트랜잭션 종료

BEGIN 또는 START TRANSACTION을 사용하여 대화형 세션에서 트랜잭션을 시작하면 트랜잭션이 idle in transaction 상태로 이동합니다. COMMIT, ROLLBACK, END 명령을 실행하여 트랜잭션을 종료하거나 연결을 완전히 끊어 트랜잭션을 롤백할 때까지 이 상태로 유지됩니다.

연결 종료

다음 쿼리를 사용하여 유휴 트랜잭션과의 연결을 종료합니다.

```
SELECT pg_terminate_backend(pid);
```

pid는 연결의 프로세스 ID입니다.

idle_in_transaction_session_timeout 파라미터 구성

파라미터 그룹에서 `idle_in_transaction_session_timeout` 파라미터를 구성합니다. 이 파라미터를 구성하면 트랜잭션의 오랜 유휴 상태를 종료하기 위해 수동 개입이 필요하지 않다는 이점이 있습니다. 이 파라미터에 대한 자세한 내용은 PostgreSQL 설명서를 참조하세요.

트랜잭션이 idle_in_transaction 상태로 지정된 시간보다 오래 열려 있는 경우 연결이 종료된 후 PostgreSQL 로그 파일에 다음 메시지가 보고됩니다.

```
FATAL: terminating connection due to idle in transaction timeout
```

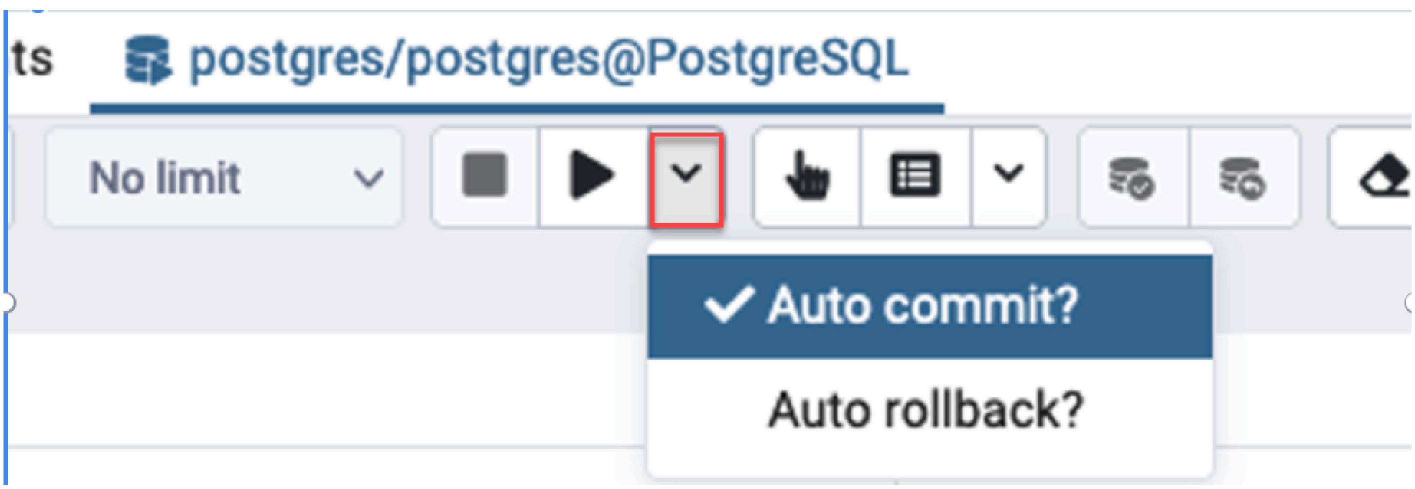
AUTOCOMMIT 상태 확인

AUTOCOMMIT은 기본적으로 활성화되어 있습니다. 하지만 클라이언트에서 실수로 비활성화된 경우에는 반드시 다시 활성화해야 합니다.

- psql 클라이언트에서 다음 명령을 실행합니다.

  ```
  postgres=> \set AUTOCOMMIT on
  ```

- pgadmin에서 아래쪽 화살표에서 AUTOCOMMIT 옵션을 선택하여 활성화합니다.



애플리케이션 코드에서 트랜잭션 로직 확인

애플리케이션 로직에 문제가 있는지 조사합니다. 다음 조치를 고려해 보세요.

- JDBC 자동 커밋이 애플리케이션에서 참으로 설정되어 있는지 확인합니다. 또한 코드에 명시적 `COMMIT` 명령을 사용하는 것도 고려해 보세요.

- 오류 처리 로직을 확인하여 오류 발생 후 트랜잭션이 종료되는지 확인합니다.
- 트랜잭션이 열려 있는 동안 애플리케이션이 쿼리에서 반환된 행을 처리하는 데 시간이 오래 걸리는지 확인합니다. 그렇다면 행을 처리하기 전에 트랜잭션을 닫도록 애플리케이션을 코딩하는 것을 고려해 보세요.
- 트랜잭션에 장기 실행 작업이 많이 포함되어 있는지 확인합니다. 그렇다면 단일 트랜잭션을 여러 트랜잭션으로 나누세요.

관련 지표

이 인사이트와 관련된 PI 지표는 다음과 같습니다.

- idle_in_transaction_count - `idle in transaction` 상태에 있는 세션 수입니다.
- idle_in_transaction_max_time - `idle in transaction` 상태에서 가장 오래 실행되는 트랜잭션의 지속 시간입니다.

Amazon Aurora PostgreSQL 모범 사례

다음에서는 Amazon Aurora PostgreSQL DB 클러스터를 관리하기 위한 몇 가지 모범 사례를 찾을 수 있습니다. 기본적인 유지 관리 작업도 검토하세요. 자세한 내용은 Amazon Aurora PostgreSQL의 성능 및 규모 조정 섹션을 참조하세요.

주제

Aurora PostgreSQL DB 인스턴스의 성능 저하, 자동 재시작 및 장애 조치 방지

워크로드가 많거나 워크로드가 DB 인스턴스의 할당된 리소스 이상으로 급증하는 경우 애플리케이션과 Aurora 데이터베이스를 실행하는 데 사용되는 리소스가 소진될 수 있습니다. CPU 사용률, 메모리 사용량, 사용된 데이터베이스 연결 수와 같은 데이터베이스 인스턴스에 대한 지표를 얻으려면 Amazon CloudWatch, 성능 개선 도우미 및 향상된 모니터링에서 제공하는 지표를 참조할 수 있습니다. DB 인스턴스 모니터링에 대한 자세한 내용은 Amazon Aurora 클러스터에서 지표 모니터링 섹션을 참조하세요.

워크로드가 사용 중인 리소스를 소진하면 DB 인스턴스가 느려지거나, 다시 시작되거나, 다른 DB 인스턴스로 장애 조치될 수 있습니다. 이를 방지하려면 리소스 사용률을 모니터링하고, DB 인스턴스에서 실행되는 워크로드를 검사하고, 필요한 경우 최적화하세요. 최적화가 인스턴스 지표를 개선하지 못하고 리소스 소진을 완화할 수 없다면 한도에 도달하기 전에 DB 인스턴스를 스케일 업하는 것을 고려해 보세요. 사용 가능한 DB 인스턴스 클래스 및 사양에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

테이블 및 인덱스 팽창 진단

PostgreSQL 다중 버전 동시성 제어(MVCC)를 사용하여 데이터 무결성을 유지할 수 있습니다. PostgreSQL MVCC는 트랜잭션이 커밋되거나 롤백될 때까지 업데이트 또는 삭제된 행(튜플이라고도 함)의 내부 복사본을 저장하는 방식으로 작동합니다. 이 저장된 내부 복사본은 사용자에게 표시되지 않습니다. 그러나 VACUUM 또는 AUTOVACUUM 유틸리티를 사용해 이러한 보이지 않는 복사본을 주기적으로 정리하지 않으면 테이블 팽창이 발생할 수 있습니다. 테이블 팽창을 확인하지 않으면 스토리지 비용이 증가하고 처리 속도가 저하될 수 있습니다.

대부분의 경우 Aurora의 VACUUUM 또는 AUTOVACUUM에 대한 기본 설정만으로도 원치 않는 테이블 팽창을 충분히 처리할 수 있습니다. 하지만 애플리케이션에서 다음과 같은 상황이 발생할 경우에는 팽창 여부를 확인하는 것이 좋습니다.

- 비교적 짧은 시간에 여러 VACUUM 프로세스 간에 많은 수의 트랜잭션을 처리하는 경우.
- 성능이 저하되고 스토리지 용량이 부족한 경우.

시작하려면 죽은 튜플이 얼마나 많은 공간을 사용하고 있는지, 그리고 테이블 및 인덱스 팽창을 정리하면 어느 정도의 공간을 복구할 수 있는지에 대한 가장 정확한 정보를 수집하세요. 이를 수행하려면 `pgstattuple` 확장을 사용하여 Aurora 클러스터에서 통계를 수집합니다. 자세한 내용은 pgstattuple을 참조하세요. `pgstattuple` 확장을 사용할 수 있는 권한은 `pg_stat_scan_tables` 역할 및 데이터베이스 수퍼유저로 제한됩니다.

Aurora에서 pgstattuple 확장을 생성하려면 클라이언트 세션을 클러스터(예: psql 또는 pgAdmin)에 연결하고 다음 명령을 사용하세요.

```
CREATE EXTENSION pgstattuple;
```

프로파일링하려는 각 데이터베이스에서 확장을 생성합니다. 확장을 생성한 후 명령줄 인터페이스(CLI)를 사용하여 회수할 수 있는 사용 불가 공간이 얼마인지 측정하세요. 통계를 수집하기 전에 AUTOVACUUM을 0으로 설정하여 클러스터 파라미터 그룹을 수정합니다. 0으로 설정하면 Aurora가 애플리케이션에서 남긴 죽은 튜플을 자동으로 정리할 수 없으므로, 결과의 정확도에 영향을 미칠 수 있습니다. 다음 명령을 입력하여 간단한 테이블을 생성합니다.

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

다음 예에서는 DB 클러스터에 대해 AUTOVACUUM이 켜진 상태에서 쿼리를 실행합니다. dead_tuple_count는 0이며, 이는 AUTOVACUUM이 사용되지 않는 데이터나 튜플을 PostgreSQL 데이터베이스에서 삭제했음을 나타냅니다.

pgstattuple을 사용하여 테이블에 대한 정보를 수집하려면 쿼리에 테이블 이름 또는 객체 식별자(OID)를 지정합니다.

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

다음 쿼리에서는 AUTOVACUUM을 끄고 테이블에서 25,000개의 행을 삭제하는 명령을 입력합니다. 그 결과, dead_tuple_count가 25000으로 증가합니다.

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+--------------------+-------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

죽은 튜플을 회수하려면 VACUUM 프로세스를 시작합니다.

애플리케이션 중단 없이 팽창 관찰

Aurora 클러스터의 설정은 대부분의 워크로드에 대한 모범 사례를 제공하도록 최적화되어 있습니다. 하지만 사용자의 애플리케이션 및 사용 패턴에 더 잘 맞도록 클러스터를 최적화할 수도 있습니다. 이 경우 사용량이 많은 애플리케이션을 중단하지 않고 `pgstattuple` 확장을 사용하면 됩니다. 이렇게 하려면 다음 단계를 수행합니다.

1. Aurora 인스턴스를 복제합니다.
2. 파라미터 파일을 수정하여 클론에서 AUTOVACUUM을 끕니다.
3. 샘플 워크로드 또는 pgbench(PostgreSQL에서 벤치마크 테스트를 실행하기 위한 프로그램)를 사용하여 클론을 테스트하는 동안 `pgstattuple` 쿼리를 수행합니다. 자세한 내용은 [pgbench](https://www.postgresql.org/docs/current/pgbench.html) 섹션을 참조하세요.

애플리케이션을 실행하고 결과를 확인한 후, 복원된 사본에서 pg_repack 또는 VACUUM FULL을 사용하여 차이점을 비교해 봅니다. dead_tuple_count, dead_tuple_len 또는 dead_tuple_percent가 크게 감소한 경우 프로덕션 클러스터의 정리 일정을 조정하여 팽창을 최소화하세요.

임시 테이블의 팽창 방지

애플리케이션에서 임시 테이블을 생성할 경우, 이러한 임시 테이블이 더 이상 필요하지 않을 때는 애플리케이션에서 해당 임시 테이블을 제거해야 합니다. 자동 정리 프로세스에서는 임시 테이블을 찾지 않습니다. 임시 테이블을 확인하지 않은 상태로 두면 데이터베이스 팽창이 빠르게 발생할 수 있습니다. 또한 팽창 현상은 시스템 테이블로 확장될 수 있습니다. 시스템 테이블은 pg_attribute 및 pg_depend와 같은 PostgreSQL 객체 및 속성을 추적하는 내부 테이블입니다.

임시 테이블이 더 이상 필요하지 않은 경우 TRUNCATE 문을 사용하여 테이블을 비우고 공간을 확보할 수 있습니다. 그런 다음, pg_attribute 및 pg_depend 테이블을 수동으로 정리합니다. 이러한 테이블을 정리하면 임시 테이블을 생성하고 잘라내거나 삭제해도 계속해서 튜플이 추가되거나 시스템 팽창이 발생하지 않습니다.

콘텐츠가 커밋될 때 새 행을 삭제하는 다음 구문을 포함하면 임시 테이블을 생성하는 동안 이러한 문제를 방지할 수 있습니다.

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

이 `ON COMMIT DELETE ROWS` 절은 트랜잭션이 커밋될 때 임시 테이블을 잘라냅니다.

인덱스의 팽창 방지

테이블에서 인덱싱된 필드를 변경하면 인덱스 업데이트로 인해 해당 인덱스에 하나 이상의 죽은 튜플이 생성됩니다. 기본적으로 autovacuum 프로세스는 인덱스의 팽창을 정리하지만, 이러한 정리에는 상당히 많은 시간과 리소스가 사용됩니다. 테이블을 생성할 때 인덱스 정리 기본 설정을 지정하려면 vacuum_index_cleanup 절을 포함하세요. 기본적으로 테이블 생성 시 해당 절은 AUTO로 설정됩니다. 즉, 테이블을 정리할 때 서버가 인덱스 정리가 필요한지 여부를 결정합니다. 이 절을 ON으로 설정하면 특정 테이블에 대한 인덱스 정리를 켤 수 있고, OFF로 설정하면 해당 테이블에 대한 인덱스 정리를 끌 수 있습니다. 인덱스 정리를 끄면 시간이 절약될 수 있지만 인덱스가 팽창할 수 있다는 점을 유의하세요.

명령줄에서 테이블에 VACUUM을 실행할 때 인덱스 정리를 수동으로 제어할 수 있습니다. 테이블을 정리하고 인덱스에서 죽은 튜플을 제거하려면 값이 ON이고 테이블 이름이 있는 INDEX_CLEANUP 절을 포함하세요.

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

인덱스를 지우지 않고 테이블을 정리하려면 값을 OFF로 지정합니다.

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Aurora PostgreSQL의 향상된 메모리 관리

Aurora PostgreSQL에는 이제 다양한 워크로드에서 데이터베이스 성능과 복원력을 최적화하는 고급 메모리 관리 기능이 포함되어 있습니다. 이러한 개선 사항을 통해 Aurora PostgreSQL은 메모리 수요가 높은 기간에도 일관된 가용성과 응답성을 유지할 수 있습니다.

이 기능은 다음 Aurora PostgreSQL 버전에서 프로비저닝된 인스턴스에 대해 사용할 수 있으며 기본적으로 활성화되어 있습니다.

- 15.3 이상의 모든 마이너 버전
- 14.8 이상의 마이너 버전
- 13.11 이상의 마이너 버전
- 12.15 이상의 마이너 버전
- 11.20 이상의 마이너 버전

이 기능은 다음 Aurora PostgreSQL 버전에서 Aurora Serverless 인스턴스에 대해 사용할 수 있으며 기본적으로 활성화되어 있습니다.

- 16.3 이상의 모든 마이너 버전
- 15.7 이상의 모든 마이너 버전
- 14.12 이상의 마이너 버전
- 13.15 이상의 마이너 버전

고객 워크로드가 사용 가능한 여유 메모리를 모두 사용하는 경우 운영 체제는 리소스를 보호하기 위해 데이터베이스를 다시 시작하여 일시적으로 사용할 수 없게 될 수 있습니다. Aurora PostgreSQL의 새로운 메모리 관리 개선 사항은 시스템에 메모리 압박이 높을 때 특정 트랜잭션을 사전에 취소하여 데이터베이스 안정성을 유지하는 데 도움이 됩니다.

개선된 메모리 관리의 주요 기능은 다음과 같습니다.

- 시스템의 메모리 사용량이 한계에 다다랐을 때 더 많은 메모리를 요청하는 데이터베이스 트랜잭션을 취소합니다.
- 시스템이 물리적 메모리를 모두 소진하고 스왑을 소진하려고 할 때 메모리 사용량이 한계에 다다랐다고 합니다. 이러한 상황에서는 DB 인스턴스의 메모리 사용량을 즉시 줄이기 위해 메모리를 요청하는 모든 트랜잭션이 취소됩니다.
- 필수 PostgreSQL 시작 관리자 및 Autovacuum 작업자 등의 백그라운드 작업자는 항상 보호됩니다.

메모리 관리 파라미터 취급

메모리 관리를 사용 설정하는 방법

이 옵션은 기본적으로 켜져 있습니다. 다음 예제와 같이 메모리 부족으로 트랜잭션이 취소되면 오류 메시지가 표시됩니다.

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

메모리 관리를 사용 해제하는 방법

이 기능을 해제하려면 아래와 같이 psql로 Aurora PostgreSQL DB 클러스터에 연결하고 파라미터 값에 SET 문을 사용합니다.

> **Note**
> 메모리 관리는 활성화된 상태로 유지하는 것이 좋습니다. 이렇게 하면 메모리 소진으로 인해 워크로드에서 유발되는 데이터베이스 재시작으로 이어질 수 있는 메모리 부족 오류를 방지할 수 있습니다.

다음 표는 다양한 Aurora PostgreSQL 버전에서 메모리 관리 기능을 끄는 방법을 보여줍니다.

Aurora PostgreSQL 버전	파라미터	Default	세션 수준에서 메모리 관리를 끄는 명령
11.20, 11.21, 12.15, 12.16, 13.11, 13.12, 14.8, 14.9, 15.3, 15.4	`rds.memory_allocation_guard`	`false`	`SET rds.memory_allocation_guard = true;`

Aurora PostgreSQL 버전	파라미터	Default	세션 수준에서 메모리 관리를 끄는 명령
12.17, 13.13, 14.10, 15.5 이상 버전	`rds.enable_memory_management`	`true`	`SET rds.enable_memory_management = false;`

> **Note**
>
> Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5 이상 버전에서는 `rds.memory_allocation_guard` 파라미터가 더 이상 사용되지 않습니다.

DB 클러스터 파라미터 그룹의 이러한 파라미터 값을 설정하면 쿼리가 취소되지 않습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 [Amazon Aurora의 파라미터 그룹](#) 섹션을 참조하세요.

제한 사항

- 이 기능은 db.t3.medium 인스턴스 클래스에서는 지원되지 않습니다.

Amazon Aurora PostgreSQL를 사용한 빠른 장애 조치

다음에서는 장애 조치가 가능한 한 빨리 발생하도록 하는 방법을 배울 수 있습니다. 장애 조치 후 신속하게 복구하기 위해 Aurora PostgreSQL DB 클러스터에 대한 클러스터 캐시 관리를 사용할 수 있습니다. 자세한 내용은 [장애 조치 후 Aurora PostgreSQL용 클러스터 캐시 관리를 통한 신속한 복구](#) 섹션을 참조하세요.

장애 조치가 빠르게 이루어지도록 하기 위해 수행할 수 있는 몇 가지 단계는 다음과 같습니다.

- 장애가 있는 경우 읽기 제한 시간이 만료되기 전에 더 긴 실행 쿼리를 중지하려면 짧은 시간 프레임으로 전송 제어 프로토콜(TCP) Keepalives를 설정합니다.
- Java 도메인 이름 시스템(DNS) 캐싱에 대한 제한 시간을 적극적으로 설정합니다. 이를 통해 Aurora 읽기 전용 엔드포인트가 나중에 연결을 시도할 때 읽기 전용 노드를 적절히 순환할 수 있습니다.
- JDBC 연결 문자열에 사용되는 제한 시간 변수를 가능한 낮게 설정합니다. 단기 및 장기 실행 중인 쿼리에 별도의 연결 객체를 사용합니다.

- 읽기 및 쓰기 Aurora 엔드포인트를 사용하여 클러스터에 연결합니다.
- RDS API 작업을 사용하여 서버 측 장애에 대한 애플리케이션 응답을 테스트합니다. 또한 패킷 삭제 도구를 사용하여 클라이언트 측 장애에 대한 애플리케이션 응답을 테스트합니다.
- AWS JDBC 드라이버를 사용하여 Aurora PostgreSQL의 장애 조치 기능을 최대한 활용합니다. AWS JDBC 드라이버에 대한 자세한 내용 및 사용 방법에 대한 전체 지침은 Amazon Web Services (AWS) JDBC Driver GitHub repository를 참조하세요.

이에 대해서는 다음에 이어지는 섹션에서 자세히 설명합니다.

주제

TCP Keepalives 파라미터 설정

TCP 연결을 설정하면 타이머 집합이 연결과 연결됩니다. keepalive 타이머가 0에 도달하면 keepalive 프로브 패킷을 연결 엔드포인트로 전송합니다. 프로브가 응답을 수신하면 연결이 계속 유지되는 것으로 가정할 수 있습니다.

TCP keepalive 파라미터를 활성화하고 적극적으로 설정하면 클라이언트가 더 이상 데이터베이스에 연결할 수 없을 경우 활성 연결이 빠르게 종료됩니다. 그러면 애플리케이션이 새 엔드포인트에 연결할 수 있습니다.

다음 TCP keepalive 파라미터를 설정해야 합니다.

- `tcp_keepalives_idle`은 시간을 초 단위로 제어하며, 이후 소켓으로부터 데이터가 전송되지 않을 경우 keepalive 패킷이 전송됩니다. ACK는 데이터로 간주되지 않습니다. 다음 설정을 권장합니다.

  ```
  tcp_keepalives_idle = 1
  ```

- `tcp_keepalives_interval`은 초기 패킷이 전송된 후 후속 keepalive 패킷을 전송하는 시간 주기를 초 단위로 제어합니다. `tcp_keepalives_idle` 파라미터를 사용하여 이 시간을 설정합니다. 다음 설정을 권장합니다.

  ```
  tcp_keepalives_interval = 1
  ```

- tcp_keepalives_count는 애플리케이션에 알림이 전달되기 전 발생하는 승인되지 않은 keepalive 프로브의 개수입니다. 다음 설정을 권장합니다.

  ```
  tcp_keepalives_count = 5
  ```

이러한 설정은 데이터베이스가 응답을 중단하고 5초 안에 애플리케이션에 알려야 합니다. keepalive 패킷이 애플리케이션의 네트워크 내에서 자주 삭제되는 경우 tcp_keepalives_count 값을 높게 설정할 수 있습니다. 이렇게 하면 실제 장애를 탐지하는 데 걸리는 시간이 늘어나지만 덜 안정적인 네트워크에서 더 많은 버퍼를 허용합니다.

Linux에서 TCP keepalive 파라미터를 설정하려면

1. TCP keepalive 파라미터가 어떻게 구성되어 있는지 테스트합니다.

   명령줄에서 다음 명령을 사용하여 이 작업을 수행하는 것이 좋습니다. 이 권장 구성은 시스템 전체에 적용됩니다. 즉, SO_KEEPALIVE 옵션이 켜진 상태에서 소켓을 생성하는 다른 모든 애플리케이션에도 영향을 미칩니다.

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```

2. 애플리케이션에 적합한 구성을 찾은 경우, 사용자가 변경한 내용을 포함해 /etc/sysctl.conf에 다음 줄을 추가하여 이러한 설정을 유지합니다.

   ```
   tcp_keepalive_time = 1
   tcp_keepalive_intvl = 1
   tcp_keepalive_probes = 5
   ```

애플리케이션에 빠른 장애 조치 구성

다음에서는 빠른 장애 조치를 위해 수행할 수 있는 Aurora PostgreSQL의 여러 구성 변경 사항에 대한 설명을 찾을 수 있습니다. PostgreSQL JDBC 드라이버 설정 및 구성에 대한 자세한 내용은 PostgreSQL JDBC Driver를 방문하여 확인하세요.

주제

- DNS 캐시 제한 시간 축소
- 빠른 장애 조치를 위한 Aurora PostgreSQL 연결 문자열 설정

- [호스트 문자열을 가져올 수 있는 다른 옵션]

DNS 캐시 제한 시간 축소

애플리케이션이 장애 조치 이후 연결을 설정할 때는 이전 리더가 새로운 Aurora PostgreSQL 라이터로 사용됩니다. 이전 리더는 DNS 업데이트가 완전히 전파되기 전에 Aurora 읽기 전용 엔드포인트를 사용해 찾을 수 있습니다. java DNS 유지 시간(TTL) 값을 30초 미만과 같이 낮게 설정하면 후속 연결 시도 시 리더 노드 사이에 순환이 이루어집니다.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

빠른 장애 조치를 위한 Aurora PostgreSQL 연결 문자열 설정

Aurora PostgreSQL 빠른 장애 조치를 사용하려면 애플리케이션의 연결 문자열에 단일 호스트가 아닌 호스트 목록이 있어야 합니다. 다음은 Aurora PostgreSQL 클러스터에 연결하는 데 사용할 수 있는 연결 문자열의 예입니다. 이 예에서는 호스트가 굵게 표시됩니다.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

최상의 가용성과 RDS API에 대한 종속성을 피하기 위해 연결할 파일을 유지 관리하는 것이 좋습니다. 이 파일에는 데이터베이스에 연결할 때 애플리케이션이 읽는 호스트 문자열이 포함되어 있습니다. 이 호스트 문자열에는 클러스터에 제공된 모든 Aurora 엔드포인트가 있습니다. Aurora 엔드포인트에 대한 자세한 내용은 [Amazon Aurora 엔드포인트 연결] 섹션을 참조하세요.

예를 들어, 다음과 같이 로컬 파일에 엔드포인트를 저장할 수 있습니다.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

애플리케이션이 이 파일에서 읽어 들여 JDBC 연결 문자열의 호스트 섹션을 채웁니다. DB 클러스터 이름을 바꾸면 이러한 엔드포인트가 변경됩니다. 이벤트가 발생할 경우 애플리케이션이 이 이벤트를 처리하는지 확인하세요.

다른 방법은 다음과 같이 DB 인스턴스 노드 목록을 사용하는 것입니다.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

이 방식의 장점은 PostgreSQL JDBC 연결 드라이버가 이 목록에서 모든 노드를 통해 순환하며 유효한 연결을 찾는다는 것입니다. 이와는 반대로 Aurora 엔드포인트를 사용하는 경우 각 연결 시도에서 두 개의 노드만 시도됩니다. 하지만 DB 인스턴스 노드를 사용하는 데에는 단점이 있습니다. 사용자가 클러스터에 노드를 추가하거나 제거하고 인스턴스 엔드포인트 목록이 무효로 되면 연결 드라이버가 연결할 올바른 호스트를 찾을 수 없다는 것입니다.

애플리케이션이 한 호스트에 연결하기 위해 너무 오래 기다리지 않도록 하려면 다음 파라미터를 적극적으로 설정합니다.

- `targetServerType` – 드라이버가 쓰기 노드에 연결되는지 아니면 읽기 노드에 연결되는지를 제어합니다. 애플리케이션이 쓰기 노드에만 다시 연결되도록 하려면 `targetServerType` 값을 `primary`로 설정합니다.

  `targetServerType` 파라미터에 사용할 수 있는 값에는 `primary`, `secondary`, any, `preferSecondary`가 있습니다. 이 `preferSecondary` 값은 먼저 리더와의 연결 설정을 시도합니다. 리더 연결을 설정할 수 없는 경우 라이터에 연결합니다.
- `loginTimeout` – 소켓 연결이 설정된 후 애플리케이션이 데이터베이스에 로그인하기까지 대기하는 시간을 제어합니다.
- `connectTimeout` – 소켓이 데이터베이스에 연결을 설정하기까지 대기하는 시간을 제어합니다.

애플리케이션을 얼마나 적극적으로 설정하고 싶은지에 따라, 다른 애플리케이션 파라미터를 수정하여 연결 프로세스의 속도를 높일 수 있습니다.

- `cancelSignalTimeout` – 일부 애플리케이션에서는 시간 초과 쿼리에 "최대한" 취소 신호를 보내야 할 수 있습니다. 이 취소 신호가 장애 조치 경로에 있는 경우, 잘못된 호스트로 이 신호가 전달되지 않도록 적극적으로 설정하는 것이 좋습니다.
- `socketTimeout` – 이 파라미터는 소켓이 읽기 작업을 대기하는 시간을 제어합니다. 쿼리가 이 값보다 길게 대기하지 않도록 이 파라미터를 글로벌 "쿼리 제한 시간"으로 사용할 수 있습니다. 연결 핸들러를 두 개 사용하는 것이 좋습니다. 하나의 연결 핸들러는 수명이 짧은 쿼리를 실행하고 이 값을 낮게 설정합니다. 장기 실행 쿼리를 위한 또 다른 연결 핸들러에서는 이 값이 훨씬 더 높게 설정되어 있

습니다. 이 방식으로는 서버가 다운될 경우 TCP keepalive 파라미터를 이용해 장기 실행 쿼리를 중단할 수 있습니다.

- `tcpKeepAlive` – 이 파라미터를 켜면 사용자가 설정한 TCP keepalive 파라미터가 적용되도록 합니다.
- `loadBalanceHosts` – 이 파라미터를 `true`로 설정하면 애플리케이션을 후보 호스트 목록에서 선택한 임의의 호스트에 연결합니다.

호스트 문자열을 가져올 수 있는 다른 옵션

`aurora_replica_status` 함수 및 Amazon RDS API 사용을 비롯해 여러 소스로부터 호스트 문자열을 가져올 수 있습니다.

대부분의 경우 클러스터 라이터를 확인하거나 클러스터의 다른 리더 노드를 찾아야 합니다. 이를 위해 애플리케이션에서 DB 클러스터의 DB 인스턴스에 연결하고 `aurora_replica_status` 함수를 쿼리할 수 있습니다. 이 함수를 사용하면 연결할 호스트를 찾는 데 걸리는 시간을 줄일 수 있습니다. 그러나 특정 네트워크 장애 시나리오에서는 `aurora_replica_status` 기능이 오래되었거나 불완전한 정보를 표시할 수 있습니다.

애플리케이션이 연결할 노드를 찾도록 보장하는 좋은 방법 중 하나는클러스터 라이터 엔드포인트에 연결한 다음 클러스터 리더 엔드포인트에 연결을 시도하는 것입니다. 읽기 가능한 연결을 설정할 때까지 이 작업을 수행합니다. 이러한 엔드포인트는 DB 클러스터의 이름을 변경하지 않는 한 변경되지 않습니다. 따라서 일반적으로 이들을 애플리케이션의 정적 멤버로 남겨두거나 애플리케이션이 읽는 리소스 파일에 저장할 수 있습니다.

이러한 엔드포인트 중 하나를 사용하여 연결을 설정한 후 클러스터의 나머지 부분에 대한 정보를 얻을 수 있습니다. 이렇게 하려면 `aurora_replica_status` 함수를 호출합니다. 예를 들어 다음 명령은 `aurora_replica_status`로 정보를 검색합니다.

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+------------------
+-----------------------------+----------------------------------+------------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
```

```
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

이를테면 연결 문자열의 호스트 섹션은 라이터와 리더 클러스터 엔드포인트 모두로 시작할 수 있습니다.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

이 시나리오에서 애플리케이션은 기본 또는 보조 노드 유형에 연결을 설정하려 할 것입니다. 애플리케이션이 연결되면 먼저 명령 의 결과를 쿼리하여 노드의 읽기/쓰기 상태를 검사하는 것이 좋습니다. 이렇게 하려면 `SHOW transaction_read_only` 명령 결과를 쿼리해야 합니다.

쿼리의 반환 값이 `OFF`인 경우 프라이머리 노드에 성공적으로 연결된 것입니다. 그러나 반환 값이 `ON`이고 애플리케이션에 읽기/쓰기 연결이 필요하다고 가정합니다. 이 경우 `aurora_replica_status` 함수를 호출하여 `session_id='MASTER_SESSION_ID'`인 `server_id`를 확인할 수 있습니다. 이 함수는 프라이머리 노드의 이름을 반환합니다. 다음에 설명하는 `endpointPostfix`와 함께 사용할 수 있습니다.

단, 오래된 데이터가 있는 복제본에 연결하는 경우에는 주의해야 합니다. 이 경우 `aurora_replica_status` 함수에 오래된 정보가 표시될 수 있습니다. 애플리케이션 수준에서 부실 임계값을 설정할 수 있습니다. 이를 확인하기 위해 서버 시간과 `last_update_timestamp` 값의 차이를 확인할 수 있습니다. 일반적으로 애플리케이션은 `aurora_replica_status` 함수에 의해 반환된 충돌 정보로 인한 두 호스트 사이의 대칭 이동을 방지해야 합니다. 애플리케이션은 `aurora_replica_status`에서 반환된 데이터를 따르는 대신 알려진 모든 호스트를 먼저 시도해야 합니다.

DescribeDBClusters API 작업을 사용하여 인스턴스를 나열(Java 예제)

[AWS SDK for Java](), 특히 [DescribeDBClusters]() API 작업을 사용하여 프로그래밍 방식으로 인스턴스 목록을 찾을 수 있습니다.

다음은 java 8에서 이와 같이 검색하는 방법에 대한 작은 예입니다.

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);
```

```
DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

여기에서 `pgJDBCEndpointStr`은 다음과 같이 형식이 지정된 끝점 목록을 포함합니다.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

변수 `endpointPostfix`는 애플리케이션이 설정하는 상수일 수 있습니다. 또는 클러스터의 단일 인스턴스에 대해 `DescribeDBInstances` API 작업을 쿼리하여 애플리케이션에서 이를 가져올 수 있습니다. 이 값은 AWS 리전 내에서 그리고 개별 고객에 대해 일정하게 유지됩니다. 따라서 API 호출을 저장하여 애플리케이션이 읽어 들이는 리소스 파일에서 이 상수를 유지합니다. 앞의 예에서는 다음과 같이 설정됩니다.

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

API가 응답하지 않거나 응답하는 데 너무 오래 걸리는 경우 가용성을 위해 기본적으로 DB 클러스터의 Aurora 엔드포인트를 사용하는 것이 좋습니다. 엔드포인트는 DNS 레코드를 업데이트하는 데 걸리는 시간 내에 최신 상태가 됩니다. DNS 레코드를 엔드포인트로 업데이트하는 데 보통 30초도 걸리지 않습니다. 이는 애플리케이션이 사용하는 리소스 파일에 저장할 수 있습니다.

장애 조치 테스트

모든 경우에 두 개 이상의 DB 인스턴스를 포함한 DB 클러스터가 있어야 합니다.

서버 측에서 특정 API가 애플리케이션의 응답 방식을 테스트하는 데 사용할 수 있는 중단을 유발할 수 있습니다.

- FailoverDBCluster - 이 작업은 DB 클러스터의 새 DB 인스턴스를 기록기로 승격하려고 시도합니다.

  다음 코드 예에서는 `failoverDBCluster`를 사용하여 중단을 일으키는 방법을 보여줍니다. Amazon RDS 클라이언트 설정에 대한 자세한 내용은 AWS SDK for Java 사용을 참조하세요.

```
public void causeFailover() {

    final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

    FailoverDBClusterRequest request = new FailoverDBClusterRequest();
    request.setDBClusterIdentifier("cluster-identifier");

    rdsClient.failoverDBCluster(request);
}
```

- RebootDBInstance – 이 API 작업에서는 장애 조치가 보장되지 않습니다. 하지만 라이터의 데이터베이스는 종료됩니다. 이를 사용하여 애플리케이션이 연결 끊기에 어떻게 반응하는지 테스트할 수 있습니다. `ForceFailover` 파라미터는 Aurora 엔진에 적용되지 않습니다. 대신 `FailoverDBCluster` API 작업을 사용합니다.
- ModifyDBCluster – `Port` 파라미터를 변경하면 클러스터의 노드가 새 포트에서 수신을 시작할 경우 중단을 유발합니다. 일반적으로 애플리케이션만 포트 변경을 제어하도록 하여 이 실패에 먼저 응답할 수 있습니다. 또한 종속된 엔드포인트를 적절하게 업데이트할 수 있는지 확인하세요. 이렇게 하려면 API 수준에서 수정한 사람이 포트를 수동으로 업데이트하도록 하면 됩니다. 또는 애플리케이션의 RDS API를 사용하여 포트가 변경되었는지 확인할 수 있습니다.
- ModifyDBInstance - `DBInstanceClass` 파라미터를 수정하면 중단이 발생합니다.
- DeleteDBInstance – 기본(라이터)을 삭제하면 새 DB 인스턴스가 DB 클러스터의 라이터로 승격됩니다.

애플리케이션 또는 클라이언트 측에서 Linux를 사용하는 경우 애플리케이션이 갑작스러운 패킷 드롭에 응답하는 방식을 테스트할 수 있습니다. 포트, 호스트 또는 iptables 명령을 사용하여 TCP keepalive 패킷을 보내거나 받는지 여부에 따라 이 작업을 수행할 수 있습니다.

Java의 빠른 장애 조치 예

다음 코드 예는 애플리케이션이 Aurora PostgreSQL 드라이버 관리자를 설정하는 방식을 보여줍니다.

애플리케이션이 연결해야 할 때 `getConnection` 함수를 호출합니다. `getConnection` 호출은 유효한 호스트를 찾지 못할 수 있습니다. 라이터를 찾을 수 없지만 `targetServerType` 파라미터가 `primary`로 설정된 경우를 예로 들 수 있습니다. 이 경우 호출 애플리케이션에서 함수 호출을 다시 시도해야 합니다.

재시도 동작을 애플리케이션에 푸시하지 않으려면 이 재시도 호출을 연결 풀러로 래핑할 수 있습니다. 대부분의 연결 풀러에서는 JDBC 연결 문자열을 지정할 수 있습니다. 따라서 애플리케이션은

`getJdbcConnectionString`을 호출하고 이를 연결 풀러에 전달할 수 있습니다. 이렇게 하면 Aurora PostgreSQL에서 더 빠른 장애 조치를 사용할 수 있습니다.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
   }

   public Connection getConnection(String targetServerType) throws SQLException {
       return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
   }
```

```
    public Connection getConnection(String targetServerType, Duration queryTimeout)
 throws SQLException {
        Connection conn =
 DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

       /*
         * A good practice is to set socket and statement timeout to be the same thing
 since both
         * the client AND server will stop the query at the same time, leaving no
 running queries
         * on the backend
         */
        Statement st = conn.createStatement();
        st.execute("set statement_timeout to " + queryTimeout.getMillis());
        st.close();

       return conn;
   }

   private static String urlFormat = "jdbc:postgresql://%s"
           + "/postgres"
           + "?user=%s"
           + "&password=%s"
           + "&loginTimeout=%d"
           + "&connectTimeout=%d"
           + "&cancelSignalTimeout=%d"
           + "&socketTimeout=%d"
           + "&targetServerType=%s"
           + "&tcpKeepAlive=true"
           + "&ssl=true"
           + "&loadBalanceHosts=true";
   public String getJdbcConnectionString(String targetServerType, Duration
 queryTimeout) {
       return String.format(urlFormat,
               getFormattedEndpointList(getLocalEndpointList()),
               CredentialManager.getUsername(),
               CredentialManager.getPassword(),
               LOGIN_TIMEOUT.getStandardSeconds(),
               CONNECT_TIMEOUT.getStandardSeconds(),
               CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
               queryTimeout.getStandardSeconds(),
               targetServerType
       );
   }
```

```
    private List<String> getLocalEndpointList() {
        /*
          * As mentioned in the best practices doc, a good idea is to read a local
  resource file and parse the cluster endpoints.
          * For illustration purposes, the endpoint list is hardcoded here
          */
         List<String> newEndpointList = new ArrayList<>();
         newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
         newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

장애 조치 후 Aurora PostgreSQL용 클러스터 캐시 관리를 통한 신속한 복구

장애 조치가 발생한 경우 Aurora PostgreSQL 클러스터에 라이터 DB 인스턴스를 신속하게 복구하려면 Amazon Aurora PostgreSQL용 클러스터 캐시 관리를 사용하십시오. 장애 조치가 발생한 경우 클러스터 캐시 관리를 통해 애플리케이션 성능을 유지할 수 있습니다.

일반적인 장애 조치 상황에서는 장애 조치 후 일시적이지만 큰 성능 저하를 겪을 수 있습니다. 이러한 현상이 일어나는 이유는 장애 조치 DB 인스턴스가 시작될 때 버퍼 캐시가 비어 있기 때문입니다. 빈 캐시는 콜드 캐시라고도 합니다. 콜드 캐시로 인해 성능이 저하되는 이유는 DB 인스턴스가 버커 캐시에 저장된 값을 이용하는 대신에 속도가 느려진 디스크에서 읽어야 하기 때문입니다.

클러스터 캐시 관리를 통해 특정 리더 DB 인스턴스를 장애 조치 대상으로 설정하십시오. 클러스터 캐시 관리는 지정된 리더의 캐시가 라이터 DB 인스턴스의 캐시에 있는 데이터와 동기화된 상태를 유지하도록 보장합니다. 미리 채워진 값이 있는 지정된 리더의 캐시는 웜 캐시라고도 합니다. 장애 조치가 발생하면 지정된 리더는 새로운 라이터 DB 인스턴스로 승격될 때 즉시 웜 캐시에 있는 값을 사용합니다. 이러한 접근 방식을 통해 애플리케이션의 복구 성능이 대폭 향상됩니다.

클러스터 캐시 관리를 사용하려면 지정된 리더 인스턴스의 인스턴스 클래스 유형 및 크기(예: `db.r5.2xlarge` 또는 `db.r5.xlarge`)가 라이터와 같아야 합니다. Aurora PostgreSQL DB 클러스터

를 생성할 때 이 점을 염두에 두어 장애 조치 중에 클러스터가 복구될 수 있도록 해야 합니다. 인스턴스 클래스 유형 및 크기 목록은 Aurora용 DB 인스턴스 클래스의 하드웨어 사양을 참조하세요.

> **Note**
>
> Aurora 글로벌 데이터베이스의 일부인 Aurora PostgreSQL 보조 DB 클러스터에는 클러스터 캐시 관리가 지원되지 않습니다. 이 기능이 지원되는 프라이머리 클러스터의 경우 지정된 tier-0 리더에서 워크로드를 실행해서는 안 됩니다.

목차

클러스터 캐시 관리 구성

클러스터 캐시 관리를 구성하려면 다음 프로세스를 순서대로 수행합니다.

주제

> **Note**
>
> 구성 절차를 마친 후 클러스터 캐시 관리 기능이 완전하게 실행되려면 최소 1분이 지나야 합니다.

클러스터 캐시 관리 활성화

클러스터 캐시 관리를 활성화하려면 아래 설명된 단계를 수행합니다.

콘솔

클러스터 캐시 관리를 활성화하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 Aurora PostgreSQL DB 클러스터의 파라미터 그룹을 선택합니다.

   DB 클러스터에서는 기본값이 아닌 파라미터 그룹을 사용해야 합니다. 기본 파라미터 그룹에서는 값을 변경할 수 없기 때문입니다.
4. 파라미터 그룹 작업에서 편집을 선택합니다.
5. `apg_ccm_enabled` 클러스터 파라미터의 값을 1로 설정합니다.
6. 변경 사항 저장을 선택합니다.

AWS CLI

Aurora PostgreSQL DB 클러스터에 대해 클러스터 캐시 관리를 사용 설정하려면 다음 필수 파라미터와 함께 AWS CLI `modify-db-cluster-parameter-group` 명령을 사용합니다.

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

라이터 DB 인스턴스에 대한 승격 티어 우선 순위 설정

클러스터 캐시 관리를 위해 Aurora PostgreSQL DB 클러스터의 라이터 DB 인스턴스에 대한 승격 우선 순위는 tier-0이어야 합니다. 승격 티어 우선 순위는 장애 조치 후 Aurora 리더가 라이터 DB 인스턴스로 승격할 순서를 지정하는 값입니다. 유효한 값은 0–15이며, 여기에서 0은 최우선 순위이고 15는 마지막 우선 순위입니다. 승격 티어에 대한 자세한 내용은 [Aurora DB 클러스터의 내결함성](#) 단원을 참조하십시오.

콘솔

라이터 DB 인스턴스에 대해 승격 우선 순위를 설정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Aurora PostgreSQL DB 클러스터의 쓰기 DB 인스턴스를 선택합니다.
4. 수정을 선택합니다. [Modify DB Instance] 페이지가 나타납니다.
5. 추가 구성 패널에서 장애 조치 우선 순위로 tier-0을 선택합니다.
6. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
7. 변경 사항을 저장한 후 즉시 적용하려면 즉시 적용을 선택합니다.
8. DB 인스턴스 수정을 선택하여 변경 사항을 저장합니다.

AWS CLI

AWS CLI를 사용해 라이터 DB 인스턴스에 대한 승격 티어 우선 순위를 0으로 설정하려면 다음 필수 파라미터와 함께 modify-db-instance 명령을 호출하십시오.

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
```

```
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

리더 DB 인스턴스에 대한 승격 티어 우선 순위 설정

클러스터 캐시 관리에 대해 리더 DB 인스턴스를 하나만 설정합니다. 이를 위해서는 라이터 DB 인스턴스와 동일한 인스턴스 클래스와 크기인 Aurora PostgreSQL 클러스터에서 리더를 선택해야 합니다. 예를 들어, 라이터에서 `db.r5.xlarge`을(를) 사용하면 동일한 인스턴스 클래스 유형 및 크기를 사용하는 리더를 선택합니다. 그런 다음 리더의 승격 티어 우선 순위를 0으로 설정합니다.

승격 티어 우선 순위는 장애 조치 후 Aurora 리더가 라이터 DB 인스턴스로 승격할 순서를 지정하는 값입니다. 유효한 값은 0–15이며, 여기에서 0은 최우선순위이고 15는 마지막 우선순위입니다.

콘솔

라이터 DB 인스턴스의 승격 우선 순위를 tier-0으로 설정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 라이터 DB 인스턴스와 동일한 인스턴스 클래스인 Aurora PostgreSQL DB 클러스터의 읽기 DB 인스턴스를 선택합니다.
4. 수정을 선택합니다. [Modify DB Instance] 페이지가 나타납니다.
5. 추가 구성 패널에서 장애 조치 우선 순위로 tier-0을 선택합니다.
6. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
7. 변경 사항을 저장한 후 즉시 적용하려면 즉시 적용을 선택합니다.
8. DB 인스턴스 수정을 선택하여 변경 사항을 저장합니다.

AWS CLI

AWS CLI를 사용해 리더 DB 인스턴스에 대한 승격 티어 우선 순위를 0으로 설정하려면 다음 필수 파라미터와 함께 [modify-db-instance](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-instance.html) 명령을 호출하십시오.

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

버퍼 캐시 모니터링

클러스터 캐시 관리를 설정하였으면 이제 라이터 DB 인스턴스의 버퍼 캐시와 지정된 리더의 웜 버퍼 캐시 사이의 동기화 상태를 모니터링할 수 있습니다. 라이터 DB 인스턴스와 지정된 리더 DB 인스턴스에 있는 버퍼 캐시 콘텐츠를 검토하려면 PostgreSQL `pg_buffercache` 모듈을 사용하십시오. 자세한 내용은 PostgreSQL `pg_buffercache` 설명서를 참조하십시오.

`aurora_ccm_status` 함수 사용

클러스터 캐시 관리는 `aurora_ccm_status` 함수도 제공합니다. 라이터 DB 인스턴스에서 `aurora_ccm_status` 함수를 사용해 지정된 리더에서 캐시 워밍이 진행되는 과정에 대해 다음과 같은 정보를 얻으십시오.

- buffers_sent_last_minute – 마지막 순간에 지정된 리더로 전송된 버퍼의 수.
- buffers_found_last_minute - 지난 1분 동안 식별된 자주 액세스된 버퍼 수입니다.
- buffers_sent_last_scan – 버퍼 캐시에 대한 마지막 전체 스캔 중에 지정된 리더로 전송된 버퍼의 수.
- buffers_found_last_scan – 자주 액세스되는 것으로 식별되어 버퍼 캐시에 대한 마지막 전체 스캔 중에 전송되어야 했던 버퍼의 수. 지정된 리더에 이미 캐시된 버퍼는 전송되지 않습니다.
- buffers_sent_current_scan – 현재 스캔 중에 지금까지 전송된 버퍼의 수.
- buffers_found_current_scan – 현재 스캔 중에 자주 액세스되는 것으로 식별된 버퍼의 수.
- current_scan_progress – 현재 스캔 중에 지금까지 방문을 받은 버퍼의 수.

다음 예에서는 aurora_ccm_status 함수를 사용해 출력 중 일부를 웜 속도 및 웜 비율로 변환하는 방법을 보여줍니다.

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

CCM 구성 문제 해결

apg_ccm_enabled 클러스터 파라미터를 활성화하면 라이터 DB 인스턴스의 인스턴스 수준 및 Aurora PostgreSQL DB 클러스터의 리더 DB 인스턴스 1개에서 클러스터 캐시 관리가 자동으로 설정됩니다. 라이터 및 리더 인스턴스는 동일한 인스턴스 클래스 유형과 크기를 사용해야 합니다. 프로모션 티어 우선 순위는 0으로 설정되어 있습니다. DB 클러스터의 다른 리더 인스턴스는 0이 아닌 승격 티어가 있어야 하며 해당 인스턴스에 대해서는 클러스터 캐시 관리가 비활성화되어 있습니다.

다음과 같은 이유로 구성 문제가 발생하고 클러스터 캐시 관리가 비활성화될 수 있습니다.

- 승격 티어가 0으로 설정된 리더 DB 인스턴스가 하나도 없는 경우
- 라이터 DB 인스턴스의 승격 티어가 0으로 설정되지 않은 경우
- 2개 이상의 리더 DB 인스턴스에서 승격 티어가 0으로 설정된 경우
- 승격 티어가 0인 라이터와 리더 DB 인스턴스 1개의 인스턴스 크기가 같지 않은 경우

풀링으로 Aurora PostgreSQL 연결 이탈 관리

클라이언트 애플리케이션이 너무 자주 연결 및 연결 해제되어 Aurora PostgreSQL DB 클러스터 응답 시간이 느려지면 클러스터에 연결 이탈이 발생한다고 합니다. Aurora PostgreSQL DB 클러스터 엔드

포인트에 대한 각각의 새로운 연결은 리소스를 소비하므로 실제 워크로드를 처리하는 데 사용할 수 있는 리소스가 줄어듭니다. 연결 이탈은 다음에 설명하는 모범 사례 중 일부를 따라 관리하는 것이 좋습니다.

우선 연결 이탈률이 높은 Aurora PostgreSQL DB 클러스터에서 응답 시간을 개선할 수 있습니다. 이렇게 하려면 RDS 프록시와 같은 연결 풀을 사용할 수 있습니다. 연결 풀러는 클라이언트에 사용할 수 있는 연결 캐시를 제공합니다. Aurora PostgreSQL에서는 거의 모든 버전이 RDS 프록시를 지원합니다. 자세한 내용은 Aurora PostgreSQL과 Amazon RDS 프록시 단원을 참조하십시오.

특정 버전의 Aurora PostgreSQL에서 RDS 프록시를 지원하지 않는 경우 PGBouncer와 같은 다른 PostgreSQL 호환 연결 풀러를 사용할 수 있습니다. 자세히 알아보려면 PgBouncer 웹 사이트를 참조하세요.

Aurora PostgreSQL DB 클러스터가 연결 풀링의 이점을 누릴 수 있는지 알아보려면 `postgresql.log` 파일에서 연결 및 연결 해제를 확인하면 됩니다. 또한 성능 개선 도우미를 사용하여 Aurora PostgreSQL DB 클러스터에서 발생하는 연결 이탈의 정도를 확인할 수 있습니다. 아래에서 두 주제에 대한 정보를 확인할 수 있습니다.

로깅 연결 및 연결 해제

PostgreSQL `log_connections` 및 `log_disconnections` 파라미터는 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 연결 및 연결 해제를 캡처할 수 있습니다. 기본적으로 이 파라미터는 해제되어 있습니다. 이러한 파라미터를 설정하려면 사용자 지정 파라미터 그룹을 사용하고 값을 1로 변경하여 설정합니다. 사용자 지정 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 단원을 참조하세요. 설정을 확인하려면 psql을 사용하여 Aurora PostgreSQL용 DB 클러스터 엔드포인트에 연결하고 다음과 같이 쿼리합니다.

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

이 두 파라미터가 모두 설정되어 있으면 로그가 모든 새 연결 및 연결 해제를 캡처합니다. 인증된 각 연결의 사용자 및 데이터베이스를 볼 수 있습니다. 연결이 끊기면 다음 예와 같이 세션 기간도 기록됩니다.

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

애플리케이션의 연결 이탈을 확인하기 위해 해당 파라미터가 아직 설정되어 있지 않다면 설정하세요. 그런 다음 실제 워크로드 및 기간으로 애플리케이션을 실행하여 분석을 위해 PostgreSQL 로그에 데이터를 수집합니다. RDS 콘솔에서 로그 파일을 볼 수 있습니다. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택한 다음 Logs & events(로그 및 이벤트) 탭을 선택합니다. 자세한 내용은 데이터베이스 로그 파일 보기 및 나열 단원을 참조하십시오.

또는 콘솔에서 로그 파일을 다운로드하고 다음 명령 순서를 사용할 수 있습니다. 이 시퀀스는 분당 인증되거나 삭제된 총 연결 수를 찾습니다.

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

예제 출력에서는 인증된 연결이 급증한 후 16:12:10 부터 연결이 끊기는 것을 볼 수 있습니다.

```
.....
,......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
```

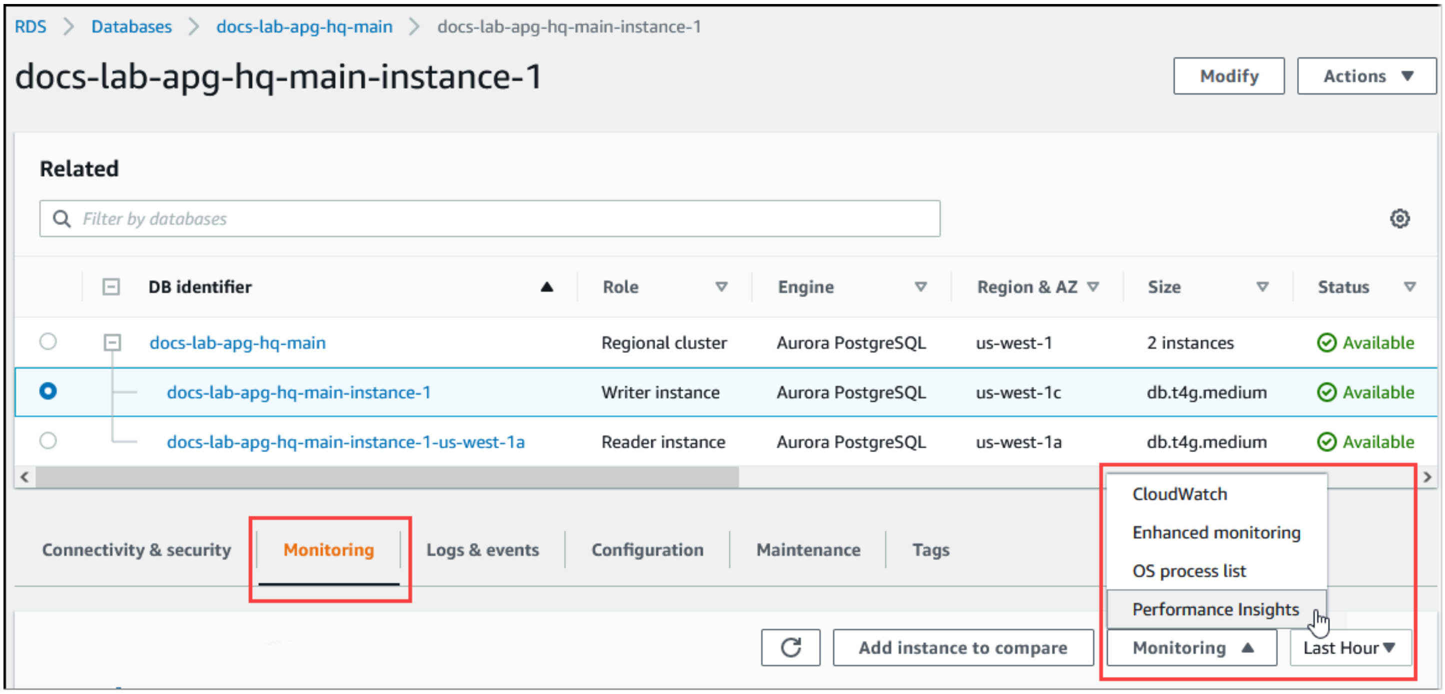
```
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
1 2022-03-21 16:16:00 connection authorized:
```

이 정보를 바탕으로 워크로드가 연결 풀러의 이점을 누릴 수 있는지 여부를 결정할 수 있습니다. 자세한 분석을 위해 성능 개선 도우미를 사용할 수 있습니다.

성능 개선 도우미로 연결 이탈 감지

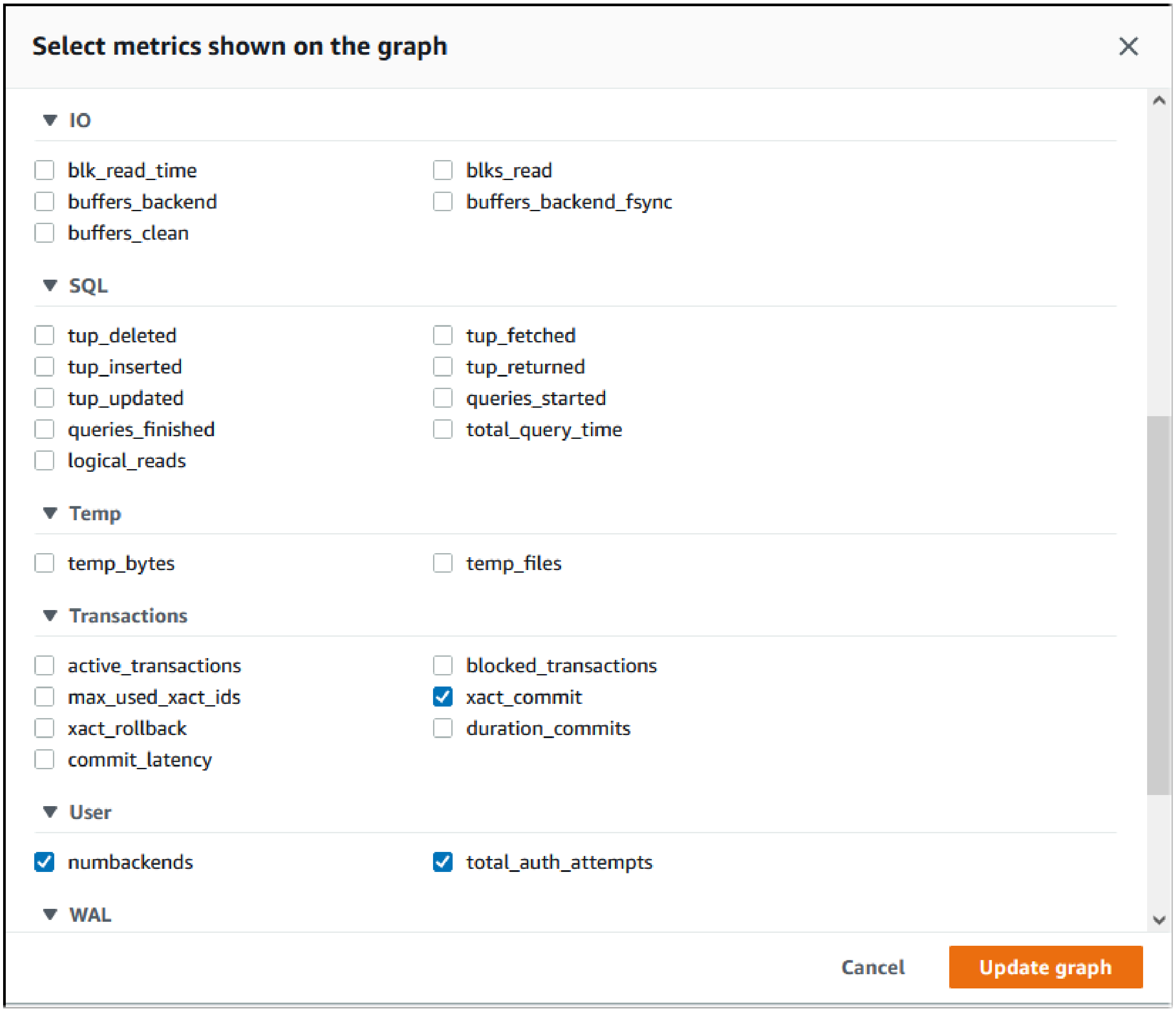
성능 개선 도우미를 사용하여 Aurora PostgreSQL 호환 에디션 DB 클러스터의 연결 변동 정도를 평가할 수 있습니다. Aurora PostgreSQL DB 클러스터를 생성하면 성능 개선 도우미에 대한 설정이 기본적으로 켜져 있습니다. DB 클러스터를 생성할 때 이 선택을 취소했다면 클러스터를 수정하여 기능을 켜세요. 자세한 내용은 Amazon Aurora DB 클러스터 수정 단원을 참조하십시오.

Aurora PostgreSQL DB 클러스터에서 성능 개선 도우미를 실행하면 모니터링할 지표를 선택할 수 있습니다. 콘솔의 탐색 창에서 성능 개선 도우미에 액세스할 수 있습니다. 다음 이미지와 같이 Aurora PostgreSQL DB 클러스터에 대한 라이터 인스턴스의 Monitoring(모니터링) 탭에서 성능 개선 도우미에 액세스할 수도 있습니다.



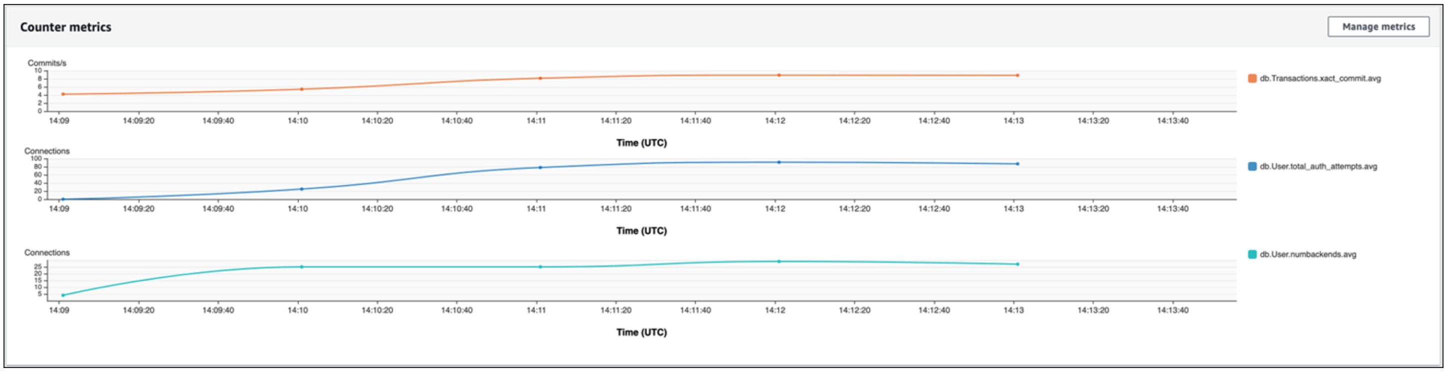
성능 개선 도우미 콘솔에서 Manage metrics(지표 관리)를 선택합니다. Aurora PostgreSQL DB 클러스터의 연결 및 연결 해제 활동을 분석하려면 다음 지표를 선택하세요. 다음은 모두 PostgreSQL의 지표입니다.

- `xact_commit` - 커밋된 트랜잭션의 수입니다.
- `total_auth_attempts` - 분당 인증된 사용자 연결 시도 횟수입니다.
- `numbackends` - 현재 데이터베이스에 연결된 백엔드 수입니다.



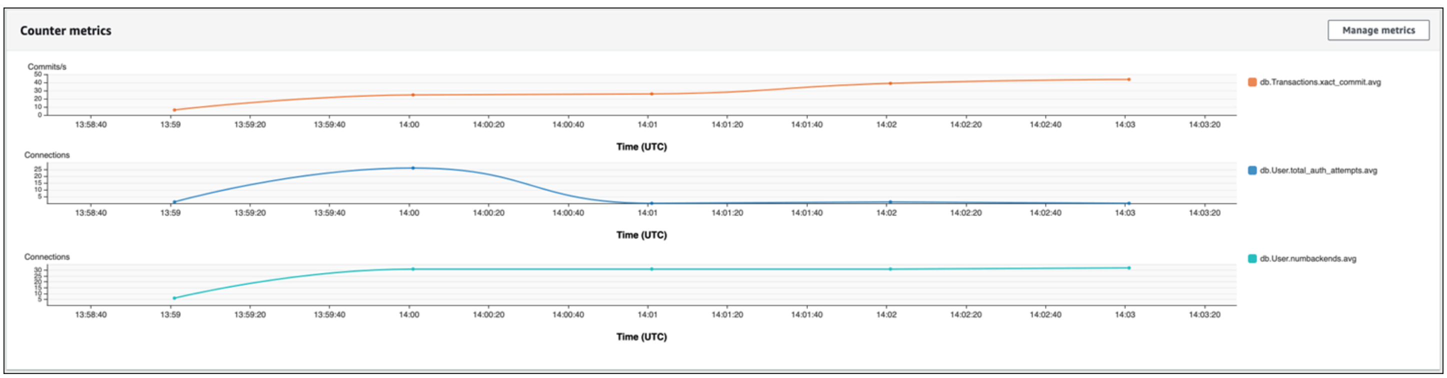
설정을 저장하고 연결 활동을 표시하려면그래프 업데이트를 선택합니다.

다음 이미지에서 100명의 사용자와 함께 pgbench를 실행할 때의 영향을 확인할 수 있습니다. 연결을 나타내는 선은 일정한 상향 경사면에 있습니다. pgbench 및 사용 방법에 대한 자세한 내용은 PostgreSQL 설명서의 pgbench를 참조하세요.



이 이미지는 연결 풀러 없이 100명 정도의 사용자로 워크로드를 실행하면 워크로드 처리 기간 동안 `total_auth_attempts` 수가 크게 증가할 수 있음을 보여줍니다. `total_auth_attempts`가 가능한 한 0에 가깝게 유지되도록 하는 것이 좋습니다.

RDS 프록시 연결 풀링을 사용하면 워크로드가 시작될 때 연결 시도 횟수가 늘어납니다. 연결 풀을 설정한 후에는 평균이 감소합니다. 트랜잭션과 백엔드 사용에 사용되는 리소스는 워크로드 처리 전반에 걸쳐 일관되게 유지됩니다.



Aurora PostgreSQL DB 클러스터에서 성능 개선 도우미를 사용하는 방법에 대한 자세한 내용은 [성능 개선 도우미를 통한 Amazon Aurora 모니터링](#) 단원을 참조하세요. 지표를 분석하려면 [성능 개선 도우미 대시보드를 사용한 지표 분석](#) 단원을 참조하세요.

연결 풀링의 이점 시연

앞서 설명한 것처럼 Aurora PostgreSQL DB 클러스터에 연결 이탈 문제가 있다고 판단되면 RDS 프록시를 사용하여 성능을 개선할 수 있습니다. 다음에서는 연결이 풀링된 경우와 연결되지 않은 경우의 워크로드 처리 차이점을 보여주는 예를 확인할 수 있습니다. 이 예에서는 pgbench를 사용하여 트랜잭션 워크로드를 모델링합니다.

psql과 마찬가지로 pgbench는 로컬 클라이언트 컴퓨터에서 설치하고 실행할 수 있는 PostgreSQL 클라이언트 애플리케이션입니다. Aurora PostgreSQL DB 클러스터를 관리하는 데 사용하는 Amazon

EC2 인스턴스에서 설치하고 실행할 수도 있습니다. 자세한 내용은 PostgreSQL 설명서의 pgbench를 참조하세요.

이 예를 단계별로 진행하려면 먼저 데이터베이스에 pgbench 환경을 만들어야 합니다. 다음 명령은 지정된 데이터베이스에서 pgbench 테이블을 초기화하기 위한 기본 템플릿입니다. 이 예에서는 로그인에 기본 주 사용자 계정인 `postgres`를 사용합니다. Aurora PostgreSQL DB 클러스터에서 필요에 따라 변경합니다. 클러스터의 라이터 인스턴스에 있는 데이터베이스에 pgbench 환경을 생성합니다.

> **Note**
>
> pgbench 초기화 프로세스는 `pgbench_accounts`,`pgbench_branches`,`pgbench_history`, 및 `pgbench_tellers`라는 테이블을 삭제하고 다시 만듭니다. pgbench를 초기화할 때 *dbname*에 대해 선택한 데이터베이스가 이러한 이름을 사용하지 않는지 확인합니다.

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

pgbench의 경우 다음 파라미터를 지정합니다.

-d

pgbench 실행 시 디버깅 보고서를 출력합니다.

-h

Aurora PostgreSQL DB 클러스터의 라이터 인스턴스의 엔드포인트를 지정합니다.

-i

벤치마크 테스트를 위해 데이터베이스에서 pgbench 환경을 초기화합니다.

-p

데이터베이스 연결에 사용되는 포트를 식별합니다. Aurora PostgreSQL 기본값은 일반적으로 5432 또는 5433입니다.

-s

테이블을 행으로 채우는 데 사용할 배율 인수를 지정합니다. 기본 배율 인수는 1이며, `pgbench_branches` 테이블에 1개 행, `pgbench_tellers` 테이블에 10개 행, `pgbench_accounts` 테이블에 100000개 행을 생성합니다.

-U

Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 사용자 계정을 지정합니다.

pgbench 환경이 설정되면 연결 풀링을 사용하거나 사용하지 않고 벤치마킹 테스트를 실행할 수 있습니다. 기본 테스트는 지정된 시간 동안 반복적으로 실행되는 트랜잭션당 5개의 SELECT, UPDATE 및 INSERT 명령 시리즈로 구성됩니다. 배율 인수, 클라이언트 수 및 기타 세부 정보를 지정하여 고유한 사용 사례를 모델링할 수 있습니다.

예를 들어 다음 명령은 20개의 동시 연결(-c 옵션)로 60초(-T 옵션, 시간을 나타냄) 동안 벤치마크를 실행합니다. -C 옵션은 클라이언트 세션당 한 번이 아니라 매번 새로운 연결을 사용하여 테스트를 실행합니다. 이 설정은 연결 오버헤드를 나타냅니다.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

연결을 재사용하지 않고 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에서 pgbench를 실행하면 초당 약 8개의 트랜잭션만 처리됨을 알 수 있습니다. 이는 1분 테스트 동안 총 495건의 트랜잭션을 제공합니다.

연결을 재사용하면 사용자 수에 대한 Aurora PostgreSQL DB 클러스터의 응답이 거의 20배 빨라집니다. 재사용을 통해 동일한 시간 및 동일한 수의 사용자 연결에 대해 495개에 비해 총 9,042개의 트랜잭션이 처리됩니다. 차이점은 다음에서는 각 연결이 재사용된다는 것입니다.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
```

```
        starting vacuum...end.
        transaction type: <builtin: TPC-B (sort of)>
        scaling factor: 50
        query mode: simple
        number of clients: 20
        number of threads: 1
        duration: 60 s
        number of transactions actually processed: 9042
        latency average = 127.880 ms
        initial connection time = 2311.188 ms
        tps = 156.396765 (without initial connection time)
```

이 예에서는 연결 풀링이 응답 시간을 크게 개선할 수 있음을 보여줍니다. Aurora PostgreSQL DB 클러스터용 RDS 프록시 설정에 대한 자세한 내용은 Amazon RDS Proxy for Aurora 단원을 참조하세요.

PostgreSQL에서 연결 중단 처리

클라이언트 애플리케이션이 중단되거나 비정상적으로 종료된 경우에도 서버에서 데이터베이스 세션이 활성 상태로 유지될 때 연결 중단이 발생합니다. 이 상황은 일반적으로 클라이언트가 데이터베이스 연결을 제대로 닫거나 진행 중인 요청을 취소하지 않고 충돌을 처리하거나 예기치 않게 종료할 때 발생합니다.

PostgreSQL은 서버 프로세스가 유휴 상태이거나 클라이언트로 데이터를 전송하려고 할 때 연결 중단을 효율적으로 식별하고 정리합니다. 그러나 유휴 상태이거나 클라이언트 입력을 기다리거나 쿼리를 적극적으로 실행하는 세션의 경우 감지가 어렵습니다. 이러한 시나리오를 처리하기 위해 PostgreSQL은 `tcp_keepalives_*`, `tcp_user_timeout` 및 `client_connection_check_interval` 파라미터를 제공합니다.

주제

- TCP 킵얼라이브 이해
- Aurora PostgreSQL의 주요 TCP 킵얼라이브 파라미터
- TCP 킵얼라이브 설정 사용 사례
- 모범 사례

TCP 킵얼라이브 이해

TCP Keepalive는 연결 무결성을 유지하고 확인하는 데 도움이 되는 프로토콜 수준 메커니즘입니다. 각 TCP 연결은 킵얼라이브 동작을 제어하는 커널 수준 설정을 유지 관리합니다. 킵얼라이브 타이머가 만료되면 시스템은 다음을 수행합니다.

- 데이터 및 ACK 플래그 세트가 없는 프로브 패킷을 전송합니다.
- TCP/IP 사양에 따라 원격 엔드포인트에서 응답을 예상합니다.
- 응답 또는 응답 없음을 기반으로 연결 상태를 관리합니다.

Aurora PostgreSQL의 주요 TCP 킵얼라이브 파라미터

파라미터	설명	기본값
`tcp_keepalives_idle`	Specifies number of seconds of inactivity before sending keepalive message.	300
`tcp_keepalives_interval`	Specifies number of seconds between retransmissions of unacknowledged keepalive messages.	30
`tcp_keepalives_count`	Maximum lost keepalive messages before declaring connection dead	2
`tcp_user_timeout`	Specifies how long (in Milliseconds) unacknowledged data can remain before forcibly closing the connection.	0
`client_connection_check_interval`	Sets the interval (in Milliseconds) for checking client connection status during long-running queries. This ensures quicker detection of closed connections.	0

TCP 킵얼라이브 설정 사용 사례

유휴 세션 연결 유지

활동이 없어 방화벽 또는 라우터에 의해 유휴 연결이 종료되지 않도록 하는 방법:

- 정기적으로 킵얼라이브 패킷을 전송하도록 `tcp_keepalives_idle`을 구성합니다.

연결 중단 감지

연결 중단을 즉시 감지하는 방법:

- `tcp_keepalives_idle`, `tcp_keepalives_interval` 및 `tcp_keepalives_count`를 조정합니다. 예를 들어 Aurora PostgreSQL 기본값을 사용하면 연결 중단을 감지하는 데 약 1분(프로브 2개 × 30초)이 걸립니다. 이러한 값을 낮추면 감지 속도가 빨라질 수 있습니다.
- `tcp_user_timeout`을 사용하여 확인의 최대 대기 시간을 지정합니다.

TCP 킵얼라이브 설정은 커널이 연결 중단을 감지하는 데 도움이 되지만 소켓을 사용할 때까지 PostgreSQL이 작동하지 않을 수 있습니다. 세션이 긴 쿼리를 실행하는 경우 쿼리가 완료된 후에야 연결 중단이 감지될 수 있습니다. PostgreSQL 14 이상 버전에서는 `client_connection_check_interval`이 쿼리 실행 중에 소켓을 주기적으로 폴링하여 연결 중단 감지를 가속화할 수 있습니다.

모범 사례

- 적절한 킵얼라이브 간격 설정: `tcp_user_timeout`, `tcp_keepalives_idle`, `tcp_keepalives_count` 및 `tcp_keepalives_interval`을 조정하여 감지 속도와 리소스 사용의 균형을 맞춥니다.
- 환경에 최적화: 네트워크 동작, 방화벽 정책 및 세션 요구 사항에 맞게 설정을 조정합니다.
- PostgreSQL 기능 활용: 효율적인 연결 확인을 위해 PostgreSQL 14 이상 버전에서 `client_connection_check_interval`을 사용합니다.

Aurora PostgreSQL의 메모리 파라미터 조정

Amazon Aurora PostgreSQL에서는 다양한 처리 작업에 사용되는 메모리 양을 제어하는 여러 파라미터를 사용할 수 있습니다. 작업에서 지정된 파라미터에 설정된 양보다 많은 메모리를 사용하는 경우

Aurora PostgreSQLdms 다른 리소스를 사용하여 처리합니다(예: 디스크에 쓰기). 이로 인해 Aurora PostgreSQL DB 클러스터가 느려지거나 중단될 수 있으며 메모리 부족 오류가 발생할 수 있습니다.

각 메모리 파라미터의 기본 설정은 일반적으로 의도한 처리 작업을 취급할 수 있습니다. 그러나 Aurora PostgreSQL DB 클러스터의 의 메모리 관련 파라미터를 조정할 수도 있습니다. 이렇게 조정하여 특정 워크로드를 처리하기에 충분한 메모리가 할당되도록 합니다.

다음에서 메모리 관리를 제어하는 파라미터에 관한 정보를 확인할 수 있습니다. 메모리 사용률을 평가하는 방법도 알아볼 수 있습니다.

파라미터 값 확인 및 설정

메모리를 관리하고 Aurora PostgreSQL DB 클러스터의 메모리 사용량을 평가하기 위해 설정할 수 있는 파라미터는 다음과 같습니다.

- `work_mem` - Aurora PostgreSQL DB 클러스터가 임시 디스크 파일에 쓰기 전에 내부 정렬 작업 및 해시 테이블에 사용하는 메모리 양을 지정합니다.
- `log_temp_files` - 임시 파일 생성, 파일 이름 및 크기를 기록합니다. 이 파라미터가 켜져 있으면 생성되는 각 임시 파일에 대해 로그 항목이 저장됩니다. 이 기능을 켜면 Aurora PostgreSQL DB 클러스터가 디스크에 써야 하는 빈도를 확인할 수 있습니다. 과도한 로깅을 방지하려면 Aurora PostgreSQL DB 클러스터의 임시 파일 생성에 대한 정보를 수집한 후 다시 끄세요.
- `logical_decoding_work_mem` – 디스크로 유출되기 전에 각 내부 재정렬 버퍼에서 사용할 메모리 양(KB)을 지정합니다. 이 메모리는 복제본을 만드는 프로세스인 논리적 디코딩에 사용됩니다. 이는 미리 쓰기 로그(WAL) 파일의 데이터를 대상에 필요한 논리적 스트리밍 출력으로 변환하여 수행됩니다.

  이 파라미터의 값은 각 복제 연결에 지정된 크기의 단일 버퍼를 생성합니다. 기본값은 65,536KB입니다. 이 버퍼가 채워지면 초과분은 디스크에 파일로 기록됩니다. 디스크 활동을 최소화하기 위해 이 파라미터의 값을 `work_mem`보다 훨씬 높게 설정할 수 있습니다.

이러한 파라미터는 모두 동적 파라미터이므로 현재 세션에 맞게 변경할 수 있습니다. 이렇게 하려면 다음과 같이 psql과 `SET` 문을 사용하여 Aurora PostgreSQL DB 클러스터 에 연결합니다.

```
SET parameter_name TO parameter_value;
```

세션 설정은 세션 기간 동안만 지속됩니다. 세션이 끝나면 파라미터는 DB 클러스터 파라미터 그룹의 설정으로 되돌아갑니다. 파라미터를 변경하려면 먼저 다음과 같이 `pg_settings` 테이블을 쿼리하여 현재 값을 확인해야 합니다.

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

예를 들어 work_mem 파라미터의 값을 찾으려면 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 쿼리를 실행합니다.

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val   | pg_size_pretty
------+----------+-----------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

파라미터 설정이 유지되도록 변경하려면 사용자 지정 DB 클러스터 파라미터 그룹을 사용해야 합니다. SET 문을 사용하여 이러한 파라미터에 대해 다른 값으로 Aurora PostgreSQL DB 클러스터 를 실행한 후 사용자 지정 파라미터 그룹을 생성하고 Aurora PostgreSQL DB 클러스터에 적용할 수 있습니다. 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.

작업 메모리 파라미터 이해

작업 메모리 파라미터(work_mem)는 Aurora PostgreSQL이 복잡한 쿼리를 처리하는 데 사용할 수 있는 최대 메모리 양을 지정합니다. 복잡한 쿼리에는 정렬 또는 그룹화 작업, 즉 다음 절을 사용하는 쿼리가 포함됩니다.

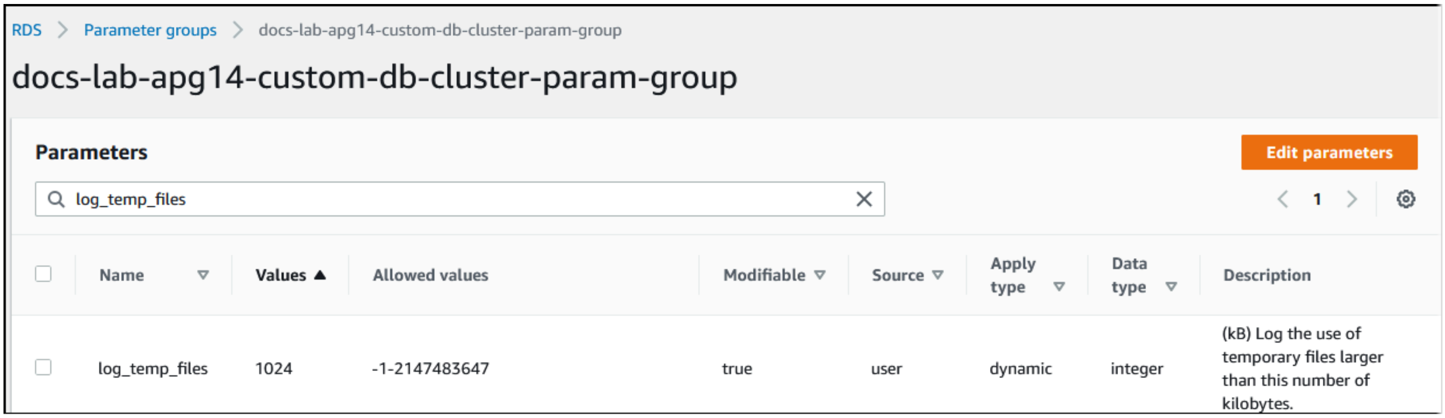
- ORDER BY
- DISTINCT
- GROUP BY
- JOIN(MERGE 및 HASH)

쿼리 플래너는 Aurora PostgreSQL DB 클러스터가 작업 메모리를 사용하는 방식에 간접적으로 영향을 미칩니다. 쿼리 플래너는 SQL 문을 처리하기 위한 실행 계획을 생성합니다. 주어진 계획을 따르면 복잡한 쿼리를 병렬로 실행할 수 있는 여러 작업 단위로 분할할 수 있습니다. 가능한 경우 Aurora PostgreSQL은 각 병렬 프로세스에서 디스크에 쓰기 전에 각 세션에 대해 work_mem 파라미터에 지정된 메모리 양을 사용합니다.

여러 데이터베이스 사용자가 동시에 여러 작업을 실행하고 여러 작업 단위를 병렬로 생성하면 Aurora PostgreSQL DB 클러스터에 할당된 작업 메모리가 소진될 수 있습니다. 이로 인해 임시 파일 생성 및 디스크 I/O가 과도하게 발생하거나 더 심각한 경우 메모리 부족 오류가 발생할 수 있습니다.

임시 파일 사용 확인

쿼리 처리에 필요한 메모리가 `work_mem` 파라미터에 지정된 값을 초과할 때마다 작업 데이터가 임시 파일의 디스크에 오프로드됩니다. `log_temp_files` 파라미터를 켜면 발생하는 빈도를 파악할 수 있습니다. 기본적으로 이 파라미터는 해제(-1로 설정됨)되어 있습니다. 모든 임시 파일 정보를 캡처하려면 이 파라미터를 0으로 설정합니다. `log_temp_files`를 다른 양의 정수로 설정하여 해당 데이터 양(KB) 이상인 파일의 임시 파일 정보를 캡처하세요. 다음 이미지에서는 AWS Management Console의 예를 볼 수 있습니다.



임시 파일 로깅을 구성한 후 자체 워크로드로 테스트하여 작업 메모리 설정이 충분한지 확인할 수 있습니다. PostgreSQL 커뮤니티의 간단한 벤치마킹 애플리케이션인 pgbench를 사용하여 워크로드를 시뮬레이션할 수도 있습니다.

다음 예제는 테스트를 실행하는 데 필요한 테이블과 행을 생성하여 pgbench를 초기화합니다(-i). 이 예에서 배율 인수(-s 50)는 `labdb` 데이터베이스의 `pgbench_branches` 테이블에 50개 행, `pgbench_tellers`에 500개 행, `pgbench_accounts` 테이블에 5,000,000개 행을 생성합니다.

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
```

```
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

환경을 초기화한 후 특정 시간(-T) 동안 특정 클라이언트 수(-c)에 대해 벤치마크를 실행할 수 있습니다. 또한 이 예에서는 -d 옵션을 사용하여 Aurora PostgreSQL DB 클러스터에서 트랜잭션을 처리할 때 디버깅 정보를 출력합니다.

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

pgbench에 대한 자세한 내용은 PostgreSQL 설명서를 참조하세요.

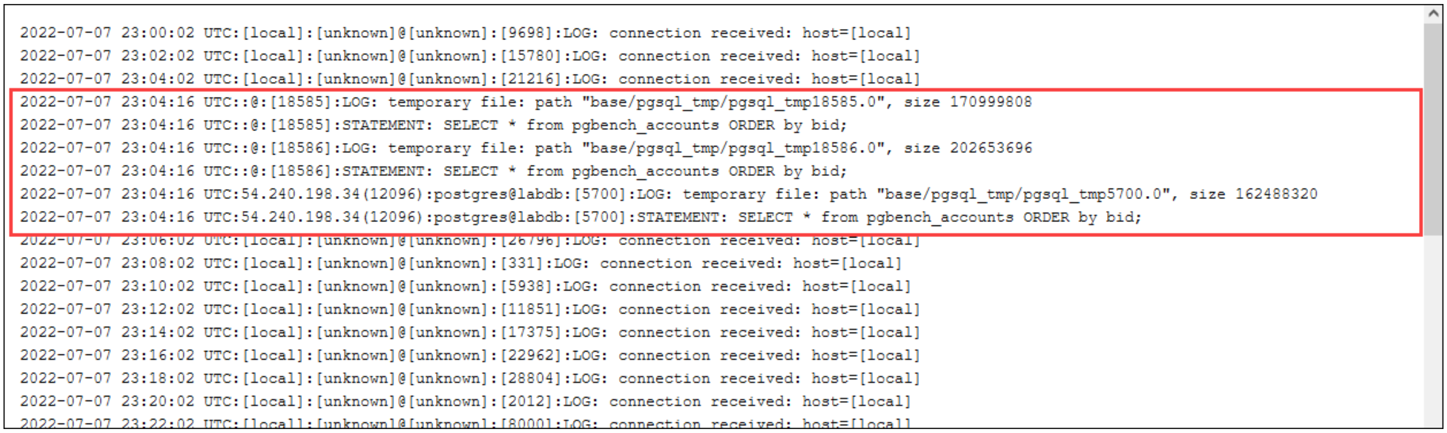
psql metacommand 명령(\d)을 사용하여 pgbench에서 생성한 테이블, 뷰 및 인덱스와 같은 관계를 나열할 수 있습니다.

```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

출력에 표시된 대로 pgbench_accounts 테이블은 aid 열에서 인덱싱됩니다. 이 다음 쿼리가 작업 메모리를 사용하도록 하려면 다음 예에 표시된 것과 같이 인덱싱되지 않은 열을 쿼리합니다.

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

로그에서 임시 파일을 확인합니다. 이렇게 하려면 AWS Management Console을 열고 Aurora PostgreSQL DB 클러스터 인스턴스를 선택한 다음 로그 및 이벤트 탭을 선택합니다. 로그는 콘솔에서 확인하거나 다운로드하여 추가 분석을 수행할 수 있습니다. 다음 이미지에 표시된 것처럼 쿼리를 처리하는 데 필요한 임시 파일의 크기는 `work_mem` 파라미터에 지정된 양을 늘려야 함을 나타냅니다.



```
2022-07-07 23:00:02 UTC:[local]:[unknown]@[unknown]:[9698]:LOG: connection received: host=[local]
2022-07-07 23:02:02 UTC:[local]:[unknown]@[unknown]:[15780]:LOG: connection received: host=[local]
2022-07-07 23:04:02 UTC:[local]:[unknown]@[unknown]:[21216]:LOG: connection received: host=[local]
2022-07-07 23:04:16 UTC::@:[18585]:LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp18585.0", size 170999808
2022-07-07 23:04:16 UTC::@:[18585]:STATEMENT: SELECT * from pgbench_accounts ORDER by bid;
2022-07-07 23:04:16 UTC::@:[18586]:LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp18586.0", size 202653696
2022-07-07 23:04:16 UTC::@:[18586]:STATEMENT: SELECT * from pgbench_accounts ORDER by bid;
2022-07-07 23:04:16 UTC:54.240.198.34(12096):postgres@labdb:[5700]:LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp5700.0", size 162488320
2022-07-07 23:04:16 UTC:54.240.198.34(12096):postgres@labdb:[5700]:STATEMENT: SELECT * from pgbench_accounts ORDER by bid;
2022-07-07 23:06:02 UTC:[local]:[unknown]@[unknown]:[26796]:LOG: connection received: host=[local]
2022-07-07 23:08:02 UTC:[local]:[unknown]@[unknown]:[331]:LOG: connection received: host=[local]
2022-07-07 23:10:02 UTC:[local]:[unknown]@[unknown]:[5938]:LOG: connection received: host=[local]
2022-07-07 23:12:02 UTC:[local]:[unknown]@[unknown]:[11851]:LOG: connection received: host=[local]
2022-07-07 23:14:02 UTC:[local]:[unknown]@[unknown]:[17375]:LOG: connection received: host=[local]
2022-07-07 23:16:02 UTC:[local]:[unknown]@[unknown]:[22962]:LOG: connection received: host=[local]
2022-07-07 23:18:02 UTC:[local]:[unknown]@[unknown]:[28804]:LOG: connection received: host=[local]
2022-07-07 23:20:02 UTC:[local]:[unknown]@[unknown]:[2012]:LOG: connection received: host=[local]
2022-07-07 23:22:02 UTC:[local]:[unknown]@[unknown]:[8000]:LOG: connection received: host=[local]
```

운영상의 필요에 따라 개인 및 그룹에 대해 이 파라미터를 다르게 구성할 수 있습니다. 예를 들어 `dev_team`이라는 역할에 대해 `work_mem` 파라미터를 8GB로 설정할 수 있습니다.

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

`work_mem`에 대한 이 설정을 사용하면 `dev_team` 역할의 구성원인 역할에 최대 8GB의 작업 메모리가 할당됩니다.

응답 시간 단축을 위한 인덱스 사용

쿼리가 결과를 반환하는 데 너무 오래 걸리는 경우 인덱스가 예상대로 사용되고 있는지 확인할 수 있습니다. 먼저 다음과 같이 psql 메타 명령인 `\timing`을 켭니다.

```
postgres=>  \timing on
```

타이밍을 켠 후 간단한 SELECT 문을 사용합니다.

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
```

```
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

출력에 표시된 대로 이 쿼리를 완료하는 데 3초가 조금 넘게 걸렸습니다. 응답 시간을 개선하려면 다음과 같이 pgbench_accounts에 인덱스를 생성합니다.

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

쿼리를 다시 실행하면 응답 시간이 더 빨라지는 것을 확인할 수 있습니다. 이 예시에서는 쿼리가 약 0.5초 만에 5배 더 빠르게 완료되었습니다.

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

논리적 디코딩을 위한 작업 메모리 조정

논리적 복제는 PostgreSQL 버전 10에 도입된 이후 모든 버전의 Aurora PostgreSQL 에서 사용할 수 있습니다. 논리적 복제를 구성할 때 논리적 디코딩 프로세스가 디코딩 및 스트리밍 프로세스에 사용할 수 있는 메모리 양을 지정하도록 logical_decoding_work_mem 파라미터를 설정할 수도 있습니다.

논리적 디코딩 중에 미리 쓰기 로그(WAL) 레코드는 SQL 문으로 변환된 다음 논리적 복제나 다른 작업을 위해 다른 대상으로 전송됩니다. 트랜잭션이 WAL에 작성된 다음 변환되면 전체 트랜잭션이 logical_decoding_work_mem에 지정된 값에 맞아야 합니다. 기본적으로 이 파라미터는 65,536MB로 설정됩니다. 모든 오버플로우는 디스크에 기록됩니다. 따라서 디스크에서 다시 읽어야 대상으로 보낼 수 있기 때문에 전체 프로세스 속도가 느려집니다.

다음 예와 같이 aurora_stat_file 함수를 사용하여 특정 시점에서 현재 워크로드의 트랜잭션 오버플로 양을 평가할 수 있습니다.

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
```

```
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

이 쿼리는 쿼리가 호출될 때 Aurora PostgreSQL DB 클러스터에 있는 유출 파일의 수와 크기를 반환합니다. 더 오래 실행되는 워크로드는 아직 디스크에 유출 파일이 없을 수 있습니다. 장기 실행 워크로드를 프로파일링하려면 워크로드가 실행될 때 유출 파일 정보를 캡처하는 테이블을 생성하는 것이 좋습니다. 다음과 같이 테이블을 만들 수 있습니다.

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

논리적 복제 중에 스필 파일이 사용되는 방식을 보려면 게시자와 구독자를 설정한 다음 단순 복제를 시작합니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터의 논리적 복제 설정 단원을 참조하십시오. 복제가 진행되면 다음과 같이 `aurora_stat_file()` 유출 파일 함수에서 결과 집합을 캡처하는 작업을 만들 수 있습니다.

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

다음 psql 명령을 사용하여 초당 한 번씩 작업을 실행합니다.

```
\watch 0.5
```

작업이 실행 중일 때 다른 psql 세션에서 라이터 인스턴스에 연결합니다. 다음 일련의 명령문을 사용하여 메모리 구성을 초과하는 Aurora PostgreSQL이 유출 파일을 생성하도록 하는 워크로드를 실행하세요.

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
```

```
UPDATE 10000001
```

이러한 명령문은 완료하는 데 몇 분 정도 걸립니다. 완료되면 Ctrl 키와 C 키를 함께 눌러 모니터링 기능을 중지합니다. 그런 다음 아래 명령을 사용하여 Aurora PostgreSQL DB 클러스터의 유출 파일 사용에 대한 정보를 보관하는 테이블을 생성합니다.

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                   spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
------------------------------+-----------------------+--------+------------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

출력은 예를 실행하면 611MB의 메모리를 사용하는 유출 파일 5개가 생성되었음을 보여줍니다. 디스크에 쓰지 않으려면 `logical_decoding_work_mem` 파라미터는 다음으로 높은 메모리 크기인 1024로 설정하는 것이 좋습니다.

Amazon CloudWatch 지표를 사용한 Aurora PostgreSQL의 리소스 사용량 분석

Aurora는 1분 단위로 CloudWatch에 지표 데이터를 자동 전송합니다. CloudWatch 지표를 사용하여 Aurora PostgreSQL의 리소스 사용량을 분석할 수 있습니다. 지표를 사용하여 네트워크 처리량(throughput)과 네트워크 사용량을 평가할 수 있습니다.

CloudWatch를 사용한 네트워크 처리량(throughput) 평가

시스템 사용량이 인스턴스 유형의 리소스 한도에 가까워지면 처리 속도가 느려질 수 있습니다. CloudWatch Logs Insights를 사용하여 스토리지 리소스 사용량을 모니터링하고 사용할 수 있는 리소스가 충분한지 확인할 수 있습니다. 필요한 경우 DB 인스턴스를 더 큰 인스턴스 클래스로 수정할 수 있습니다.

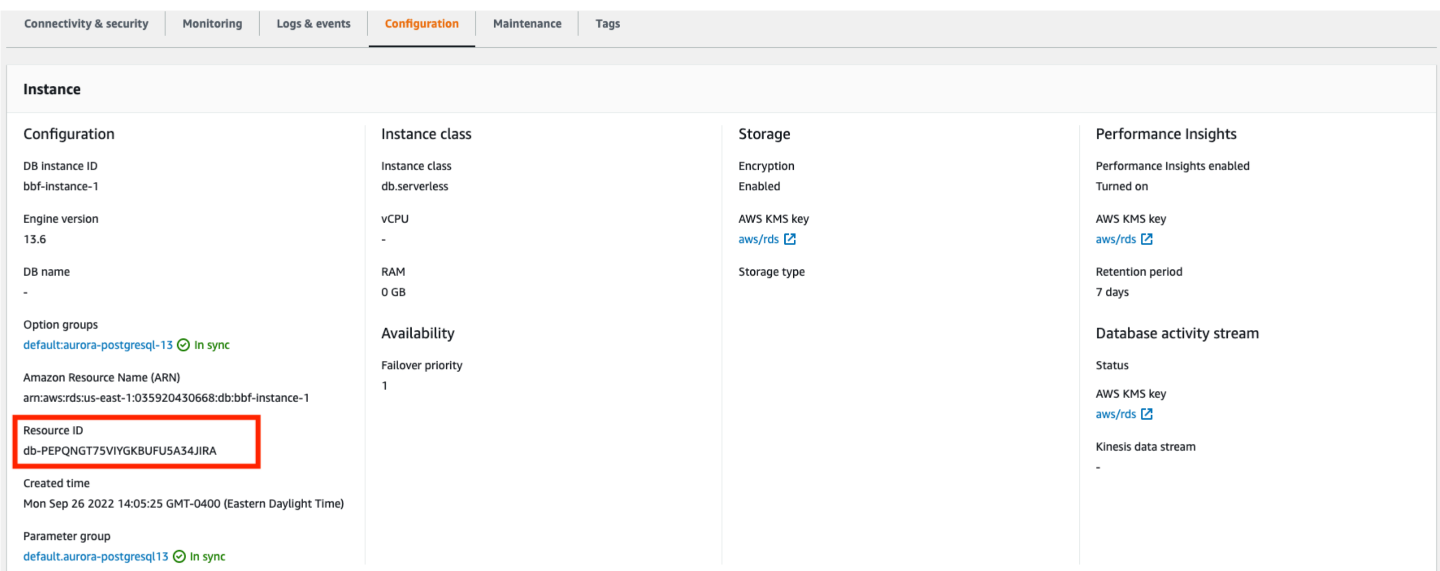
다음과 같은 이유로 Aurora 스토리지 처리 속도가 느릴 수 있습니다.

- 클라이언트와 DB 인스턴스 간의 네트워크 대역폭이 부족합니다.
- 스토리지 하위 시스템에 대한 네트워크 대역폭이 부족합니다.
- 인스턴스 유형에 비해 규모가 큰 워크로드입니다.

CloudWatch Logs Insights를 쿼리하여 Aurora 스토리지 리소스 사용량 그래프를 생성하여 리소스를 모니터링할 수 있습니다. 그래프는 더 큰 인스턴스 크기로 스케일 업할지를 결정하는 데 도움이 되는 CPU 사용률 및 지표를 보여줍니다. CloudWatch Logs Insights의 쿼리 구문에 대한 자세한 내용은 [CloudWatch Logs Insights query syntax](CloudWatch Logs Insights 쿼리 구문)를 참조하세요.

CloudWatch를 사용하려면 Aurora PostgreSQL 로그 파일을 CloudWatch로 내보내야 합니다. CloudWatch로 로그를 내보내도록 기존 클러스터를 수정할 수도 있습니다. CloudWatch로 로그를 내보내는 방법에 대한 자세한 내용은 Amazon CloudWatch로 로그를 게시하는 옵션 설정을 참조하세요.

CloudWatch Logs Insights를 쿼리하려면 DB 인스턴스의 Resource ID(리소스 ID)가 필요합니다. Resource ID(리소스 ID)는 콘솔의 Configuration(구성) 탭에서 확인할 수 있습니다.



로그 파일에서 리소스 스토리지 지표를 쿼리하려면 다음을 수행하세요.

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

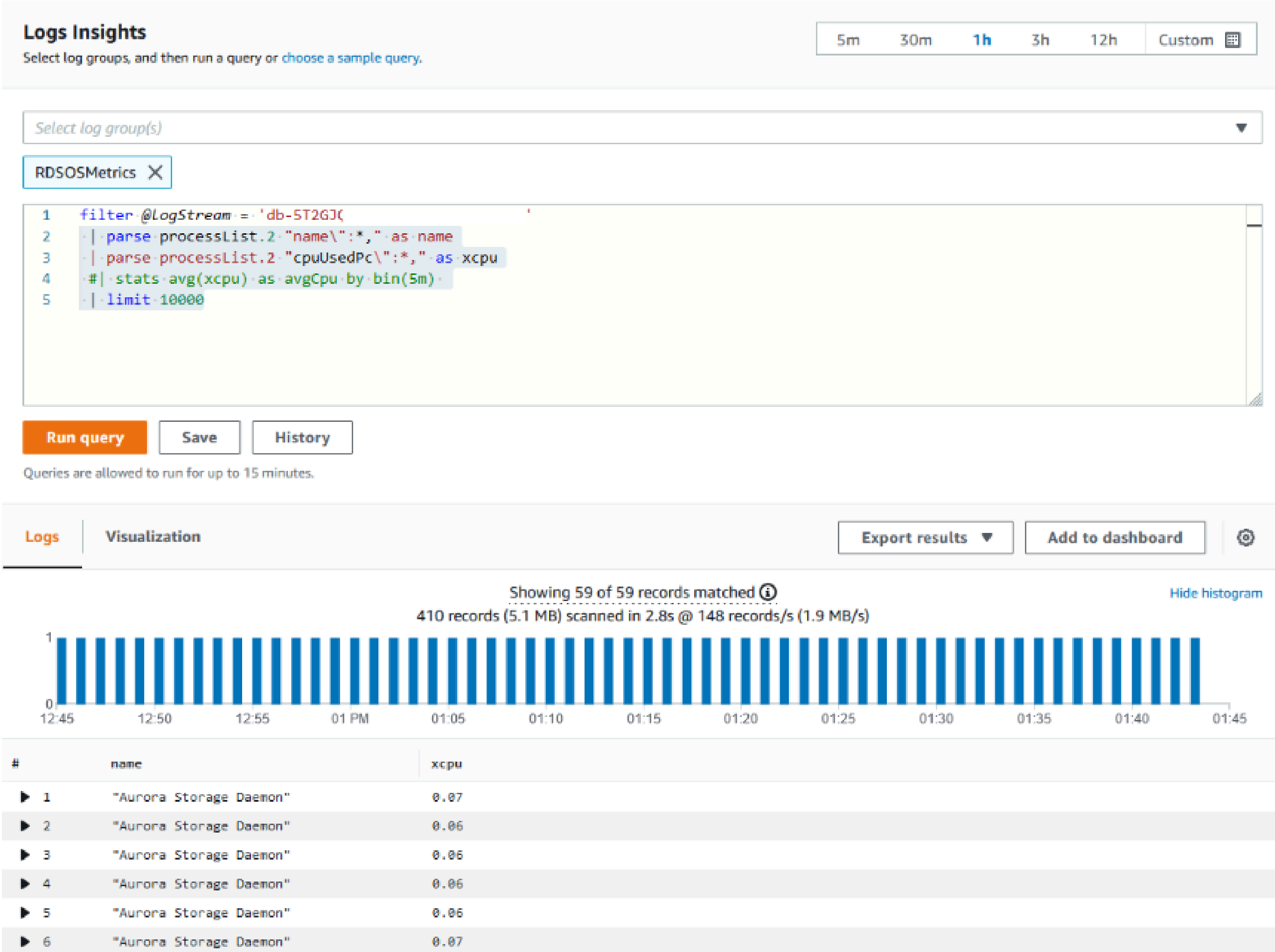
   CloudWatch 개요 홈페이지가 표시됩니다.
2. 필요한 경우 AWS 리전을 변경합니다. 탐색 모음에서 AWS가 있는 AWS 리전을 선택합니다. 자세한 내용은 리전 및 엔드포인트 섹션을 참조하세요.
3. 탐색 창에서 Logs(로그)를 선택한 다음, Logs Insights를 선택합니다.

   Logs Insights 페이지가 나타납니다.
4. 드롭다운 목록에서 분석할 로그 파일을 선택합니다.
5. 필드에 다음 쿼리를 입력하고 `<resource ID>`를 DB 클러스터의 리소스 ID로 바꿉니다.

   ```
   filter @logStream = <resource ID> | parse @message "\"Aurora Storage
   Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc
   | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by
   bin(5m) | limit 10000
   ```
6. Run query(쿼리 실행)를 클릭합니다.

   스토리지 사용률 그래프가 표시됩니다.

   다음 이미지는 Logs Insights 페이지와 그래프를 나타냅니다.



CloudWatch 지표를 사용한 Aurora PostgreSQL에 대한 DB 인스턴스 사용량 평가

CloudWatch 지표를 사용하여 인스턴스 처리량(throughput)을 관찰하고 인스턴스 클래스가 애플리케이션에 제공하는 리소스가 충분한지 확인할 수 있습니다. DB 인스턴스 클래스 한도에 대한 자세한 내용은 [Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양]에서 DB 인스턴스 클래스의 사양을 찾아 네트워크 성능을 확인하세요.

DB 인스턴스 사용량이 인스턴스 클래스 한도에 가까워지면 성능이 느려질 수 있습니다. CloudWatch 지표를 통해 이러한 상황을 확인할 수 있으므로 더 큰 인스턴스 클래스로 수동 스케일 업할 계획을 세울 수 있습니다.

다음 CloudWatch 지표 값을 조합하여 인스턴스 클래스 한도에 가까워졌는지 확인하세요.

- NetworkThroughput – Aurora DB 클러스터의 각 인스턴스에 대해 클라이언트가 수신 및 전송한 네트워크 처리량(throughput)입니다. 이 처리량(throughput) 값에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkThroughput – Aurora DB 클러스터의 각 인스턴스에서 Aurora 스토리지 하위 시스템으로 수신 및 전송한 네트워크 처리량(throughput)입니다.

NetworkThroughput을 StorageNetworkThroughput에 추가하여 Aurora DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신 및 전송한 네트워크 처리량(throughput)을 확인하세요. 인스턴스에 대한 인스턴스 클래스 한도는 이 두 지표를 합한 값보다 커야 합니다.

다음 지표를 사용하여 전송 및 수신 시 클라이언트 애플리케이션의 네트워크 트래픽에 대한 추가 세부 정보를 검토할 수 있습니다.

- NetworkReceiveThroughput – Aurora PostgreSQL DB 클러스터의 각 인스턴스가 클라이언트로부터 수신한 네트워크 처리량(throughput)입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- NetworkTransmitThroughput – Aurora DB 클러스터의 각 인스턴스가 클라이언트로 전송한 네트워크 처리량(throughput)입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkReceiveThroughput – DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신한 네트워크 처리량(throughput)입니다.
- StorageNetworkTransmitThroughput – DB 클러스터의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송한 네트워크 처리량(throughput)입니다.

이러한 지표를 모두 더해 네트워크 사용량을 인스턴스 클래스 한도와 비교하는 방법을 평가합니다. 인스턴스 클래스 한도는 이러한 결합 지표를 합한 값보다 커야 합니다.

스토리지의 네트워크 한도와 CPU 사용률은 상호적인 관계가 있습니다. 네트워크 처리량(throughput)이 증가하면 CPU 사용률도 증가합니다. CPU 및 네트워크 사용량을 모니터링하면 리소스가 소진되는 방식 및 이유에 대한 정보를 얻을 수 있습니다.

네트워크 사용을 최소화하려면 다음을 고려하면 됩니다.

- 규모가 더 큰 인스턴스 클래스를 사용합니다.
- `pg_partman` 파티셔닝 전략을 사용합니다.
- 쓰기 요청을 일괄적으로 나누어 전체 트랜잭션을 줄입니다.

- 읽기 전용 워크로드를 읽기 전용 인스턴스로 지정합니다.
- 사용하지 않는 인덱스를 삭제합니다.
- 부풀려진 객체 및 VACUUM이 있는지 확인합니다. 팽창이 심한 경우에는 PostgreSQL 확장 `pg_repack`을 사용합니다. `pg_repack`에 대한 자세한 내용은 Reorganize tables in PostgreSQL databases with minimal locks(최소 잠금으로 PostgreSQL 데이터베이스의 테이블 재구성)를 참조하세요.

논리적 복제를 사용하여 Aurora PostgreSQL에 대한 메이저 버전 업그레이드 수행

논리적 복제와 Aurora 고속 복제를 사용하면 변경되는 데이터를 새로운 메이저 버전 데이터베이스로 점진적으로 마이그레이션하면서도 현재 Aurora PostgreSQL 데이터베이스 버전을 사용하는 메이저 버전 업그레이드를 수행할 수 있습니다. 가동 중지 시간이 짧은 이 업그레이드 프로세스를 블루/그린 업그레이드라고 합니다. 데이터베이스의 현재 버전을 "블루" 환경이라고 하고 새 데이터베이스 버전을 "그린" 환경이라고 합니다.

Aurora 고속 클로닝은 소스 데이터베이스의 스냅샷을 생성하여 기존 데이터를 완전히 로드합니다. 고속 클로닝은 Aurora 스토리지 계층에 구축되는 기록 중 복사(copy-on-write) 프로토콜을 사용하므로, 짧은 시간 안에 데이터베이스 클론을 생성할 수 있습니다. 이 방법은 대규모 데이터베이스로 업그레이드할 때 매우 효과적입니다.

PostgreSQL에서의 논리적 복제는 사용자가 PostgreSQL 신규 버전으로 전환할 때까지 초기 인스턴스의 데이터 변경 사항을 추적하고 병렬로 실행되는 새 인스턴스로 전송합니다. 논리적 복제에서는 게시 및 구독 모델을 사용합니다. Aurora PostgreSQL 논리적 복제에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 사용한 복제 섹션을 참조하세요.

> **Tip**
>
> 관리형 Amazon RDS 블루/그린 배포 기능을 사용하여 메이저 버전 업그레이드에 필요한 가동 중지를 최소화할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 단원을 참조하십시오.

주제

- 요구 사항
- 제한 사항

- 파라미터 값 설정 및 확인
- Aurora PostgreSQL를 새로운 메이저 버전으로 업그레이드
- 업그레이드 후 작업 수행

요구 사항

가동 중지 시간이 짧은 이 업그레이드 프로세스를 수행하려면 다음 요구 사항을 충족해야 합니다.

- rds_superuser 권한이 있어야 합니다.
- 업그레이드하려는 Aurora PostgreSQL DB 클러스터가 논리적 복제를 사용하여 메이저 버전 업그레이드를 수행할 수 있는 지원 버전을 실행해야 합니다. DB 클러스터에 마이너 버전 업데이트 및 패치가 있다면 적용해야 합니다. 이 기법에 사용되는 `aurora_volume_logical_start_lsn` 함수는 다음 버전의 Aurora PostgreSQL에서 지원됩니다.
  - 15.2 이상의 15 버전
  - 14.3 이상의 14 버전
  - 13.6 이상의 13 버전
  - 12.10 이상의 12 버전
  - 11.15 이상의 11 버전
  - 10.20 이상의 10 버전

  `aurora_volume_logical_start_lsn` 함수에 대한 자세한 내용은 aurora_volume_logical_start_lsn 섹션을 참조하세요.
- 모든 테이블에는 프라이머리 키가 있거나 PostgreSQL ID 열이 포함되어야 합니다.
- 두 (이전 및 신규 클러스터를 포함한) Aurora PostgreSQL DB 클러스터 간의 인바운드 및 아웃바운드 액세스를 허용하도록 VPC의 보안 그룹을 구성합니다. 특정 Classless Inter-Domain Routing(CIDR) 범위 또는 VPC나 피어 VPC의 다른 보안 그룹에 대한 액세스 권한을 부여할 수 있습니다. (피어 VPC에는 VPC 피어링 연결이 필요합니다.)

> **Note**
>
> 실행 중인 논리적 복제 시나리오를 구성하고 관리하는 데 필요한 권한에 대한 자세한 내용은 PostgreSQL 핵심 설명서를 참조하십시오.

제한 사항

Aurora PostgreSQL DB 클러스터를 새 메이저 버전으로 업그레이드하기 위해 가동 중지 시간이 짧은 업그레이드를 수행할 때는 기본 PostgreSQL 논리적 복제 기능을 사용하게 됩니다. PostgreSQL 논리적 복제와 동일한 기능과 제한 사항이 있습니다. 자세한 내용은 [PostgreSQL 논리적 복제](#)를 참조하세요.

- 데이터 정의 언어(DDL) 명령은 복제되지 않습니다.
- 복제는 라이브 데이터베이스에서의 스키마 변경을 지원하지 않습니다. 스키마는 클로닝 프로세스 중에 원래 형태로 재생성됩니다. 클로닝 후 업그레이드가 완료되기 전에 스키마를 변경하면 변경 사항이 업그레이드된 인스턴스에 반영되지 않습니다.
- 대형 객체는 복제되지 않지만 일반 테이블에 데이터를 저장할 수 있습니다.
- 복제는 파티션을 나눈 테이블을 포함한 테이블에서만 지원됩니다. 뷰, 구체화된 뷰 또는 외부 테이블 같은 다른 유형의 관계로의 복제는 지원되지 않습니다.
- 시퀀스 데이터는 복제되지 않으며 장애 조치 후 수동 업데이트가 필요합니다.

> **Note**
>
> 이 업그레이드는 자동 스크립팅을 지원하지 않습니다. 모든 단계를 직접 수행해야 합니다.

파라미터 값 설정 및 확인

업그레이드하기 전에 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 게시 서버로 작동하도록 구성합니다. 인스턴스는 다음 설정이 적용된 사용자 지정 DB 클러스터 파라미터 그룹을 사용해야 합니다.

- `rds.logical_replication` – 이 파라미터를 1로 설정합니다. `rds.logical_replication` 파라미터는 미리 쓰기 로그 파일 관리를 제어하는 독립 실행형 PostgreSQL 서버의 `wal_level` 파라미터 및 기타 파라미터와 동일한 용도로 사용됩니다.
- `max_replication_slots` – 이 파라미터를 생성할 총 구독 수로 설정합니다. AWS DMS를 사용 중인 경우 이 파라미터를 이 DB 클러스터에서 변경된 데이터를 캡처하는 데 사용할 AWS DMS 태스크의 숫자로 설정합니다.
- `max_wal_senders` - 관리 태스크와 새 세션에 사용할 수 있도록 동시 연결 수에 약간의 여분을 더한 수로 설정합니다. AWS DMS를 사용하는 경우 max_wal_senders 수는 총 동시 세션 수에 특정 시간에 작동하는 AWS DMS 태스크 수를 더한 값과 같아야 합니다.

- max_logical_replication_workers - 예상하는 논리적 복제 작업자 및 테이블 동기화 작업자 수로 설정합니다. 일반적으로는 복제 작업자 수를 max_wal_senders에 사용한 것과 동일한 값으로 설정하는 것이 안전합니다. 작업자는 서버에 할당된 백그라운드 프로세스 풀(max_worker_processes)에서 가져옵니다.
- max_worker_processes - 서버의 백그라운드 프로세스 수로 설정합니다. 이 수는 복제, 자동 진공 처리 프로세스 및 동시에 수행될 수 있는 기타 유지 관리 프로세스에 작업자를 할당할 수 있을만큼 커야 합니다.

신규 버전의 Aurora PostgreSQL로 업그레이드하는 경우 이전 버전 파라미터 그룹에서 수정한 모든 파라미터를 복제해야 합니다. 이러한 파라미터는 업그레이드된 버전에 적용됩니다. pg_settings 테이블을 쿼리하여 파라미터 설정 목록을 얻어 새 Aurora PostgreSQL DB 클러스터에서 파라미터 설정을 다시 생성할 수 있습니다.

예를 들어 복제 파라미터의 설정을 얻으려면 다음 쿼리를 실행합니다.

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

Aurora PostgreSQL를 새로운 메이저 버전으로 업그레이드

게시자(블루) 준비 방법

1. 다음 예제에서 소스 라이터 인스턴스(블루)는 PostgreSQL 버전 11.15를 실행하는 Aurora PostgreSQL DB 클러스터입니다. 이 복제 시나리오에서는 게시 노드입니다. 이 데모에서 소스 라이터 인스턴스는 일련의 값을 포함하는 샘플 테이블을 호스팅합니다.

   ```
   CREATE TABLE my_table (a int PRIMARY KEY);
   INSERT INTO my_table VALUES (generate_series(1,100));
   ```

2. 원본 인스턴스에서 게시자를 만들려면 psql(CLI for PostgreSQL) 또는 원하는 클라이언트를 사용하여 인스턴스의 라이터 노드에 연결해야 합니다. 각 데이터베이스에 다음 명령을 입력합니다.

   ```
   CREATE PUBLICATION publication_name FOR ALL TABLES;
   ```

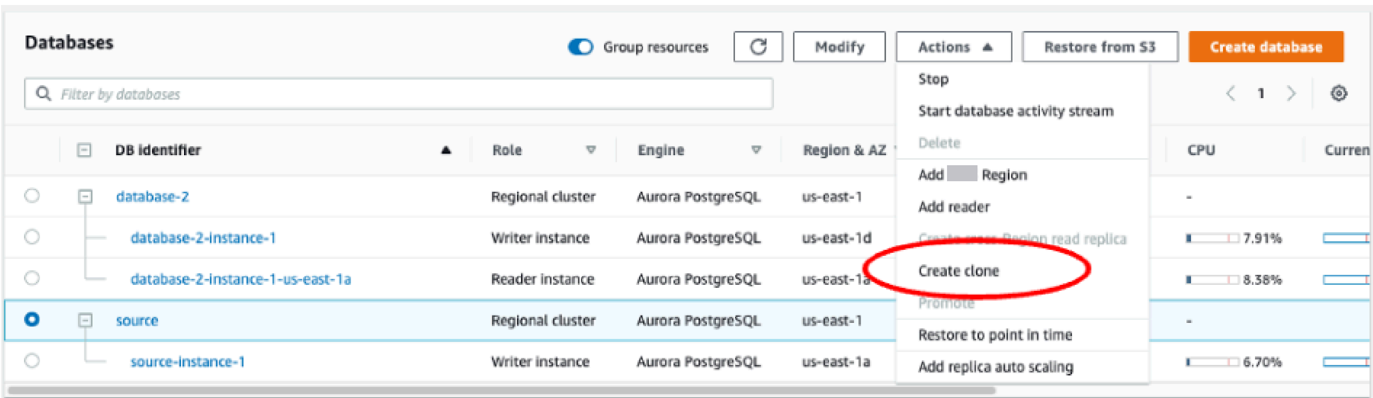
   publication_name은 게시의 이름을 지정합니다.

3. 인스턴스에서 복제 슬롯을 생성해야 합니다. 다음 명령은 복제 슬롯을 생성하고 pgoutput [논리적 디코딩 플러그인]을 로드합니다. 이 플러그인은 미리 쓰기 로그(WAL)에서 읽은 콘텐츠를 논리적 복제 프로토콜로 변경하고, 게시 사양에 따라 데이터를 필터링합니다.

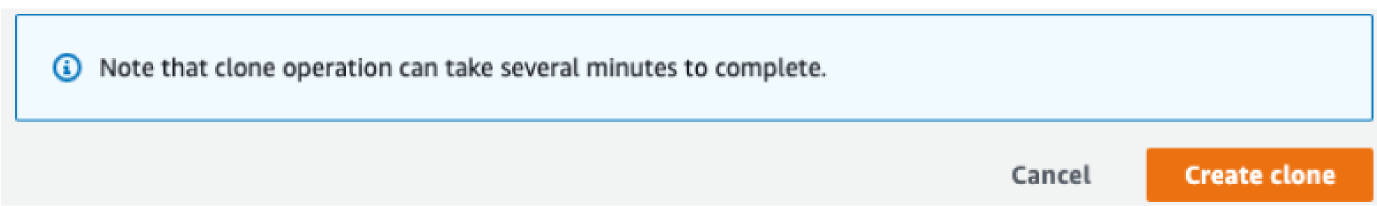
```
SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
```

게시자를 복제하는 방법

1. Amazon RDS 콘솔을 사용하여 원본 인스턴스의 클론을 만듭니다. Amazon RDS 콘솔에서 인스턴스 이름을 강조 표시한 다음 Actions(작업) 메뉴에서 Create clone(복제본 생성)을 선택합니다.



2. 인스턴스의 고유 이름을 입력합니다. 대부분의 설정은 소스 인스턴스의 기본값입니다. 새 인스턴스에 필요한 변경 사항을 적용했다면 Create clone(복제본 생성)을 선택합니다.



3. 대상 인스턴스가 시작되는 동안 라이터 노드의 Status(상태) 열에는 Status(상태) 열에 생성 중(Creating)이 표시됩니다. 인스턴스가 준비되면 인스턴스의 상태가 사용 가능(Aavailable)으로 변경됩니다.

업그레이드를 위해 클론 준비

1. 클론은 배포 모델의 '그린' 인스턴스입니다. 이것은 복제 구독 노드의 호스트입니다. 노드를 사용할 수 있게 되면 psql에 연결하고 새 라이터 노드에 아래의 쿼리를 수행해 LSN(로그 시퀀스 번호)을 얻습니다. LSN은 WAL 스트림에서 레코드가 시작되는 부분을 나타냅니다.

```
SELECT aurora_volume_logical_start_lsn();
```

2. 쿼리의 응답에서 LSN 번호를 찾을 수 있습니다. 프로세스 후반부에 필요하니 이 번호를 메모해 두십시오.

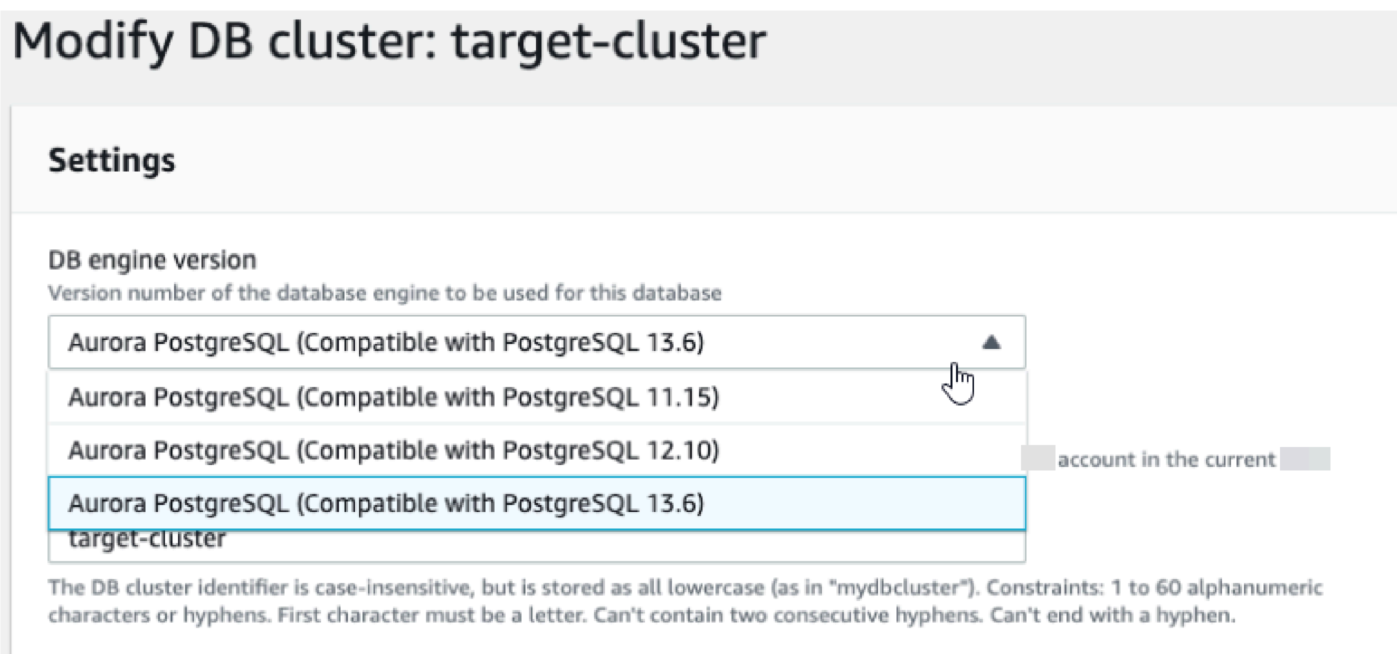
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. 클론을 업그레이드하기 전에 클론의 복제 슬롯을 삭제하세요.

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

클러스터를 새 메이저 버전으로 업그레이드

- 공급자 노드를 복제한 후 Amazon RDS 콘솔을 사용하여 구독 노드에서 메이저 버전 업그레이드를 시작합니다. RDS 콘솔에서 인스턴스 이름을 강조 표시하고 Modify(수정) 버튼을 선택합니다. 업데이트된 버전과 업데이트된 파라미터 그룹을 선택하고, 설정을 즉시 적용하여 대상 인스턴스를 업그레이드합니다.



- CLI를 사용하여 업그레이드를 수행할 수도 있습니다.

```
aws rds modify-db-cluster —db-cluster-identifier $TARGET_Aurora_ID —engine-version
 13.6 —allow-major-version-upgrade —apply-immediately
```

가입자(그린)를 준비하는 방법

1. 업그레이드 후에 클론이 사용 가능해지면 psql에 연결하고 구독을 정의합니다. 이렇게 하려면 `CREATE SUBSCRIPTION` 명령에서 다음 옵션을 지정해야 합니다.
   - `subscription_name` – 구독의 이름입니다.
   - `admin_user_name` - rds_superuser 권한이 있는 관리자 사용자의 이름입니다.
   - `admin_user_password` - 관리자 사용자와 연결된 암호입니다.
   - `source_instance_URL` - 게시 서버 인스턴스의 URL입니다.
   - `database` - 구독 서버가 연결할 데이터베이스입니다.
   - `publication_name` - 게시 서버의 이름입니다.
   - `replication_slot_name` - 복제 슬롯의 이름입니다.

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. 구독을 생성한 후 [pg_replication_origin](https://www.postgresql.org/docs/current/catalog-pg-replication-origin.html) 뷰를 쿼리하여 복제 원본의 식별자인 roname 값을 검색합니다. 각 인스턴스에는 `roname`이 하나씩 있습니다.

```
SELECT * FROM pg_replication_origin;
```

예:

```
postgres=>
SELECT * FROM pg_replication_origin;

roident | roname
---------+----------
1 | pg_24586
```

3. 게시 노드의 이전 쿼리에서 저장한 LSN과 명령의 [INSTANCE] 구독 노드에서 반환된 roname을 입력합니다. 이 명령은 pg_replication_origin_advance 함수를 사용하여 복제를 위한 로그 시퀀스의 시작점을 지정합니다.

```
SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
```

roname은 pg_replication_origin 뷰에서 반환된 식별자입니다.

log_sequence_number는 aurora_volume_logical_start_lsn 함수의 이전 쿼리에서 반환된 값입니다.

4. 그런 다음 ALTER SUBSCRIPTION... ENABLE 절을 사용하여 논리적 복제를 켭니다.

```
ALTER SUBSCRIPTION subscription_name ENABLE;
```

5. 이 시점에서 복제가 제대로 작동하는지 확인할 수 있습니다. 게시 인스턴스에 값을 추가한 다음 값이 구독 노드에 복제되었는지 확인합니다.

그런 다음 아래 명령을 사용하여 게시 노드의 복제 지연을 모니터링합니다.

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

예:

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                     | slot_name              | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

`diff_size` 및 `diff_bytes` 값을 사용하여 복제 지연을 모니터링할 수 있습니다. 이러한 값이 0에 도달하면 복제본이 원본 DB 인스턴스를 따라잡은 것입니다.

업그레이드 후 작업 수행

업그레이드가 완료되면 콘솔 대시보드 Status(상태) 열에 인스턴스 상태가 Available(사용 가능)로 표시됩니다. 새 인스턴스에서는 다음을 수행하는 것이 좋습니다.

- 라이터 노드를 가리키도록 애플리케이션을 리디렉션합니다.
- 케이스로드를 관리하고 라이터 노드에 문제가 발생할 경우 고가용성을 제공할 수 있도록 리더 노드를 추가합니다.
- Aurora PostgreSQL DB 클러스터는 때때로 운영 체제 업데이트가 필요합니다. 이러한 업데이트에는 glibc 라이브러리의 최신 버전도 포함될 수 있습니다. 이러한 업데이트를 진행하는 중에는 [Aurora PostgreSQL에서 지원되는 데이터 정렬]에 설명된 지침을 따르는 것이 좋습니다.
- 액세스를 보장하기 위해 새 인스턴스의 사용자 권한을 업데이트합니다.

새 인스턴스에서 애플리케이션과 데이터를 테스트한 후에는 초기 인스턴스를 제거하기 전에 최종 백업을 만드는 것이 좋습니다. Aurora 호스트에서 논리적 복제를 사용하는 자세한 방법은 [Aurora PostgreSQL DB 클러스터의 논리적 복제 설정] 단원을 참조하십시오.

Aurora PostgreSQL의 스토리지 문제 해결

정렬이나 인덱스 생성 작업에 필요한 작업 메모리 양이 `work_mem` 파라미터에 의해 할당된 메모리양을 초과할 경우, Aurora PostgreSQL는 잉여 데이터를 임시 디스크 파일에 기록합니다. Aurora PostgreSQL은 데이터를 기록할 때 오류와 메시지 로그 저장에 사용하는 것과 동일한 스토리지 공간, 즉 로컬 스토리지를 사용합니다. Aurora PostgreSQL DB 클러스터에 있는 각 인스턴스는 일정한 양의 로컬 스토리지를 사용할 수 있습니다. 스토리지의 양은 스토리지의 DB 인스턴스 클래스에 따라 달라집니다. 로컬 스토리지의 양을 늘리려면 더 큰 DB 인스턴스 클래스를 사용하도록 인스턴스를 수정해야 합니다. DB 인스턴스 클래스 사양은 [Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양] 단원을 참조하십시오.

`FreeLocalStorage`용 Amazon CloudWatch 지표를 관찰하면 Aurora PostgreSQL DB 클러스터의 로컬 스토리지 공간을 모니터링할 수 있습니다. 이 지표는 Aurora DB의 각 DB 인스턴스에서 임시 테이블 및 로그로 사용할 수 있는 스토리지 크기를 보고합니다. 자세한 내용은 [Amazon CloudWatch로 Amazon Aurora 지표 모니터링] 섹션을 참조하세요.

정렬, 인덱싱 및 그룹화 작업은 작업 메모리에서 시작되지만 로컬 스토리지에 오프로드해야 하는 경우가 많습니다. 이러한 유형의 작업 때문에 Aurora PostgreSQL DB 클러스터의 로컬 스토리지가 부족한 경우, 다음 작업 중 하나를 수행하여 문제를 해결할 수 있습니다.

- 작업 메모리 양을 늘립니다. 이렇게 하면 로컬 스토리지를 사용할 필요가 줄어듭니다. 기본적으로 PostgreSQL은 각 정렬, 그룹 및 인덱스 작업에 4MB를 할당합니다. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 현재 작업 메모리 값을 확인하려면, `psql`을 사용하여 인스턴스에 연결하고 다음 명령을 실행합니다.

  ```
  postgres=> SHOW work_mem;
  work_mem
  ----------
   4MB
  (1 row)
  ```

  다음과 같이 정렬, 그룹화 및 기타 작업을 수행하기 전에 세션 수준에서 작업 메모리를 늘릴 수 있습니다.

  ```
  SET work_mem TO '1 GB';
  ```

  작업 메모리에 대한 자세한 내용은 PostgreSQL 설명서에서 리소스 소비를 참조하세요.
- 로그가 더 짧은 기간 동안 저장되도록 로그 보존 기간을 변경합니다. 자세한 방법은 Aurora PostgreSQL 데이터베이스 로그 파일을 참조하세요.

Aurora PostgreSQL DB 클러스터가 40TB보다 큰 경우 db.t2, db.t3 또는 db.t4g 인스턴스 클래스를 사용하지 마세요. T DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비프로덕션 서버에만 사용하는 것이 좋습니다. 자세한 내용은 DB 인스턴스 클래스 유형 섹션을 참조하세요.

Amazon Aurora PostgreSQL를 사용한 복제

아래에서 Amazon Aurora PostgreSQL을 이용한 복제에 관한 정보를 확인할 수 있습니다. 논리적 복제를 모니터링하고 사용하는 방법에 관한 내용도 포함되어 있습니다.

주제

Aurora 복제본 사용

Aurora복제본Aurora은 DB 클러스터의 독립 엔드포인트로서, 읽기 작업을 조정하고 가용성을 높이는 데 사용하기에 가장 적합합니다. Aurora DB 클러스터는 Aurora DB 클러스터의 AWS 리전의 가용 영역 전체에서 최대 15개의 Aurora 복제본까지 포함할 수 있습니다.

DB 클러스터 볼륨은 DB 클러스터의 데이터 사본들로 구성됩니다. 하지만 DB 클러스터의 기본 라이터 DB 인스턴스 및 Aurora 복제본에는 클러스터 볼륨의 데이터가 단 하나의 논리 볼륨으로 표시됩니다. Aurora 복제본에 대한 자세한 내용은 Aurora 복제본 단원을 참조하십시오.

Aurora 복제본은 클러스터 볼륨의 읽기 연산에 전적으로 사용되므로 읽기 조정에 유용합니다. 라이터 DB 인스턴스는 쓰기 작업을 관리합니다. 클러스터 볼륨은 Aurora PostgreSQL DB 클러스터에 있는 모든 인스턴스에 공유됩니다. 따라서 각 Aurora 복제본에 대해 데이터 사본을 복제하기 위해 추가 작업을 할 필요가 없습니다.

Aurora PostgreSQL에서 Aurora 복제본이 삭제되면 인스턴스 엔드포인트가 즉시 제거되고 Aurora 복제본이 리더 엔드포인트에서 제거됩니다. 삭제되는 Aurora 복제본을 실행하는 문이 있는 경우 3분의 유예 기간이 있습니다. 기존 문은 유예 기간 동안 정상적으로 완료할 수 있습니다. 유예 기간이 종료되면 Aurora 복제본이 종료 및 삭제됩니다.

Aurora PostgreSQL DB 클러스터는 Aurora Global Database를 사용하여 서로 다른 AWS 리전에서 Aurora 복제본을 지원합니다. 자세한 내용은 Amazon Aurora Global Database 사용 단원을 참조하십시오.

> **Note**
>
> 읽기 가용성 기능을 사용할 때는 DB 클러스터의 Aurora 복제본을 재부팅해야 하는 경우 수동으로 재부팅해야 합니다. 이 기능 재부팅 전에 생성된 DB 클러스터의 경우 라이터 DB 인스턴스가 Aurora 복제본을 자동으로 재부팅합니다. 자동 재부팅을 통해 DB 클러스터 전반에 걸친 읽기/쓰기 일관성을 보장하는 진입점이 다시 구성됩니다.

Aurora 복제본의 읽기 가용성 향상

Aurora PostgreSQL은 라이터 DB 인스턴스가 다시 시작되거나 Aurora 복제본이 쓰기 트래픽을 따라가지 못할 때 읽기 요청을 지속적으로 처리하여 DB 클러스터의 읽기 가용성을 향상시킵니다.

읽기 가용성 기능은 다음 버전의 Aurora PostgreSQL에서 기본적으로 사용할 수 있습니다.

- 16.1 이상의 모든 버전
- 15.2 이상의 15 버전
- 14.7 이상의 14 버전
- 13.10 이상의 13 버전
- 12.14 이상의 12 버전

읽기 가용성 기능은 다음 버전의 Aurora Global Database에서 지원됩니다.

- 16.1 이상의 모든 버전
- 15.4 이상의 15 버전
- 14.9 이상의 14 버전
- 13.12 이상의 13 버전
- 12.16 이상의 12 버전

이 출시 전에 이러한 버전 중 하나에서 생성된 DB 클러스터의 읽기 가용성 기능을 사용하려면 DB 클러스터의 라이터 인스턴스를 다시 시작하세요.

Aurora PostgreSQL DB 클러스터의 정적 파라미터를 수정하는 경우, 파라미터 변경 사항이 적용되도록 라이터 인스턴스를 다시 시작해야 합니다. 예를 들어 `shared_buffers` 값을 설정할 때 라이터 인스턴스를 다시 시작해야 합니다. Aurora 복제본의 읽기 가용성을 통해 DB 클러스터는 개선된 가용성을 유지하므로 라이터 인스턴스가 다시 시작할 때 영향이 줄어듭니다. 리더 인스턴스는 다시 시작되지 않고 읽기 요청에 계속 응답합니다. 정적 파라미터 변경 사항을 적용하려면 각 개별 리더 인스턴스를 재부팅합니다.

Aurora PostgreSQL DB 클러스터의 Aurora 복제본은 라이터와 다시 연결된 후 인 메모리 데이터베이스 상태로 빠르게 복구하여 라이터 다시 시작, 장애 조치, 느린 복제 및 네트워크 문제와 같은 복제 오류를 복구할 수 있습니다. 이 접근 방식을 통해 클라이언트 데이터베이스의 가용성을 유지하면서도 Aurora 복제본 인스턴스가 최신 스토리지 업데이트로 일관성을 달성할 수 있습니다.

복제 복구와 충돌하는 진행 중인 트랜잭션에서는 오류가 발생할 수 있지만, 리더가 라이터를 따라잡은 후에 클라이언트가 이러한 트랜잭션을 다시 시도할 수 있습니다.

Aurora 복제본 모니터링

라이터 연결이 끊긴 상태에서 복구될 때 Aurora 복제본을 모니터링할 수 있습니다. 아래 지표를 사용하여 리더 인스턴스에 대한 최신 정보를 확인하고 처리 중인 읽기 전용 트랜잭션을 추적하세요.

- 리더 인스턴스가 계속 연결되어 있는 동안 리더 인스턴스에 대한 최신 정보를 반환하도록 `aurora_replica_status` 기능이 업데이트되었습니다. `aurora_replica_status`에서 쿼리가 실행되는 DB 인스턴스에 해당하는 행의 마지막 업데이트 타임스탬프는 항상 비어 있습니다. 이는 리더 인스턴스에 최신 데이터가 있음을 나타냅니다.
- Aurora 복제본이 라이터 인스턴스와의 연결을 끊었다가 다시 연결되면 다음 데이터베이스 이벤트가 발생합니다.

```
Read replica has been disconnected from the writer instance and
reconnected.
```

- 복구 충돌로 인해 읽기 전용 쿼리가 취소되면 데이터베이스 오류 로그에 다음과 같은 오류 메시지가 하나 이상 표시될 수 있습니다.

```
Canceling statement due to conflict with recovery.

User query may not have access to page data to replica disconnect.

User query might have tried to access a file that no longer exists.

When the replica reconnects, you will be able to repeat your command.
```

제한 사항

읽기 가용성 기능을 사용하는 Aurora 복제본에는 다음 제한 사항이 적용됩니다.

- 복제 복구 중에 쓰기 인스턴스에서 데이터를 스트리밍할 수 없는 경우 보조 DB 클러스터의 Aurora 복제본이 다시 시작될 수 있습니다.
- Aurora 복제본은 온라인 복제 복구가 이미 진행 중이고 다시 시작될 경우 온라인 복제 복구를 지원하지 않습니다.
- Aurora 복제본은 DB 인스턴스가 트랜잭션 ID 랩어라운드에 가까워지면 다시 시작됩니다. 트랜잭션 ID 랩어라운드에 대한 자세한 내용은 트랜잭션 ID 랩어라운드 실패 방지를 참조하세요.

- 특정 상황에서 복제 프로세스가 차단되면 Aurora 복제본을 다시 시작할 수 있습니다.

Aurora PostgreSQL 복제 모니터링

읽기 조정과 고가용성은 최소 지연 시간에 따라 달라집니다. Amazon CloudWatch `ReplicaLag` 지표를 모니터링하면 Aurora 복제본의 Aurora PostgreSQL DB 클러스터 라이터 DB 인스턴스 지연 시간을 모니터링할 수 있습니다. Aurora 복제본은 라이터 DB 인스턴스와 동일한 클러스터 볼륨에서 읽으므로 `ReplicaLag` 지표가 Aurora PostgreSQL DB 클러스터에서와는 다른 의미를 갖습니다. Aurora 복제본의 `ReplicaLag` 지표는 라이터 DB 인스턴스 대비 Aurora 복제본의 페이지 캐시 지연 시간을 나타냅니다.

RDS 인스턴스 및 CloudWatch 지표 모니터링에 대한 자세한 내용은 [Amazon Aurora 클러스터에서 지표 모니터링](#) 단원을 참조하십시오.

Aurora에서의 PostgreSQL 논리적 복제 개요

PostgreSQL의 논리적 복제 기능을 Aurora PostgreSQL DB 클러스터와 함께 사용하면 전체 데이터베이스 인스턴스가 아닌 개별 테이블을 복제하고 동기화할 수 있습니다. 논리적 복제에서는 게시 및 구독 모델을 사용하여 원본에서 한 명 이상의 수신자에게 변경 내용을 복제합니다. 이 작업은 PostgreSQL 미리 쓰기 로그(WAL)의 변경 레코드를 사용하여 작동합니다. 소스 또는 게시자는 지정된 테이블에 대한 WAL 데이터를 한 명 이상의 수신자(구독자)에게 전송하여 변경 사항을 복제하고 구독자의 테이블을 게시자의 테이블과 동기화된 상태로 유지합니다. 게시자의 변경 사항 집합은 게시를 사용하여 식별됩니다. 구독자는 게시자의 데이터베이스 및 게시에 대한 연결을 정의하는 구독을 만들어 변경 사항을 적용합니다. 복제 슬롯은 이 체계에서 구독 진행 상황을 추적하는 데 사용되는 메커니즘입니다.

Aurora PostgreSQL DB 클러스터의 경우 WAL 레코드가 Aurora 스토리지에 저장됩니다. 논리적 복제 시나리오에서 게시자 역할을 하는 Aurora PostgreSQL DB 클러스터는 Aurora 스토리지에서 WAL 데이터를 읽고 디코딩하고 구독자에게 전송하여 변경 사항이 해당 인스턴스의 테이블에 적용될 수 있도록 합니다. 게시자는 논리적 디코더를 사용하여 구독자가 사용할 데이터를 디코딩합니다. 기본적으로 Aurora PostgreSQL DB 클러스터는 데이터를 전송할 때 네이티브 PostgreSQL `pgoutput` 플러그인을 사용합니다. 다른 논리적 디코더도 사용할 수 있습니다. 예를 들어, Aurora PostgreSQL는 WAL 데이터를 JSON으로 변환하는 [`wal2json`](#) 플러그인도 지원합니다.

Aurora PostgreSQL 버전 14.5, 13.8, 12.12 및 11.17부터 Aurora PostgreSQL은 라이트-스루 캐시를 통해 PostgreSQL 논리적 복제 프로세스를 보강하여 성능을 개선합니다. WAL 트랜잭션 로그는 버퍼에 로컬로 캐시되어 디스크 I/O의 양, 즉 논리적 디코딩 중에 Aurora 스토리지에서 읽는 양을 줄입니다. 기본적으로 라이트-스루 캐시는 Aurora PostgreSQL DB 클러스터에 대한 논리적 복제를 사용할 때마다

사용됩니다. Aurora는 캐시 관리에 사용할 수 있는 다양한 함수를 제공합니다. 자세한 내용은 Aurora PostgreSQL 논리적 복제 라이트-스루 캐시 모니터링 단원을 참조하십시오.

논리적 복제는 현재 사용 가능한 모든 Aurora PostgreSQL 버전에서 지원됩니다. 자세한 정보는 Aurora PostgreSQL 릴리스 정보에서 Amazon Aurora PostgreSQL 업데이트 내용을 참조하세요.

Babelfish for Aurora PostgreSQL은 다음 버전에서 논리적 복제를 지원합니다.

- 15.7 이상 버전
- 16.3 이상 버전

> **Note**
>
> Aurora PostgreSQL은 PostgreSQL 10에 도입된 기본 PostgreSQL 논리적 복제 기능 외에 `pglogical` 확장도 지원합니다. 자세한 내용은 pglogical을 사용하여 인스턴스 간 데이터 동기화 단원을 참조하십시오.

PostgreSQL 논리적 복제에 대한 자세한 내용은 PostgreSQL 설명서의 Logical replication(논리적 복제) 및 Logical decoding concepts(논리적 디코딩 개념)을 참조하세요.

Aurora PostgreSQL DB 클러스터의 논리적 복제 설정

논리적 복제를 설정하려면 `rds_superuser` 권한이 필요합니다. 다음 절차에 설명된 대로 필요한 파라미터를 설정하려면 사용자 지정 DB 클러스터 파라미터 그룹을 사용하도록 Aurora PostgreSQL DB 클러스터를 구성해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 단원을 참조하십시오.

Aurora PostgreSQL DB 클러스터에 논리적 복제를 설정하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Aurora PostgreSQL DB 클러스터를 선택합니다.
3. 구성 탭을 엽니다. 인스턴스 세부 정보 중에서 유형이 DB 클러스터 파라미터 그룹인 파라미터 그룹 링크를 찾아보세요.
4. 링크를 선택하여 Aurora PostgreSQL DB 클러스터와 연결된 사용자 지정 파라미터를 엽니다.
5. 파라미터 검색 필드에 `rds`를 입력하여 `rds.logical_replication` 파라미터를 찾습니다. 이 파라미터의 기본값은 `0`이며 기본적으로 꺼져 있음을 의미합니다.

6. 파라미터 편집을 선택하여 속성 값에 액세스한 다음 선택기에서 기능을 켜도록 1을 선택합니다. 예상 사용량에 따라 다음 파라미터의 설정을 변경해야 할 수도 있습니다. 하지만 대부분의 경우 기본값이면 충분합니다.

   - `max_replication_slots` - 이 파라미터를 적어도 계획한 총 논리적 복제 게시 및 구독 수와 동일한 값으로 설정합니다. AWS DMS를 사용하는 경우 이 파라미터를 적어도 클러스터에서 계획한 변경 데이터 캡처 작업과 논리적 복제 게시 및 구독 수를 합친 값과 동일해야 합니다.
   - `max_wal_senders` 및 `max_logical_replication_workers` - 이 파라미터를 적어도 활성화하려는 논리적 복제 슬롯 수 또는 변경 데이터 캡쳐를 위한 활성 AWS DMS 작업의 수와 동일한 값으로 설정합니다. 논리적 복제 슬롯을 비활성 상태로 두면 vacuum이 테이블에서 사용되지 않는 튜플을 제거할 수 없으므로 복제 슬롯을 모니터링하고 필요에 따라 비활성 슬롯을 제거하는 것이 좋습니다.
   - `max_worker_processes` - 이 파라미터를 적어도 `max_logical_replication_workers`, `autovacuum_max_workers` 및 `max_parallel_workers` 값의 합계와 동일한 값으로 설정합니다. 소규모 DB 인스턴스 클래스에서는 백그라운드 작업자 프로세스 수가 애플리케이션 워크로드에 영향을 줄 수 있으므로 `max_worker_processes`를 기본값보다 높게 설정한 경우 데이터베이스 성능을 모니터링합니다. (기본값은 `GREATEST(${DBInstanceVCPU*2},8}`의 결과입니다. 즉, 기본적으로 DB 인스턴스 클래스에 해당하는 CPU의 2배 또는 8 중 더 큰 값입니다.)

   > **Note**
   >
   > 고객이 생성한 DB 파라미터 그룹의 파라미터 값은 수정할 수 있지만, 기본 DB 파라미터 그룹의 파라미터 값은 변경할 수 없습니다.

7. Save changes(변경 사항 저장)를 선택합니다.
8. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 변경 사항이 적용되도록 합니다. Amazon RDS 콘솔에서 클러스터의 기본 DB 인스턴스를 선택하고 작업 메뉴에서 재부팅을 선택합니다.
9. 인스턴스를 사용할 수 있게 되면 다음과 같이 논리적 복제가 활성화되어 있는지 확인할 수 있습니다.

   a. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

   ```
   psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password --dbname=labdb
   ```

b. 다음 명령을 사용하여 논리적 복제가 활성화되었는지 확인합니다.

```
labdb=> SHOW rds.logical_replication;
 rds.logical_replication
-------------------------
 on
(1 row)
```

c. wal_level이 logical로 설정되어 있는지 확인합니다.

```
labdb=> SHOW wal_level;
  wal_level
-----------
 logical
(1 row)
```

논리적 복제를 사용하여 데이터베이스 테이블을 원본 Aurora PostgreSQL DB 클러스터의 변경 사항과 동기화된 상태로 유지하는 예는 예: Aurora PostgreSQL DB 클러스터에서 논리적 복제 사용 섹션을 참조하세요.

논리적 복제 비활성화

복제 작업을 완료한 후에는 복제 프로세스를 중지하고 복제 슬롯을 삭제한 다음 논리적 복제를 비활성화해야 합니다. 슬롯을 삭제하기 전에 슬롯이 더 이상 필요가 없는지 확인합니다. 활성 복제 슬롯은 삭제할 수 없습니다.

논리적 복제를 비활성화하는 방법

1. 모든 복제 슬롯을 삭제합니다.

   모든 복제 슬롯을 삭제하려면 게시자에 연결하고 다음 SQL 명령을 실행합니다.

```
SELECT pg_drop_replication_slot(slot_name)
  FROM pg_replication_slots
 WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
```

   이 명령을 실행할 때는 복제 슬롯을 활성화할 수 없습니다.

2. Aurora PostgreSQL DB 클러스터의 논리적 복제 설정에 설명된 대로 게시자와 연결된 사용자 지정 DB 클러스터 파라미터 그룹을 수정하되 `rds.logical_replication` 파라미터를 0으로 설정합니다.

   사용자 지정 파라미터 그룹에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 단원을 참조하세요.

3. `rds.logical_replication` 파라미터 변경 사항이 적용되도록 게시자 Aurora PostgreSQL DB 클러스터를 다시 시작합니다.

Aurora PostgreSQL 논리적 복제를 위한 라이트-스루 캐시 및 논리적 슬롯 모니터링

논리적 복제 라이트-스루 캐시를 모니터링하고 논리적 슬롯을 관리하여 Aurora PostgreSQL DB 클러스터의 성능을 개선합니다. 아래에서 라이트-스루 캐시 및 논리적 슬롯에 대한 자세한 내용을 찾아보세요.

주제

- Aurora PostgreSQL 논리적 복제 라이트-스루 캐시 모니터링
- Aurora PostgreSQL 논리적 슬롯 관리

Aurora PostgreSQL 논리적 복제 라이트-스루 캐시 모니터링

기본적으로 Aurora PostgreSQL 버전 14.5, 13.8, 12.12 및 11.17 이상에서는 라이트-스루 캐시를 사용하여 논리적 복제의 성능을 개선합니다. 라이트-스루 캐시가 없으면 Aurora PostgreSQL은 기본 PostgreSQL 논리적 복제 프로세스를 구현할 때 Aurora 스토리지 계층을 사용합니다. 이를 위해 WAL 데이터를 스토리지에 쓴 다음 스토리지에서 데이터를 다시 읽어 디코딩하고 대상(가입자)에 전송(복제)합니다. 그 결과 Aurora PostgreSQL DB 클러스터의 논리적 복제 중에 병목 현상이 발생할 수 있습니다.

라이트-스루 캐시는 Aurora 스토리지 계층에 대한 의존도를 최소화합니다. Aurora PostgreSQL은 이 계층에 지속적으로 쓰고 읽는 대신 버퍼를 사용하여 복제 프로세스 중에 사용할 논리적 WAL 스트림을 캐싱하므로 디스크에 액세스할 필요가 줄어듭니다. 이 버퍼는 논리적 복제에서 사용하는 기본 PostgreSQL 캐시이며, Aurora PostgreSQL DB 클러스터 파라미터에서 `rds.logical_wal_cache`로 식별됩니다.

(라이트-스루 캐시를 지원하는 버전의) Aurora PostgreSQL DB 클러스터에서 논리적 복제를 사용하는 경우, 캐시 적중률을 모니터링하여 사용 사례에 맞게 작동하는지 확인할 수 있습니다. 이렇게 하려면

다음 예제와 같이 psql을 사용하여 Aurora PostgreSQL DB 클러스터의 쓰기 인스턴스에 연결한 다음 Aurora 함수(aurora_stat_logical_wal_cache)를 사용해야 합니다.

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

이 함수는 다음과 같은 출력을 반환합니다.

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

가독성을 위해 last_reset_timestamp 값을 줄였습니다. 이 함수에 대한 자세한 내용은 aurora_stat_logical_wal_cache 단원을 참조하세요.

Aurora PostgreSQL는 라이트-스루 캐시를 모니터링하는 다음 두 가지 함수를 제공합니다.

- aurora_stat_logical_wal_cache 함수 – 참조 설명서는 aurora_stat_logical_wal_cache에서 확인할 수 있습니다.
- aurora_stat_reset_wal_cache 함수 – 참조 설명서는 aurora_stat_reset_wal_cache에서 확인할 수 있습니다.

자동으로 조정된 WAL 캐시 크기가 워크로드에 충분하지 않은 경우, rds.logical_wal_cache 값을 수동으로 변경할 수 있습니다. 다음을 고려하세요.

- rds.logical_replication 파라미터가 비활성화되면 rds.logical_wal_cache가 0으로 설정됩니다.
- rds.logical_replication 파라미터가 활성화되면 rds.logical_wal_cache의 기본값이 16MB입니다.
- rds.logical_wal_cache 파라미터는 정적이므로 변경 사항을 적용하려면 데이터베이스 인스턴스 재부팅이 필요합니다. 이 파라미터는 8Kb 블록으로 정의됩니다. 32kB 미만의 모든 양수 값은 32kB로 처리됩니다. wal_buffers에 대한 자세한 내용은 PostgreSQL 설명서에서 Write Ahead Log를 참조하세요.

Aurora PostgreSQL 논리적 슬롯 관리

스트리밍 활동이 pg_replication_origin_status 보기에 캡처됩니다. 이 보기의 내용을 보려면 다음과 같이 pg_show_replication_origin_status() 함수를 사용하면 됩니다.

```
SELECT * FROM pg_show_replication_origin_status();
```

다음 SQL 쿼리를 사용하여 논리적 슬롯 목록을 가져올 수 있습니다.

```
SELECT * FROM pg_replication_slots;
```

논리적 슬롯을 삭제하려면 다음 명령에서처럼 pg_drop_replication_slot를 슬롯 이름과 함께 사용합니다.

```
SELECT pg_drop_replication_slot('test_slot');
```

예: Aurora PostgreSQL DB 클러스터에서 논리적 복제 사용

다음 절차는 두 Aurora PostgreSQL DB 클러스터 간에 논리적 복제를 시작하는 방법을 보여줍니다. 게시자와 구독자 모두 [Aurora PostgreSQL DB 클러스터의 논리적 복제 설정](#)에 설명된 대로 논리적 복제를 구성해야 합니다.

지정된 게시자인 Aurora PostgreSQL DB 클러스터도 복제 슬롯에 대한 액세스를 허용해야 합니다. 그러려면 Aurora PostgreSQL DB 클러스터의 Amazon VPC 서비스를 기반으로 Virtual Public Cloud(VPC)와 연결된 보안 그룹을 수정합니다. 구독자의 VPC와 연결된 보안 그룹을 게시자의 보안 그룹에 추가하여 인바운드 액세스를 허용합니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [보안 그룹을 사용하여 리소스에 대한 트래픽 제어](#)를 참조하세요.

이러한 예비 단계가 완료되면 다음 절차에 설명된 대로 게시자에는 CREATE PUBLICATION, 구독자에는 CREATE SUBSCRIPTION PostgreSQL 명령을 사용할 수 있습니다.

두 Aurora PostgreSQL DB 클러스터 간에 논리적 복제를 시작하는 방법

이 단계에서는 Aurora PostgreSQL DB 클러스터에 예시 테이블을 생성할 데이터베이스가 있는 라이터 인스턴스가 있다고 가정합니다.

1. 게시자 Aurora PostgreSQL DB 클러스터에서

    a. 다음 SQL 문을 사용하여 테이블을 생성합니다.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

b. 다음 SQL 문을 사용해 게시자 데이터베이스에 데이터를 삽입합니다.

```
INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
```

c. 다음 SQL 문을 사용하여 테이블에 데이터가 있는지 확인합니다.

```
SELECT count(*) FROM LogicalReplicationTest;
```

d. 다음과 같이 `CREATE PUBLICATION` 문을 사용하여 이 테이블에 대한 게시를 생성합니다.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

2. 구독자 Aurora PostgreSQL DB 클러스터에서

a. 다음과 같이 게시자에서 생성한 것과 동일한 `LogicalReplicationTest` 테이블을 구독자에 생성합니다.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

b. 이 테이블이 비어 있는지 확인합니다.

```
SELECT count(*) FROM LogicalReplicationTest;
```

c. 구독을 만들어 게시자로부터 변경 사항을 받습니다. 게시자 Aurora PostgreSQL DB 클러스터에 다음 세부 정보를 사용해야 합니다.

- host - 게시자 Aurora PostgreSQL DB 클러스터의 라이터 DB 인스턴스입니다.
- 포트 - 라이터 DB 인스턴스가 수신 대기하는 포트입니다. PostgreSQL의 기본값은 5432입니다.
- dbname - 데이터베이스의 이름입니다.

```
CREATE SUBSCRIPTION testsub CONNECTION
   'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
 password=password'
   PUBLICATION testpub;
```

> **Note**
>
> 보안 모범 사례로 여기에 표시된 프롬프트 이외의 암호를 지정하는 것이 좋습니다.

구독이 생성된 후 논리적 복제 슬롯이 게시자 측에 생성됩니다.

d. 이 예시에서 초기 데이터가 구독자 측에서 복제된다는 것을 확인하려면 구독자 데이터베이스에서 다음 SQL 문을 사용하십시오.

```
SELECT count(*) FROM LogicalReplicationTest;
```

게시자에 대한 추가 변경 사항은 구독자에게로 복제됩니다.

논리적 복제는 성능에 영향을 미칩니다. 복제 작업이 완료된 후에는 논리적 복제를 해제하는 것이 좋습니다.

예: Aurora PostgreSQL 및 AWS Database Migration Service를 사용하는 논리적 복제

AWS Database Migration Service(AWS DMS)를 사용해 데이터베이스 또는 데이터베이스의 일부를 복제할 수 있습니다. AWS DMS를 사용해 데이터를 Aurora PostgreSQL 데이터베이스에서 다른 오픈 소스 또는 상용 데이터베이스로 마이그레이션합니다. AWS DMS에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서를 참조하십시오.

다음 예에서는 게시자인 Aurora PostgreSQL 데이터베이스에서 논리적 복제를 설정한 다음 마이그레이션을 위해 AWS DMS를 사용하는 방법을 보여줍니다. 이 예시에서는 예: Aurora PostgreSQL DB 클러스터에서 논리적 복제 사용에서 생성된 동일한 게시자 및 구독자를 사용합니다.

AWS DMS를 이용해 논리적 복제를 설정하려면 Amazon RDS에서 게시자 및 구독자에 대한 세부 정보를 얻어야 합니다. 특히 게시자의 라이터 DB 인스턴스와 구독자의 DB 인스턴스에 대한 세부 정보가 필요합니다.

게시자의 라이터 DB 인스턴스에 대해 다음 정보를 얻으십시오.

- Virtual Private Cloud(VPC) 식별자
- 서브넷 그룹

- 가용 영역(AZ)
- VPC 보안 그룹
- DB 인스턴스 ID

구독자의 DB 인스턴스에 대해 다음 정보를 얻으십시오.

- DB 인스턴스 ID
- 소스 엔진

Aurora PostgreSQL에서 논리적 복제를 위해 AWS DMS를 사용하려면

1. AWS DMS 작업을 할 게시자 데이터베이스를 준비합니다.

   이를 위해 PostgreSQL 10.x 이상 데이터베이스에서는 AWS DMS 래퍼 함수를 게시자 데이터베이스에 적용해야 합니다. 이와 관련된 사항과 후속 단계에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서의 [PostgreSQL 버전 10.x 이상을 AWS DMS의 소스로 사용](#)을 참조하세요.

2. AWS Management Console에 로그인한 다음 AWS DMS에서 https://console.aws.amazon.com/dms/v2 콘솔을 엽니다. 상단 오른쪽에서 게시자와 구독자가 위치한 리전과 동일한 AWS 리전을 선택합니다.

3. AWS DMS 복제 인스턴스를 생성합니다.

   게시자의 라이터 DB 인스턴스에 대한 값과 동일한 값을 선택합니다. 여기에는 다음 설정이 포함합니다.

   - VPC에서 라이터 DB 인스턴스의 VPC와 동일한 VPC를 선택합니다.
   - 복제 서브넷 그룹에서 라이터 DB 인스턴스의 서브넷 그룹과 동일한 서브넷 그룹을 선택합니다. 필요한 경우 새 것을 만듭니다.
   - 가용 영역에서 라이터 DB 인스턴스의 영역과 동일한 영역을 선택합니다.
   - VPC 보안 그룹에서 라이터 DB 인스턴스의 그룹과 동일한 그룹을 선택합니다.

4. 원본에 대해 AWS DMS 엔드포인트를 생성합니다.

   다음 설정을 사용해 게시자를 원본 엔드포인트로 지정합니다.

   - Endpoint type(엔드포인트 유형)에서 Source endpoint(원본 엔드포인트)를 선택합니다.
   - Select RDS DB Instance(RDS DB 인스턴스 선택)을 선택합니다.

- RDS Instance(RDS 인스턴스)에서 게시자의 라이터 DB 인스턴스의 DB 식별자를 선택합니다.
- 소스 엔진에서 postgres를 선택합니다.

5. 대상에 대해 AWS DMS 엔드포인트를 생성합니다.

   다음 설정을 사용해 구독자를 대상 엔드포인트로 지정합니다.

   - Endpoint type(엔드포인트 유형)에서 Target endpoint(대상 엔드포인트)를 선택합니다.
   - Select RDS DB Instance(RDS DB 인스턴스 선택)을 선택합니다.
   - RDS Instance(RDS 인스턴스)에서 구독자 DB 인스턴스의 DB 식별자를 선택합니다.
   - 소스 엔진의 값을 선택합니다. 예를 들어 구독자가 RDS PostgreSQL 데이터베이스인 경우 postgres를 선택합니다. 구독자가 Aurora PostgreSQL 데이터베이스인 경우 aurora-postgresql을 선택합니다.

6. AWS DMS 데이터베이스 마이그레이션 작업을 생성합니다.

   데이터베이스 마이그레이션 작업을 사용하여 마이그레이션할 데이터베이스 테이블을 지정하고, 대상 스키마를 사용해 데이터를 매핑하고, 대상 데이터베이스에 새 테이블을 생성합니다. 최소한 Task configuration(작업 구성)에 대해 다음 설정을 사용하십시오.

   - Replication instance(복제 인스턴스)에서 이전 단계에서 생성한 복제 인스턴스를 선택합니다.
   - Source database endpoint(원본 데이터베이스 엔드포인트)에서 이전 단계에서 생성한 게시자 원본을 선택합니다.
   - Target database endpoint(대상 데이터베이스 엔드포인트)에서 이전 단계에서 생성한 구독자 대상을 선택합니다.

   작업에 관한 나머지 세부 정보는 마이그레이션 프로젝트에 따라 다릅니다. DMS 태스크의 모든 세부 정보 지정에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서에서 [AWS DMS 태스크 사용]을 참조하세요.

작업이 생성되고 나면 AWS DMS에서 게시자에서 구독자로 데이터가 마이그레이션되기 시작합니다.

Aurora PostgreSQL의 로컬 쓰기 전달

로컬(클러스터 내) 쓰기 전달을 사용하면 애플리케이션이 Aurora 복제본에서 직접 읽기/쓰기 트랜잭션을 발행할 수 있도록 합니다. 그런 다음 쓰기 명령이 커밋될 라이터 DB 인스턴스로 전달됩니다. 가끔 쓰기가 발생하고 한 트랜잭션에서 최신 쓰기를 읽는 기능인 쓰기 후 읽기 일관성이 필요한 애플리케이션에 로컬 쓰기 전달을 사용할 수 있습니다.

쓰기 전달을 사용하지 않으면 애플리케이션이 트래픽을 적절한 엔드포인트로 전송하기 위해 두 세트의 데이터베이스 연결을 유지하면서 모든 읽기 및 쓰기 트래픽을 완전히 분할해야 합니다. 읽기 전용 복제본은 라이터 인스턴스로부터 비동기식으로 업데이트를 받습니다. 또한 복제 지연은 읽기 전용 복제본마다 다를 수 있으므로 모든 복제본에서 전역 읽기 일관성을 달성하는 것은 어렵습니다. 라이터 데이터베이스 인스턴스에서 읽기 후 쓰기 일관성이 필요한 모든 읽기를 처리해야 합니다. 또는 일관성을 보장하면서 확장성을 위해 여러 읽기 전용 복제본을 활용하려면 복잡한 사용자 지정 애플리케이션 로직을 개발해야 합니다.

쓰기 전달을 사용하면 이러한 트랜잭션을 분할하거나 라이터 인스턴스로만 전송할 필요가 없습니다. 또한 읽기 후 쓰기 일관성 일관성을 달성하기 위해 복잡한 애플리케이션 로직을 개발할 필요가 없습니다.

로컬 쓰기 전달은 Aurora PostgreSQL을 사용할 수 있는 모든 리전에서 사용할 수 있습니다. 다음과 같은 Aurora PostgreSQL 버전이 지원됩니다.

- 16.4 이상의 16 버전
- 15.8 이상의 15 버전
- 14.13 이상의 14 버전

로컬 쓰기 전달은 리전 내 복제본의 쓰기를 전달하는 데 사용됩니다. 글로벌 복제본의 쓰기를 전달하려면 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 섹션을 참조하세요.

주제

Aurora PostgreSQL의 로컬 쓰기 전달 제한 사항 및 고려 사항

현재 Aurora PostgreSQL의 로컬 쓰기 전달에는 다음과 같은 제한 사항이 적용됩니다.

- RDS 프록시에서는 로컬 쓰기 전달이 지원되지 않습니다.
- 특정 명령문은 쓰기 전달과 함께 Aurora PostgreSQL에서 사용할 때 허용되지 않거나 부실 결과를 생성할 수 있습니다. 또한 사용자 정의 함수 및 사용자 정의 프로시저는 지원되지 않습니다. 따라서

`EnableLocalWriteForwarding` 설정은 DB 클러스터에 대해 기본적으로 꺼져 있습니다. 이 설정을 켜기 전에, 애플리케이션 코드가 이러한 제한 사항의 영향을 받지 않는지 확인하십시오.

- 다음과 같은 종류의 SQL 문은 쓰기 전달에서 지원되지 않습니다.

> **Note**
>
> 이러한 문은 애플리케이션에서 암시적으로 사용하거나 PostgreSQL 프로토콜로 추론할 수 있습니다. 예를 들어 PL/SQL 예외 처리 시 지원되는 문이 아닌 SAVEPOINT를 사용할 수 있습니다.

  - `ANALYZE`
  - `CLUSTER`
  - `COPY`
  - 커서 - 커서는 지원되지 않으므로 로컬 쓰기 전달을 사용하기 전에 커서를 닫아야 합니다.
  - 데이터 정의 언어(DDL) 문
  - `GRANT`|`REVOKE`|`REASSIGN OWNED`|`SECURITY LABEL`
  - `LISTEN / NOTIFY`
  - `LOCK`
  - `SAVEPOINT`
  - `SELECT INTO`
  - `SET CONSTRAINTS`
  - 시퀀스 업데이트: `nextval()`, `setval()`
  - `TRUNCATE`
  - 2단계 커밋 명령: `PREPARE TRANSACTION`, `COMMIT PREPARED`, `ROLLBACK PREPARED`
  - 사용자 정의 함수 및 사용자 정의 절차.
  - `VACUUM`

다음과 같은 SQL 문을 쓰기 전달과 함께 사용할 수 있습니다.

- DML 명령문은 `INSERT ... SELECT` 명령문 또는 `DELETE ... WHERE` 명령문과 같이 여러 부분으로 구성될 수 있습니다. 이 경우 전체 명령문이 라이터 DB 인스턴스로 전달되어 그 위치에서 실행됩니다.

- INSERT, DELETE 및 UPDATE와 같은 데이터 조작 언어(DML) 문입니다.
- 이 목록에 있는 문이 포함된 EXPLAIN 문
- PREPARE 및 EXECUTE 문입니다.
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } 문.

로컬 쓰기 전달을 위한 Aurora PostgreSQL 구성

다음 섹션을 사용하여 Amazon Aurora PostgreSQL DB 클러스터에 대한 로컬 쓰기 전달을 활성화하고, 일관성 수준을 구성하며, 쓰기 전달을 사용하여 트랜잭션을 관리할 수 있습니다.

로컬 쓰기 전달 활성화

기본적으로 Aurora PostgreSQL DB 클러스터에는 로컬 쓰기 전달이 활성화되어 있지 않습니다. 인스턴스 수준이 아닌 클러스터 수준에서 직접 로컬 쓰기 전달을 활성화합니다.

콘솔

DB 클러스터를 만들거나 수정할 때 AWS Management Console을 사용하여 읽기 전용 복제본 쓰기 전달 아래의 로컬 쓰기 전달 켜기 확인란을 선택합니다.

AWS CLI

AWS CLI를 사용하여 로컬 쓰기 전달을 활성화하려면 `--enable-local-write-forwarding` 옵션을 사용합니다. 이 옵션은 `create-db-cluster` 명령을 사용하여 새 DB 클러스터를 생성할 때 작동합니다. `modify-db-cluster` 명령을 사용하여 기존 DB 클러스터를 수정할 때도 작동합니다. 이러한 동일한 CLI 명령과 함께 `--no-enable-local-write-forwarding` 옵션을 사용하여 로컬 쓰기 전달을 비활성화할 수 있습니다.

다음 예시에서는 로컬 쓰기 전달이 활성화된 Aurora PostgreSQL DB 클러스터를 생성합니다.

```
aws rds create-db-cluster \
--db-cluster-identifier write-forwarding-test-cluster \
--enable-local-write-forwarding \
--engine aurora-postgresql \
--engine-version 16.4 \
--master-username myuser \
--master-user-password mypassword \
--backup-retention 1
```

그런 다음 쓰기 전달을 사용할 수 있도록 라이터 및 리더 DB 인스턴스를 생성합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

RDS API

Amazon RDS API를 사용하여 로컬 쓰기 전달을 활성화하려면 `EnableLocalWriteForwarding` 파라미터를 `true`로 설정합니다. 이 파라미터는 `CreateDBCluster` 작업을 사용하여 새 DB 클러스터를 생성할 때 작동합니다. `ModifyDBCluster` 작업을 사용하여 기존 DB 클러스터를 수정할 때도 작동합니다. `EnableLocalWriteForwarding` 파라미터를 `false`로 설정하여 로컬 쓰기 전달을 비활성화할 수 있습니다.

데이터베이스 세션에 대한 로컬 쓰기 전달 활성화

`apg_write_forward.consistency_mode` 파라미터는 쓰기 전달을 가능하게 하는 DB 파라미터 및 DB 클러스터 파라미터입니다. 읽기 일관성 수준에 대해 `SESSION`, `EVENTUAL`, `GLOBAL` 또는 `OFF`를 지정할 수 있습니다. 일관성 수준에 대한 자세한 내용은 Aurora PostgreSQL에서 로컬 쓰기 전달의 일관성 및 격리 단원을 참조하세요.

이 파라미터에는 다음과 같은 규칙이 적용됩니다.

- 기본값은 `SESSION`입니다.
- `apg_write_forward.consistency_mode`를 `EVENTUAL`, `SESSION` 또는 `GLOBAL`로 설정하는 경우에만 로컬 쓰기 전달을 사용할 수 있습니다. 이 파라미터는 로컬 쓰기 전달이 활성화되어 있는 DB 클러스터의 리더 인스턴스에만 적용됩니다.
- 값을 `OFF`로 설정하면 세션에서 로컬 쓰기 전달이 비활성화됩니다.

Aurora PostgreSQL에서 로컬 쓰기 전달의 일관성 및 격리

읽기 전용 복제본에서 읽기 일관성 정도를 제어할 수 있습니다. 읽기 일관성 수준을 조정하여 세션에서 전달된 모든 쓰기 작업이 후속 쿼리 전에 읽기 전용 복제본에 표시되도록 할 수 있습니다. 또한 이 설정을 사용하여 읽기 전용 복제본의 쿼리에 항상 라이터 DB 인스턴스의 최신 업데이트가 표시되게 할 수 있습니다. 이는 다른 세션이나 다른 클러스터에서 제출된 경우에도 마찬가지입니다. 애플리케이션에 대해 이러한 유형의 동작을 지정하려면 세션 수준 파라미터 `apg_write_forward.consistency_mode`의 값을 적절히 선택합니다. `apg_write_forward.consistency_mode` 파라미터는 로컬 쓰기 전달이 활성화된 읽기 전용 복제본에만 영향을 미칩니다.

> **Note**
>
> `apg_write_forward.consistency_mode` 파라미터에 대해 SESSION, EVENTUAL, GLOBAL 또는 OFF 값을 지정할 수 있습니다. 기본적으로 이 값은 SESSION로 설정됩니다. 값을 OFF로 설정하면 쓰기 전달이 비활성화됩니다.

일관성 수준을 높이면 애플리케이션은 변경 사항이 읽기 전용 복제본에 전파될 때까지 더 오랜 시간 동안 대기합니다. 낮은 지연 시간과 쿼리가 실행되기 전에 다른 위치에서 변경한 내용을 완전히 사용할 수 있도록 보장하는 것 사이의 균형을 선택할 수 있습니다.

사용 가능한 각 일관성 보장 모드 설정의 효과는 다음과 같습니다.

- SESSION - 로컬 쓰기 전달을 사용하는 읽기 전용 복제본의 세션이 해당 세션에서 수행한 모든 변경의 결과를 표시합니다. 변경 내용은 트랜잭션이 커밋되었는지 여부와 상관없이 표시됩니다. 필요한 경우 쿼리는 전달된 쓰기 작업의 결과가 현재 리더 DB 인스턴스로 복제될 때까지 대기합니다. 쿼리는 현재 DB 클러스터 내의 다른 세션에서 수행한 쓰기 작업에서 업데이트된 결과가 나올 때까지 대기하지 않습니다.
- EVENTUAL - 로컬 쓰기 전달을 사용하는 읽기 전용 복제본의 세션이 복제 지연으로 인해 약간 오래된 데이터를 표시할 수 있습니다. 쓰기 작업이 라이터 DB 인스턴스에서 수행되어 읽기 전용 복제본에 복제될 때까지 동일한 세션에서의 쓰기 작업 결과가 표시되지 않습니다. 쿼리는 업데이트된 결과를 사용할 수 있을 때까지 대기하지 않습니다. 따라서 쿼리는 명령문의 타이밍 및 복제 지연 양에 따라 이전 데이터 또는 업데이트된 데이터를 검색할 수 있습니다.
- GLOBAL - 읽기 전용 복제본의 세션에 해당 세션에서 변경한 내용이 표시됩니다. 또한 라이터 DB 인스턴스 및 기타 읽기 전용 복제본에서 커밋된 모든 변경 내용도 표시됩니다. 각 쿼리는 세션 지연 양에 따라 다른 기간 동안 대기할 수 있습니다. 쿼리가 시작된 시간을 기준으로, 읽기 전용 복제본이 라이터 DB 인스턴스에서 커밋된 모든 데이터로 최신 상태가 되면 쿼리가 진행됩니다.

  > **Note**
  >
  > 글로벌 일관성 모드는 세션 내에서 실행되는 쿼리의 지연 시간에 영향을 미칩니다. 세션에서 쓰기 쿼리를 전송하지 않은 경우에도 대기를 수행합니다.
- OFF - 로컬 쓰기 전달이 비활성화되었습니다.

쓰기 전달을 사용하는 세션에서는 `REPEATABLE READ` 및 `READ COMMITTED` 격리 수준을 사용할 수 있습니다. 하지만 `SERIALIZABLE` 격리 수준은 지원되지 않습니다.

쓰기 전달과 관련된 모든 파라미터에 대한 자세한 내용은 쓰기 전달을 위한 기본 파라미터 설정 단원을 참조하십시오.

쓰기 전달을 사용한 트랜잭션 액세스 모드

트랜잭션 액세스 모드가 읽기 전용으로 설정된 경우 로컬 쓰기 전달이 사용되지 않습니다. DB 클러스터에 연결되어 있고 로컬 쓰기 전달이 활성화된 세션에 연결되어 있는 경우에만 읽기 쓰기를 사용하도록 액세스 모드를 설정할 수 있습니다.

트랜잭션 액세스 모드에 대한 자세한 내용은 트랜잭션 설정을 참조하세요.

Aurora PostgreSQL에 대한 로컬 쓰기 전달 작업

다음 섹션을 사용하여 데이터베이스 클러스터에 로컬 쓰기 전달이 활성화되어 있는지 확인하고, 호환성 고려 사항을 확인하며, 구성 가능한 파라미터 및 인증 설정을 참조할 수 있습니다. 이 정보는 Aurora PostgreSQL의 로컬 쓰기 전달 기능을 효과적으로 활용할 수 있는 세부 정보를 제공합니다.

> **Note**
>
> 로컬 쓰기 전달을 사용하는 클러스터의 라이터 인스턴스가 다시 시작되면 로컬 쓰기 전달을 사용하는 리더 인스턴스의 활성 상태이며 전달된 트랜잭션 및 쿼리가 모두 자동으로 닫힙니다. 라이터 인스턴스를 다시 사용할 수 있게 되면 이러한 트랜잭션을 다시 시도할 수 있습니다.

DB 클러스터에 로컬 쓰기 전달이 활성화되어 있는지 확인

DB 클러스터에서 로컬 쓰기 전달을 사용할 수 있는지 확인하려면 클러스터에 `LocalWriteForwardingStatus` 속성이 `enabled`로 되어 있는지 확인합니다.

AWS Management Console에서 클러스터의 세부 정보 페이지 구성 탭에서 로컬 읽기 복제본 쓰기 전달이 활성화됨 상태인 것을 볼 수 있습니다.

모든 클러스터에 대한 로컬 쓰기 전달 설정의 상태를 보려면 다음 AWS CLI 명령을 실행합니다.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
```

```
{
"LocalWriteForwardingStatus": "enabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-1"
},
{
"LocalWriteForwardingStatus": "disabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-2"
},
{
"LocalWriteForwardingStatus": "requested",
"DBClusterIdentifier": "test-global-cluster-2"
},
{
"LocalWriteForwardingStatus": "null",
"DBClusterIdentifier": "aurora-postgresql-v2-cluster"
}
]
```

DB 클러스터는 `LocalWriteForwardingStatus`에 대해 다음과 같은 값을 가질 수 있습니다.

- `disabled` - 로컬 쓰기 전달이 비활성화되었습니다.
- `disabling` - 로컬 쓰기 전달을 비활성화하는 중입니다.
- `enabled` - 로컬 쓰기 전달이 활성화되었습니다.
- `enabling` - 로컬 쓰기 전달을 활성화하는 중입니다.
- `null` - 이 DB 클러스터에서는 로컬 쓰기 전달을 사용할 수 없습니다.
- `requested` - 로컬 쓰기 전달이 요청되었지만 아직 활성 상태가 아닙니다.

쓰기 전달을 위한 기본 파라미터 설정

Aurora 클러스터 파라미터 그룹에는 로컬 쓰기 전달 기능에 대한 설정이 포함되어 있습니다. 이러한 파라미터는 클러스터 파라미터이므로 각 클러스터의 모든 DB 인스턴스에 이러한 변수의 동일한 값이 있어야 합니다. 이러한 파라미터에 대한 자세한 내용은 다음 표에 요약되어 있으며, 표 뒤에 사용 참고 사항이 나와 있습니다.

파라미터	범위	유형	기본값	유효값
`apg_write_forward.connect_timeout`	세션	초	30	0–2147483647

파라미터	범위	유형	기본값	유효값
apg_write_forward.consistency_mode	세션	enum	세션	SESSION, EVENTUAL, GLOBAL, 및 OFF
apg_write_forward.idle_in_transaction_session_timeout	세션	밀리초	86400000	0–2147483647
apg_write_forward.idle_session_timeout	세션	밀리초	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	전 세계	int	25	1–100

apg_write_forward.max_forwarding_connections_percent 파라미터는 리더에서 전달된 쿼리를 처리하기 위해 사용할 수 있는 데이터베이스 연결의 상한입니다. 이 값은 라이터 DB 인스턴스에 대한 max_connections 설정의 백분율로 표시됩니다. 예를 들어 max_connections가 800이고 apg_write_forward.max_forwarding_connections_percent가 10이면 라이터는 최대 80개의 동시 전달 세션을 허용합니다. 이러한 연결은 max_connections 설정을 통해 관리되는 동일한 연결 풀에서 발생합니다. 이 설정은 클러스터에 로컬 쓰기 전달이 활성화된 경우 라이터 DB 인스턴스에만 적용됩니다.

다음 설정을 사용하여 로컬 쓰기 전달 요청을 제어합니다.

- apg_write_forward.consistency_mode - 읽기 전용 복제본에서 읽기 일관성 정도를 제어하는 세션 수준 파라미터입니다. 유효한 값은 SESSION, EVENTUAL, GLOBAL 또는 OFF입니다. 기본적으로 이 값은 SESSION로 설정됩니다. 값을 OFF로 설정하면 세션에서 로컬 쓰기 전달이 비활성화됩니다. 일관성 수준에 대한 자세한 내용은 Aurora PostgreSQL에서 로컬 쓰기 전달의 일관성 및 격리 단원을 참조하세요. 이 파라미터는 로컬 쓰기 전달이 활성화되어 있는 리더 인스턴스에만 적용됩니다.
- apg_write_forward.connect_timeout - 라이터 DB 인스턴스에 연결할 때 읽기 전용 복제본이 연결을 끊기 전에 대기하는 최대 시간(초)입니다. 값이 0이면 무한정 대기할 수 있다는 뜻입니다.
- apg_write_forward.idle_in_transaction_session_timeout - 라이터 DB 인스턴스가 연결을 닫기 전에 트랜잭션이 열려 있는 읽기 전용 복제본에서 전달된 연결의 활동을 기다리는 시간(밀리초)입니다. 세션이 이 기간을 넘어 트랜잭션에서 유휴 상태로 유지되면 Aurora에서 세션이 종료됩니다. 0 값은 제한 시간을 비활성화합니다.

- `apg_write_forward.idle_session_timeout` - 라이터 DB 인스턴스가 연결을 닫기 전에 읽기 전용 복제본에서 전달된 연결의 활동을 기다리는 시간(밀리초)입니다. 세션이 이 기간을 넘어 유휴 상태로 유지되면 Aurora에서 세션이 종료됩니다. `0` 값은 제한 시간을 비활성화합니다.

rdswriteforwarduser

`rdswriteforwarduser`는 읽기 전용 복제본과 라이터 DB 인스턴스 간의 연결을 설정하는 데 사용할 사용자입니다.

> **Note**
>
> `rdswriteforwarduser`는 PUBLIC 역할을 통해 CONNECT 권한을 고객 데이터베이스에 상속합니다. PUBLIC 역할에 대한 권한이 취소된 경우 쓰기를 전달해야 하는 데이터베이스에 대한 CONNECT 권한을 부여해야 합니다.

Aurora PostgreSQL에서 로컬 쓰기 전달 모니터링

다음 섹션을 사용하면 관련 CloudWatch 지표 및 대기 이벤트를 포함하여 Aurora PostgreSQL 클러스터의 로컬 쓰기 전달을 모니터링하여 성능을 추적하고 잠재적 문제를 식별할 수 있습니다.

쓰기 전달을 위한 Amazon CloudWatch 지표 및 Aurora PostgreSQL 상태 변수

다음 Amazon CloudWatch 지표는 하나 이상의 읽기 전용 복제본에서 쓰기 전달을 사용하는 라이터 DB 인스턴스에 적용됩니다.

CloudWatch 지표	단위 및 설명
`AuroraLocal ForwardingWriterDMLThroughput`	개수(초당). 이 라이터 DB 인스턴스에서 초당 처리되는 전달된 DML 문 수입니다.
`AuroraLocalForwardingWriterOpenSessions`	개수. 전달된 쿼리를 처리하는 이 라이터 DB 인스턴스의 열린 세션 수입니다.
`AuroraLocal ForwardingWriterTotalSessions`	개수. 라이터 DB 인스턴스에 있는 전달된 세션의 총 개수입니다.

다음 CloudWatch 지표는 각 읽기 전용 복제본에 적용됩니다. 이러한 지표는 로컬 쓰기 전달이 활성화된 DB 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위 및 설명
`AuroraForwardingReplicaCommitThroughput`	개수(초당). 이 복제본이 초당 전달하는 세션의 커밋 수입니다.
`AuroraForwardingReplicaDMLLatency`	밀리초. 복제본에 있는 전달된 DML의 평균 응답 시간(밀리초)입니다.
`AuroraForwardingReplicaDMLThroughput`	개수(초당). 이 복제본에서 초당 처리되는 전달된 DML 문 수입니다.
`AuroraForwardingReplicaErrorSessionsLimit`	개수. 최대 연결 또는 최대 쓰기 전달 연결 한도에 도달하여 라이터 DB 인스턴스에서 거부한 세션 수입니다.
`AuroraForwardingReplicaOpenSessions`	개수. 복제본 인스턴스에서 로컬 쓰기 전달을 사용하는 세션 수입니다.
`AuroraForwardingReplicaReadWaitLatency`	밀리초. 복제본이 라이터 DB 인스턴스의 LSN과 일치할 때까지 기다리는 평균 대기 시간(밀리초)입니다. 리더 DB 인스턴스가 대기하는 정도는 `apg_write_forward.consistency_mode` 설정에 따라 다릅니다. 이 설정에 대한 자세한 내용은 the section called “Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터” 섹션을 참조하세요.

Aurora PostgreSQL에서 로컬 쓰기 전달을 위한 대기 이벤트

Amazon Aurora는 Aurora PostgreSQL에서 쓰기 전달을 사용할 때 다음과 같은 대기 이벤트를 생성합니다.

주제

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint

- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

이 `IPC:AuroraWriteForwardConnect` 이벤트는 읽기 전용 복제본의 백엔드 프로세스가 라이터 DB 인스턴스에 대한 연결이 열릴 때까지 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

이 이벤트는 읽기 전용 복제본에서 라이터 노드로의 연결 시도 횟수가 늘어날수록 증가합니다.

작업

읽기 전용 복제본에서 라이터 노드로의 동시 연결 수를 줄이세요.

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint` 이벤트는 전달된 쓰기 작업의 결과가 현재 리전으로 복제될 때까지 읽기 전용 복제본에서 노드에서 대기하는 시간을 나타냅니다. 이 이벤트는 세션 수준 `apg_write_forward.consistency_mode` 파라미터가 다음 중 하나로 설정된 경우에만 생성됩니다.

- `SESSION` - 읽기 전용 복제본에 대한 쿼리는 해당 세션에서 수행한 모든 변경의 결과를 대기합니다.
- `GLOBAL` - 읽기 전용 복제본에 대한 쿼리는 해당 세션의 변경 사항과 라이터 DB 인스턴스 및 읽기 전용 복제본 모두에서 커밋된 모든 변경 사항의 결과를 기다립니다.

`apg_write_forward.consistency_mode` 파라미터 설정에 대한 자세한 내용은 the section called “Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터” 섹션을 참조하세요.

대기 증가의 가능한 원인

대기 시간이 길어지는 일반적인 원인은 다음과 같습니다.

- Amazon CloudWatch `ReplicaLag` 지표로 측정한 바와 같이 복제 지연이 증가했습니다. 이 지표에 대한 자세한 정보는 Aurora PostgreSQL 복제 모니터링 섹션을 참조하세요.

- 라이터 DB 인스턴스 또는 읽기 전용 복제본에 대한 로드가 증가했습니다.

작업

애플리케이션 요구 사항에 따라 일관성 모드를 변경합니다.

IPC:AuroraWriteForwardExecute

`IPC:AuroraWriteForwardExecute` 이벤트는 읽기 전용 복제본의 백엔드 프로세스가 전달된 쿼리가 완료되고 DB 클러스터의 라이터 노드로부터 결과를 얻기를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 라이터 노드에서 많은 수의 행을 가져옵니다.
- 라이터 노드와 읽기 전용 복제본 사이의 네트워크 지연 시간이 증가하면 읽기 전용 복제본이 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 읽기 전용 복제본의 로드가 증가하면 읽기 전용 복제본에서 라이터 노드로 쿼리 요청을 전송하는 것이 지연될 수 있습니다.
- 라이터 노드의 로드가 증가하면 라이터 노드에서 읽기 전용 복제본으로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 필요한 데이터만 검색하도록 쿼리를 최적화합니다.
- 데이터 조작 언어(DML) 작업을 최적화하여 필요한 데이터만 수정합니다.
- 읽기 전용 복제본 또는 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

`IPC:AuroraWriteForwardGetGlobalConsistencyPoint` 이벤트는 글로벌 일관성 모드를 사용하는 읽기 전용 복제본의 백엔드 프로세스가 쿼리를 실행하기 전에 라이터 노드로부터 글로벌 일관성 보장 지점을 가져오기를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 읽기 전용 복제본과 라이터 노드 사이의 네트워크 지연 시간이 증가하면 읽기 전용 복제본이 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 읽기 전용 복제본의 로드가 증가하면 읽기 전용 복제본에서 라이터 노드로 쿼리 요청을 전송하는 것이 지연될 수 있습니다.
- 라이터 노드의 로드가 증가하면 라이터 노드에서 읽기 전용 복제본으로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 애플리케이션 요구 사항에 따라 일관성 모드를 변경합니다.
- 읽기 전용 복제본 또는 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactAbort

`IPC:AuroraWriteForwardXactAbort` 이벤트는 읽기 전용 복제본의 백엔드 프로세스가 원격 정리 쿼리 결과를 기다리고 있을 때 발생합니다. 쓰기 전달 트랜잭션이 중단된 후 프로세스를 적절한 상태로 되돌리기 위해 정리 쿼리가 실행됩니다. Amazon Aurora는 오류가 발견되었거나 사용자가 명시적인 `ABORT` 명령을 실행했거나 실행 중인 쿼리를 취소했을 때 이러한 작업을 수행합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 읽기 전용 복제본과 라이터 노드 사이의 네트워크 지연 시간이 증가하면 읽기 전용 복제본이 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 읽기 전용 복제본의 로드가 증가하면 읽기 전용 복제본에서 라이터 노드로 정리 쿼리 요청을 전송하는 것이 지연될 수 있습니다.
- 라이터 노드의 로드가 증가하면 라이터 노드에서 읽기 전용 복제본으로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 트랜잭션이 중단된 원인을 조사하세요.
- 읽기 전용 복제본 또는 라이터 DB 인스턴스가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactCommit

`IPC:AuroraWriteForwardXactCommit` 이벤트는 읽기 전용 복제본의 백엔드 프로세스가 전달된 커밋 트랜잭션 명령의 결과를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 읽기 전용 복제본과 라이터 노드 사이의 네트워크 지연 시간이 증가하면 읽기 전용 복제본이 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 읽기 전용 복제본의 로드가 증가하면 읽기 전용 복제본에서 라이터 노드로 쿼리 요청을 전송하는 것이 지연될 수 있습니다.
- 라이터 노드의 로드가 증가하면 라이터 노드에서 읽기 전용 복제본으로의 데이터 전송이 지연될 수 있습니다.

작업

읽기 전용 복제본 또는 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactStart

`IPC:AuroraWriteForwardXactStart` 이벤트는 읽기 전용 복제본의 백엔드 프로세스가 전달된 시작 트랜잭션 명령의 결과를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 읽기 전용 복제본과 라이터 노드 사이의 네트워크 지연 시간이 증가하면 읽기 전용 복제본이 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 읽기 전용 복제본의 로드가 증가하면 읽기 전용 복제본에서 라이터 노드로 쿼리 요청을 전송하는 것이 지연될 수 있습니다.

- 라이터 노드의 로드가 증가하면 라이터 노드에서 읽기 전용 복제본으로의 데이터 전송이 지연될 수 있습니다.

작업

읽기 전용 복제본 또는 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

Aurora PostgreSQL을 Amazon Bedrock의 지식 기반으로 사용

Aurora PostgreSQL DB 클러스터를 Amazon Bedrock용 지식 기반으로 사용할 수 있습니다. 자세한 내용은 Amazon Aurora에서 벡터 저장소 생성을 참조하세요. 지식 기반은 Amazon S3 버킷에 저장된 비정형 텍스트 데이터를 자동으로 가져와 텍스트 청크 및 벡터로 변환하고 PostgreSQL 데이터베이스에 저장합니다. 생성형 AI 애플리케이션을 사용하면 Amazon Bedrock용 에이전트를 사용하여 지식 기반에 저장된 데이터를 쿼리하고 이러한 쿼리의 결과를 사용하여 기본 모델에서 제공하는 답변을 보강할 수 있습니다. 이 워크플로를 검색 증강 생성(RAG)이라고 합니다. RAG에 대한 자세한 내용은 검색 증강 생성(RAG)을 참조하세요.

Aurora PostgreSQL을 사용하여 RAG를 통한 생성형 AI 애플리케이션을 구축하는 방법에 대한 자세한 내용은 이 블로그 게시물을 참조하세요.

주제

사전 조건

Aurora PostgreSQL 클러스터를 Amazon Bedrock용 지식 기반으로 사용하려면 다음 사전 조건을 숙지하세요. 개괄적인 수준에서 Bedrock과 함께 사용할 수 있도록 다음 서비스를 구성해야 합니다.

- Amazon Aurora PostgreSQL DB 클러스터는 다음 버전 중 하나로 생성됩니다.
  - 16.1 이상의 모든 버전
  - 15.4 이상 버전

  - 14.9 이상 버전
  - 13.12 이상 버전
  - 12.16 이상 버전

> **Note**
> 대상 데이터베이스에서 `pgvector` 확장을 활성화하고 버전 0.5.0 이상을 사용해야 합니다. 자세한 내용은 HNSW 인덱싱이 포함된 pgvector v0.5.0을 참조하세요.

- RDS Data API
- AWS Secrets Manager에서 사용자가 관리합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 섹션을 참조하세요.

Aurora PostgreSQL을 Amazon Bedrock의 지식 기반으로 사용하도록 준비

아래 섹션에 설명된 단계에 따라 Amazon Bedrock용 지식 기반으로 사용할 Aurora PostgreSQL을 준비하세요.

Aurora PostgreSQL 생성 및 구성

Aurora PostgreSQL DB 클러스터를 사용하여 Amazon Bedrock을 구성하려면 먼저 Aurora PostgreSQL DB 클러스터를 생성하고 Amazon Bedrock으로 구성하는 데 필요한 중요 필드를 기록해 두어야 합니다. Aurora PostgreSQL DB 클러스터 생성에 대한 자세한 정보는 Aurora PostgreSQL DB 클러스터 생성 및 연결 섹션을 참조하세요.

- Aurora PostgreSQL DB 클러스터를 생성하는 동안 Data API를 활성화합니다. 지원되는 버전에 대한 자세한 내용은 Amazon RDS Data API 사용 섹션을 참조하세요.
- Aurora PostgreSQL DB 클러스터의 Amazon 리소스 이름(ARN)을 기록해 둡니다. Amazon Bedrock과 함께 사용할 DB 클러스터를 구성하는 데 필요합니다. 자세한 내용은 Amazon 리소스 이름(ARN)을 참조하세요.

데이터베이스에 연결 및 pgvector 설치

모든 연결 유틸리티를 사용하여 Aurora PostgreSQL에 연결할 수 있습니다. 이러한 유틸리티에 대한 자세한 내용은 Amazon Aurora PostgreSQL DB 클러스터에 연결 섹션을 참조하세요. 또는 RDS 콘솔 쿼리 편집기를 사용하여 쿼리를 실행할 수 있습니다. 쿼리 편집기를 사용하려면 RDS Data API가 활성화된 Aurora DB 클러스터가 필요합니다.

1. 마스터 사용자로 데이터베이스에 로그인하고 pgvector를 설정합니다. 확장이 설치되지 않은 경우 다음 명령을 사용하세요.

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   HNSW 인덱싱을 지원하는 `pgvector` 0.5.0 이상 버전을 사용하세요. 자세한 내용은 [HNSW 인덱싱이 포함된 pgvector v0.5.0](#)을 참조하세요.

2. 다음 명령을 사용하여 현재 설치된 `pg_vector` 버전을 확인합니다.

   ```
   SELECT extversion FROM pg_extension WHERE extname='vector';
   ```

데이터베이스 객체 및 권한 설정

1. Bedrock이 데이터를 쿼리하는 데 사용할 수 있는 특정 스키마를 만듭니다. 다음 명령을 실행해 스키마를 생성합니다.

   ```
   CREATE SCHEMA bedrock_integration;
   ```

2. Bedrock이 데이터를 쿼리하는 데 사용할 수 있는 새 역할을 생성합니다. 다음 명령을 실행해 새 역할을 생성합니다.

   ```
   CREATE ROLE bedrock_user WITH PASSWORD 'password' LOGIN;
   ```

   > **Note**
   >
   > 추후 Secrets Manager 암호를 생성할 때 필요하므로, 이 암호를 기록해 두세요.

   `psql` 클라이언트를 사용하는 경우 다음 명령을 사용하여 새 역할을 생성합니다.

   ```
   CREATE ROLE bedrock_user LOGIN;
   \PASSWORD password;
   ```

3. `bedrock_integration` 스키마를 관리할 `bedrock_user` 권한을 부여하세요. 이렇게 하면 스키마 내에 표나 인덱스를 생성할 수 있습니다.

   ```
   GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
   ```

4. bedrock_user로 로그인하고 에서 bedrock_integration schema에 테이블을 생성합니다.

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(n), chunks text, metadata json, custom_metadata jsonb);
```

이 명령은 Titan 임베딩을 사용하여 bedrock_integration 스키마에 bedrock_kb 테이블을 만듭니다.

vector(*n*) 데이터 유형의 n을 사용 중인 임베딩 모델의 적절한 차원으로 바꿉니다. 아래 권장 사항을 사용하여 차원을 선택합니다.

- Titan v2 모델의 경우 vector(1024) 또는 vector(512) 또는 vector (256)를 사용합니다. 자세한 내용은 Amazon Titan Embeddings Text를 참조하세요.
- Titan v1.2 모델의 경우 vector(1536)를 사용합니다. 자세한 내용은 Amazon Titan Multimodal Embeddings G1을 참조하세요.
- Cohere Embed 모델의 경우 vector(1024)를 사용합니다. 자세한 내용은 Cohere Embed models을 참조하세요.
- Cohere Embed Multilingual v3의 경우 vector(1024)를 사용합니다.

처음 4개 열은 필수입니다. 메타데이터 처리를 위해 Bedrock은 메타데이터 파일의 데이터를 custom_metadata 열에 씁니다. 메타데이터 및 필터링을 사용하려는 경우 이 열을 만드는 것이 좋습니다. custom_metadata 열을 만들지 않는 경우 수집을 시작하기 전에 테이블의 각 메타데이터 속성에 대해 개별 열을 추가합니다. 자세한 내용은 Configure and customize queries and response generation을 참조하세요.

5. 다음 단계에 따라 Bedrock이 데이터를 쿼리하는 데 사용하는 필수 인덱스를 만듭니다.

- Bedrock이 데이터를 쿼리하는 데 사용할 수 있는 코사인 연산자를 사용하여 인덱스를 만듭니다.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops);
```

- 병렬 인덱스 구축을 사용하는 pgvector 0.6.0 이상 버전의 경우 ef_construction 값을 256으로 설정하는 것이 좋습니다.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops) WITH (ef_construction=256);
```

- Bedrock이 텍스트 데이터를 쿼리하는 데 사용할 수 있는 인덱스를 만듭니다.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (to_tsvector('simple',
 chunks));
```

- 사용자 지정 메타데이터에 대한 열을 만든 경우 Bedrock이 메타데이터를 쿼리하는 데 사용할 수 있는 인덱스를 만듭니다.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (custom_metadata);
```

Secrets Manager에서 보안 암호 생성

Secrets Manager를 사용하면 Aurora 자격 증명을 저장하여 애플리케이션으로 안전하게 전송할 수 있습니다. Aurora PostgreSQL DB 클러스터를 생성할 때 AWS Secrets Manager 옵션을 선택하지 않았다면, 지금 암호를 생성할 수 있습니다. AWS Secrets Manager 데이터베이스 암호 생성에 대한 자세한 내용은 AWS Secrets Manager 데이터베이스 암호를 참조하세요.

Bedrock 콘솔에서 지식 기반 만들기

지식 기반의 벡터 저장소로 사용할 Aurora PostgreSQL을 준비하는 동안 Amazon Bedrock 콘솔에 제공해야 하는 다음과 같은 세부 정보를 수집해야 합니다.

- Amazon Aurora DB 클러스터 ARN – DB 클러스터의 ARN입니다.
- 암호 ARN - DB 클러스터용 AWS Secrets Manager 키의 ARN입니다.
- 데이터베이스 이름 - 데이터베이스의 이름입니다. 예를 들어, 기본 데이터베이스 *postgres*를 사용할 수 있습니다.
- 표 이름 – 다음과 유사한 명령을 사용하여 표를 생성할 때 스키마에 적합한 이름을 제공하는 것이 좋습니다.

  ```
  CREATE TABLE bedrock_integration.bedrock_kb;
  ```

  이 명령은 `bedrock_integration` 스키마에 `bedrock_kb` 표를 만듭니다.
- 표를 생성할 때는 지정된 열과 데이터 유형으로 구성해야 합니다. 표에 나열된 이름 대신 원하는 열 이름을 사용할 수 있습니다. 지식 기반을 설정하는 동안 참조용으로 선택한 이름을 기록해 두세요.

열 명칭	데이터 유형	설명
id	UUID 프라이머리 키	각 레코드의 고유 식별자를 포함합니다.
청크	텍스트	데이터 소스의 원시 텍스트 청크를 포함합니다.
임베딩	벡터	데이터 소스의 벡터 임베딩을 포함합니다.
metadata	JSON	소스 속성을 가져오고 데이터 모으기 및 쿼리를 지원하는 데 필요한 메타데이터를 포함합니다.
custom_metadata	JSONB	(선택 사항) Amazon Bedrock이 데이터 소스에서 메타데이터 세부 정보를 쓰는 대상 열을 정의합니다.

이러한 세부 정보를 사용하여 이제 Bedrock 콘솔에서 지식 기반을 만들 수 있습니다. 벡터 인덱스 설정 및 지식 기반 정보 만들기에 대한 자세한 내용은 Create a vector store in Amazon Aurora 및 Create a vector store in Amazon Aurora를 참조하세요.

Aurora를 지식 기반으로 추가한 후에는 이제 검색 및 쿼리를 위해 데이터 소스를 수집할 수 있습니다. 자세한 내용은 Ingest your data sources into the Knowledge Base를 참조하세요.

Amazon Bedrock용 Aurora PostgreSQL 지식 기반 빠른 생성

Amazon Bedrock의 검색 증강 생성(RAG) 워크플로는 Aurora PostgreSQL 데이터베이스에 저장된 벡터 데이터를 사용하여 콘텐츠 검색을 지원합니다. 이전에는 Aurora PostgreSQL을 Bedrock 지식 기반용 벡터 데이터 저장소로 설정하는 것이 다단계 프로세스였기 때문에 다양한 사용자 인터페이스에서 수많은 수동 작업이 필요했습니다. 이로 인해 데이터 과학자와 개발자가 Bedrock 프로젝트에 Aurora를 활용하는 것이 어려웠습니다.

사용자 경험을 개선하기 위해 AWS는 설정 프로세스를 간소화하는 새로운 CloudFormation 기반 빠른 생성 옵션을 만들었습니다. 이제 Aurora 빠른 생성을 통해 클릭 한 번으로 사전 구성된 Aurora PostgreSQL DB 클러스터를 Amazon Bedrock 지식 기반용 벡터 저장소로 프로비저닝할 수 있습니다.

주제

- [지원되는 리전 및 Aurora PostgreSQL 버전]
- [빠른 생성 프로세스 이해]
- [Aurora 빠른 생성 사용의 이점]
- [Aurora 빠른 생성 프로세스의 제한 사항]

지원되는 리전 및 Aurora PostgreSQL 버전

Aurora 빠른 생성 옵션은 Amazon Bedrock 지식 기반을 지원하는 모든 AWS 리전에서 사용할 수 있습니다. 기본적으로 버전 15.7의 Aurora PostgreSQL DB 클러스터를 만듭니다. 지원되는 리전에 대한 자세한 내용은 Supported models and regions for Amazon Bedrock Knowledge Bases를 참조하세요.

빠른 생성 프로세스 이해

빠른 생성 프로세스는 Amazon Aurora PostgreSQL 데이터베이스를 Bedrock 지식 기반의 벡터 데이터 저장소로 설정하기 위해 다음 리소스를 자동으로 프로비저닝합니다.

계정의 Aurora PostgreSQL DB 클러스터로, 기본 설정으로 구성되어 있습니다.

- ACU(Aurora 용량 단위)는 0~16으로 설정됩니다. 이를 통해 벡터 저장소를 사용하지 않을 때 0으로 스케일 다운하여 컴퓨팅 비용을 절감할 수 있습니다. ACU는 나중에 Amazon RDS 콘솔에서 조정할 수 있습니다.
- Aurora에 저장된 Bedrock 벡터 임베딩에 대한 유사성 척도로 유클리드 거리를 사용하는 Hierarchical Navigable Small World(HNSW) 인덱스입니다.
- DB 인스턴스는 서버리스 v2 인스턴스입니다.
- 클러스터는 기본 VPC 및 서브넷과 연결되며 RDS 데이터 API가 활성화되어 있습니다.
- 클러스터 관리자 자격 증명은 AWS Secrets Manager에서 관리합니다.

기본 설정 외에도 다음 설정이 구성되어 있습니다. 프로세스를 진행하면 워크플로를 설명하는 화면이 표시됩니다.

- Aurora 클러스터에 필요한 데이터베이스 객체 시딩:
  - Bedrock 지식 기반에 필요한 pgvector 확장, 스키마, 역할 및 테이블을 만듭니다.

- Bedrock이 클러스터와 상호 작용할 수 있도록 제한된 권한의 데이터베이스 사용자를 등록합니다.
- 리소스 프로비저닝 프로세스 전반에 걸쳐 진행률 배너가 표시되므로 다음 하위 이벤트의 상태를 추적할 수 있습니다.
  - Aurora 클러스터 만들기
  - Aurora 클러스터에 시딩
  - 지식 기반 만들기

  페이지를 벗어났다가 돌아와도 지식 기반이 완전히 만들어질 때까지 배너가 계속 표시됩니다.
- 진행률 배너에서 `View details`을 클릭하여 각 단계의 상태를 확인할 수 있습니다. 지식 기반 만들기 중 이벤트에 대한 자세한 내용은 세부 정보 보기 화면에서 CloudFormation 링크를 선택합니다. 프로세스가 완료되면 새 Bedrock 지식 기반을 사용할 준비가 됩니다.
- 모든 빠른 생성 리소스의 스택 ID는 태그를 참조해야 하는 경우 Bedrock 지식 기반의 태그에서 찾을 수 있습니다.

벡터 저장소로 사용되는 새로 프로비저닝된 Aurora 클러스터에 대한 구성이 포함된 Bedrock 지식 기반이 만들어집니다.

Aurora 빠른 생성 사용의 이점

- CloudFormation 기반 빠른 생성 프로세스는 Aurora를 벡터 저장소로 사용하는 데 필요한 시간과 복잡성을 크게 줄입니다.
- Aurora는 사용하지 않을 때 컴퓨팅 요금을 0으로 스케일 다운할 수 있는 기능과 함께 뛰어난 성능, 벡터 확장성 및 비용 이점을 제공합니다.
- 빠른 생성 프로세스는 엔드투엔드 경험을 간소화하므로 Aurora를 사용하여 Bedrock 지식 기반을 쉽게 만들고 구성할 수 있습니다.
- 고객은 CloudFormation 템플릿을 기반으로 구축하여 자체 구성으로 프로비저닝을 사용자 지정할 수 있습니다.

Aurora 빠른 생성 프로세스의 제한 사항

- Aurora 빠른 생성 옵션을 사용하면 DB 클러스터가 기본 구성으로 프로비저닝됩니다. 그러나 이러한 기본 설정은 특정 요구 사항 또는 사용 목적에 맞지 않을 수 있습니다. 빠른 생성은 프로비저닝 프로세스 중에 구성을 수정하는 옵션을 제공하지 않습니다. 구성은 배포 경험을 간소화하도록 자동으로 설정됩니다. Aurora DB 클러스터 구성을 사용자 지정해야 하는 경우 Amazon RDS 콘솔에서 빠른 생성을 통한 초기 배포 후 사용자 지정할 수 있습니다.

- 빠른 생성 흐름은 설정 프로세스를 단순화하지만 Aurora DB 클러스터를 만드는 데 걸리는 시간은 수동 배포와 마찬가지로 여전히 약 10분입니다. 이는 Aurora 인프라를 프로비저닝하는 데 필요한 시간 때문입니다.
- 빠른 생성 옵션은 실험 및 빠른 설정을 위해 설계되었습니다. 빠른 생성을 통해 만들어진 리소스는 프로덕션 사용에 적합하지 않을 수 있으며 VPC의 프로덕션 환경으로 직접 마이그레이션할 수 없습니다.

Amazon Aurora PostgreSQL를 다른 AWS 서비스와 통합

Aurora PostgreSQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora는 다른 AWS 서비스를 통합합니다. Aurora PostgreSQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- Amazon RDS 성능 개선 도우미(Performance Insights)에서 Aurora PostgreSQL DB 인스턴스를 빠르게 수집하여 살펴보고 성능을 평가합니다. 성능 개선 도우미(Performance Insights)는 기존 Amazon RDS 모니터링 기능을 확장한 것으로서 데이터베이스 성능을 표시하여 성능 문제를 분석하는 데 효과적입니다. 성능 개선 도우미 대시보드가 데이터베이스 부하를 시각화하여 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 부하를 필터링합니다. 성능 개선 도우미(Performance Insights)에 대한 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.
- Aurora PostgreSQL DB 클러스터에서 로그 데이터를 Amazon CloudWatch Logs에 게시하도록 구성합니다. CloudWatch Logs는 로그 레코드에 내구성이 우수한 스토리지를 제공합니다. CloudWatch Logs를 통해 로그 데이터에 대한 실시간 분석을 수행할 수 있고, CloudWatch를 사용하여 경보를 만들고 지표를 볼 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 섹션을 참조하세요.
- Amazon S3 버킷에서 Aurora PostgreSQL DB 클러스터로 데이터를 가져오거나 Aurora PostgreSQL DB 클러스터에서 Amazon S3 버킷으로 데이터를 내보냅니다. 자세한 내용은 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터 및 Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기 섹션을 참조하세요.
- SQL 언어를 사용해 기계 학습 기반 예측을 데이터베이스 애플리케이션에 추가할 수 있습니다. Aurora 기계 학습은 Aurora 데이터베이스와 AWS 기계 학습(ML) 서비스인 SageMaker AI와 Amazon Comprehend 간의 고도로 최적화된 통합을 활용합니다. 자세한 내용은 Aurora PostgreSQL과 함께 Amazon Aurora 기계 학습 사용 단원을 참조하십시오.
- Aurora PostgreSQL DB 클러스터에서 AWS Lambda 함수를 호출합니다. 이렇게 하려면 Aurora PostgreSQL와 함께 제공된 `aws_lambda` PostgreSQL 확장을 사용합니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터에서 AWS Lambda 함수 호출 단원을 참조하십시오.

- Amazon Redshift와 Aurora PostgreSQL의 쿼리를 통합합니다. 자세한 내용은 Amazon Redshift 데이터베이스 개발자 가이드의 PostgreSQL에 대한 연합 쿼리 사용 시작하기를 참조하세요.

PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터

Amazon Simple Storage Service를 사용하여 저장된 데이터를 RDS for PostgreSQL DB 인스턴스의 테이블로 가져올 수 있습니다. 이 작업을 수행하려면 먼저 Aurora PostgreSQL aws_s3 확장을 설치해야 합니다. 이 확장은 Amazon S3 버킷에서 데이터를 가져오는 데 사용하는 기능을 제공합니다. 버킷은 객체 또는 파일에 대한 Amazon S3 컨테이너입니다. 데이터는 쉼표로 구분된 값 (CSV) 파일, 텍스트 파일 또는 압축 (gzip) 파일에 있을 수 있습니다. 다음에서는 확장 프로그램을 설치하는 방법과 Amazon S3에서 테이블로 데이터를 가져오는 방법을 알아볼 수 있습니다.

Amazon S3를 로 가져오려면 데이터베이스에서 PostgreSQL 버전 10.7 이상을 실행 중이어야 합니다. Aurora PostgreSQL.

Amazon S3 데이터가 저장되지 않은 경우 먼저 버킷을 생성하고 데이터를 저장해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 다음 주제를 참조하세요.

- 버킷 생성
- 버킷에 객체 추가

Amazon S3에서 계정 간 가져오기가 지원됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 계정 간 권한 부여를 참조하세요.

S3에서 데이터를 가져오는 동안 고객 관리형 키를 암호화에 사용할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 AWS KMS에 저장된 KMS 키를 참조하세요.

> **Note**
>
> Amazon S3에서 데이터를 가져오는 작업은 Aurora Serverless v1에서 지원되지 않습니다. Aurora Serverless v2에서는 지원됩니다.

주제

- aws_s3 확장 설치
- Amazon S3 데이터에서 데이터 가져오기 개요
- Amazon S3 버킷에 대한 액세스 권한 설정

- [Amazon S3에서 Aurora PostgreSQL DB 클러스터로 데이터 가져오기]
- [함수 참조]

aws_s3 확장 설치

Amazon S3를 Aurora PostgreSQL DB 클러스터 와 함께 사용하려면 먼저 `aws_s3` 확장을 설치해야 합니다. 이 확장은 Amazon S3에서 데이터를 가져오기 위한 함수도 제공합니다. 또한 Aurora PostgreSQL DB 클러스터의 인스턴스에서 Amazon S3 버킷으로 데이터를 내보내는 기능도 제공합니다. 자세한 내용은 [Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기] 단원을 참조하세요. `aws_s3` 확장은 필요할 때 자동으로 설치되는 `aws_commons` 확장의 일부 도우미 기능에 따라 다릅니다.

`aws_s3` 확장을 설치하려면

1. psql(또는 PGAdmin)을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 `rds_superuser` 권한을 가진 사용자로 연결합니다. 설정 과정에서 기본 이름을 계속 사용했다면 `postgres`로 연결합니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 다음 명령을 실행하여 확장을 생성합니다.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 확장 프로그램이 설치되었는지 확인하려면 psql `\dx` 메타 명령을 사용하면 됩니다.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

이제 Amazon S3에서 데이터를 가져오고 Amazon S3로 데이터를 내보내는 기능을 사용할 수 있습니다.

Amazon S3 데이터에서 데이터 가져오기 개요

Aurora PostgreSQL로 S3 데이터 가져오기

먼저 함수에 제공해야 하는 세부 정보를 수집합니다. 여기에는 Aurora PostgreSQL DB 클러스터 인스턴스의 테이블 이름과 버킷 이름, 파일 경로, 파일 유형 및 Amazon S3 데이터가 저장된 AWS 리전이 포함됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 보기를 참조하세요.

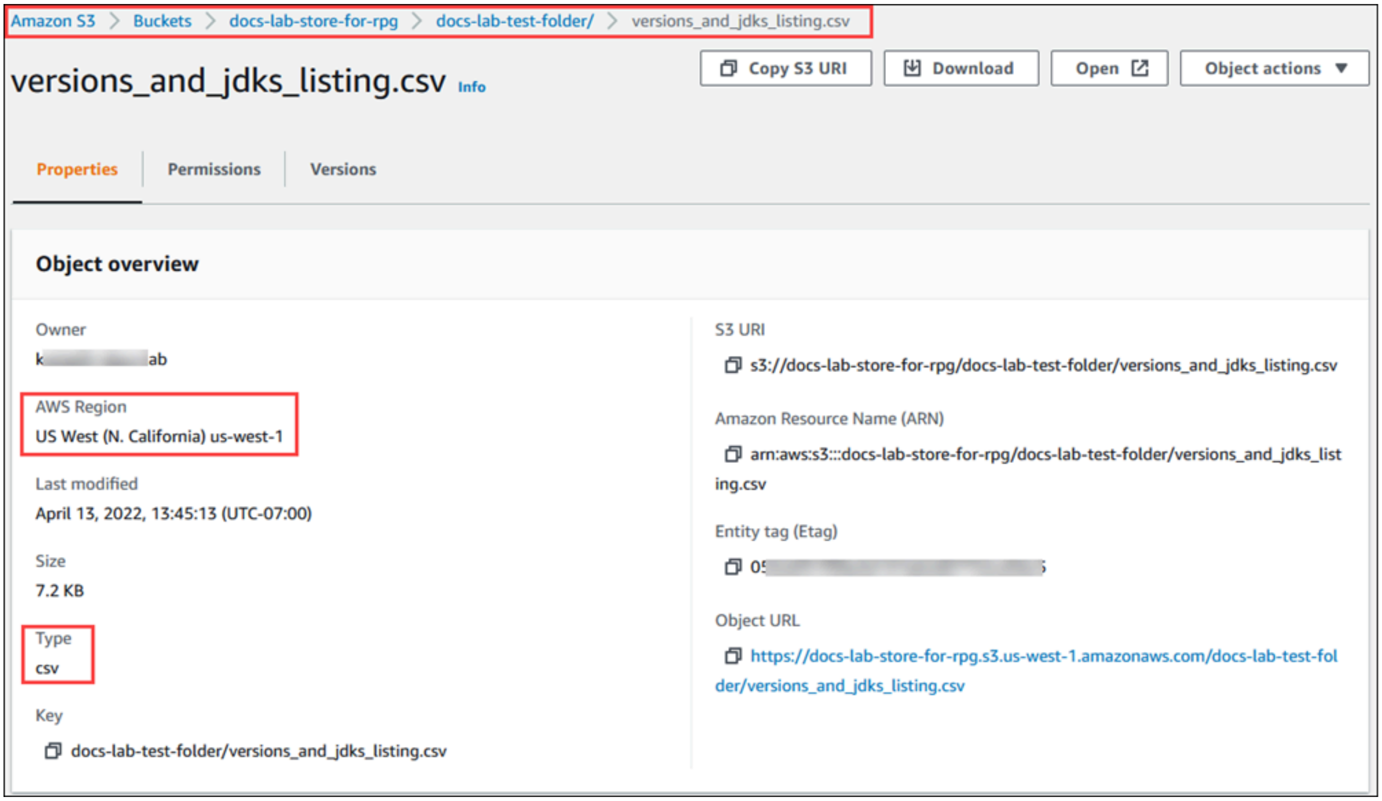
> **Note**
>
> Amazon S3에서 멀티 파트 데이터 가져오기는 현재 지원되지 않습니다.

1. `aws_s3.table_import_from_s3` 함수가 데이터를 가져올 테이블의 이름을 가져옵니다. 예를 들어 다음 명령은 이후 단계에서 사용할 수 있는 테이블 `t1`을 만듭니다.

```
postgres=> CREATE TABLE t1
    (col1 varchar(80),
    col2 varchar(80),
    col3 varchar(80));
```

2. Amazon S3 버킷 및 가져올 데이터에 대한 세부 정보를 가져옵니다. 이 작업을 수행하려면 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 열고 버킷을 선택합니다. 목록에서 데이터가 포함된 버킷을 찾습니다. 버킷을 선택하고 객체 개요 페이지를 연 다음 속성을 선택합니다.

   버킷 이름, 경로, AWS 리전, 파일 형식을 적어 둡니다. IAM 역할을 통해 Amazon S3에 대한 액세스를 설정하는 Amazon 리소스 이름(ARN) 이 나중에 필요합니다. 자세한 내용은 Amazon S3 버킷에 대한 액세스 권한 설정 단원을 참조하세요. 다음 그림에 예가 나와 있습니다.



3. AWS CLI 명령 `aws s3 cp`를 사용하여 Amazon S3 버킷의 데이터 경로를 확인할 수 있습니다. 정보가 정확하면 이 명령이 Amazon S3 파일의 복사본을 다운로드합니다.

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. Amazon S3 버킷의 파일에 대한 액세스를 허용하도록 Aurora PostgreSQL DB 클러스터에 대한 권한을 설정합니다. 이렇게 하려면 AWS Identity and Access Management(IAM) 역할 또는 보안 자격 증명을 사용합니다. 자세한 내용은 Amazon S3 버킷에 대한 액세스 권한 설정 단원을 참조하세요.

5. 경로 및 수집된 기타 Amazon S3 객체 세부 정보(2단계 참조)를 `create_s3_uri` 함수에 제공하여 Amazon S3 URI 객체를 생성합니다. 이 함수에 대한 자세한 내용은 aws_commons.create_s3_uri 단원을 참조하세요. 다음은 psql 세션 중에 이 객체를 구성하는 예입니다.

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

다음 단계에서는 이 객체(`aws_commons._s3_uri_1`)를 `aws_s3.table_import_from_s3` 함수에 전달하여 데이터를 테이블로 가져옵니다.

6. `aws_s3.table_import_from_s3` 함수를 호출하여 Amazon S3에서 테이블로 데이터를 가져옵니다. 참조 정보는 aws_s3.table_import_from_s3 단원을 참조하세요. 예시는 Amazon S3에서 Aurora PostgreSQL DB 클러스터로 데이터 가져오기 섹션을 참조하세요.

Amazon S3 버킷에 대한 액세스 권한 설정

Amazon S3 파일에서 데이터를 가져오려면 Aurora PostgreSQL DB 클러스터에 파일이 저장된 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 다음 항목에서 설명하는 두 방법 중 하나를 사용하여 Amazon S3 버킷에 대한 액세스 권한을 부여합니다.

주제

- IAM 역할을 사용해 Amazon S3 버킷에 액세스
- 보안 자격 증명을 사용해 Amazon S3 버킷에 액세스
- Amazon S3 액세스 문제 해결

IAM 역할을 사용해 Amazon S3 버킷에 액세스

Amazon S3 파일에서 데이터를 로드하기 전에 Aurora PostgreSQL DB 클러스터DB 클러스터에 파일이 저장된 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 그러면 추가 자격 증명 정보를 관리하거나 aws_s3.table_import_from_s3 함수 호출에서 제공할 필요가 없습니다.

이렇게 하려면 Amazon S3 버킷에 대한 액세스 권한을 부여하는 IAM 정책을 생성합니다. IAM 역할을 생성하여 정책을 역할에 연결합니다. 그런 다음 IAM 역할을 DB 클러스터에 할당합니다.

> **Note**
>
> IAM 역할을 Aurora Serverless v1 DB 클러스터와 연결할 수 없으므로 다음 단계에 적용되지 않습니다.

IAM 역할을 통해 Aurora PostgreSQL DB 클러스터에 Amazon S3 액세스 권한 부여

1. IAM 정책을 생성합니다.

이 정책은 Aurora PostgreSQL DB 클러스터 가 Amazon S3에 액세스할 수 있도록 허용하는 버킷 및 객체 권한을 부여합니다.

정책에 다음과 같은 필수 작업을 포함하여 Amazon S3 버킷에서 Aurora PostgreSQL로의 파일 전송을 허용합니다.

- `s3:GetObject`
- `s3:ListBucket`

정책에 다음 리소스를 포함하여 Amazon S3 버킷 및 그 안의 객체를 식별합니다. 다음은 Amazon S3에 액세스하기 위한 Amazon 리소스 이름(ARN) 형식입니다.

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

Aurora PostgreSQL용 IAM 정책 생성에 대한 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 단원을 참조하세요. IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결도 참조하세요.

다음 AWS CLI 명령은 이 옵션으로 `rds-s3-import-policy`라는 IAM 정책을 만듭니다. *amzn-s3-demo-bucket*이라는 버킷에 대한 액세스 권한을 부여합니다.

> **Note**
>
> 이 명령에서 반환되는 정책 Amazon 리소스 이름(ARN)을 적어 둡니다. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
```

```
        "Sid": "s3import",
        "Action": [
          "s3:GetObject",
          "s3:ListBucket"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
  }'
```

Windows의 경우:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

2. IAM 역할을 생성합니다.

   이렇게 하면, Aurora PostgreSQL에서 IAM 역할을 수임하여 사용자 대신 Amazon S3 버킷에 액세스할 수 있도록 역할을 생성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자에게 권한을 위임하기 위한 역할 생성을 참조하세요.

서비스 권한을 특정 리소스로 제한하는 리소스 기반 정책의 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하는 것이 좋습니다. 이는 혼동된 대리자 문제를 방지하는 가장 효과적인 방법입니다.

두 전역 조건 컨텍스트 키와 계정 ID를 포함한 aws:SourceArn 값을 모두 사용하는 경우, aws:SourceAccount 값 및 aws:SourceArn 값의 계정은 동일한 정책 문에서 사용될 경우 반드시 같은 계정 ID를 사용해야 합니다.

- 단일 리소스에 대한 교차 서비스 액세스를 원하는 경우 aws:SourceArn을 사용하세요.
- 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 aws:SourceAccount를 사용하세요.

정책에서는 리소스의 전체 ARN이 포함된 aws:SourceArn 전역 조건 컨텍스트 키를 사용해야 합니다. 다음 예제에서는 AWS CLI 명령을 사용하여 rds-s3-import-role이라는 역할을 생성하는 방법을 보여줍니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
```

```
    }'
```

Windows의 경우:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

3. 생성한 IAM 역할에 생성한 IAM 정책을 연결합니다.

   다음 AWS CLI 명령은 앞서 생성한 정책을 `rds-s3-import-role`이라는 역할에 연결합니다. *`your-policy-arn`*을 이전 단계에서 기록한 정책 ARN으로 바꿉니다.

   Example

   대상 LinuxmacOS, 또는Unix:

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-import-role
```

   Windows의 경우:

```
aws iam attach-role-policy ^
```

```
    --policy-arn your-policy-arn ^
    --role-name rds-s3-import-role
```

4. IAM 역할을 DB 클러스터에 추가합니다.

   이렇게 하려면 다음에 설명한 대로 AWS Management Console 또는 AWS CLI를 사용합니다.

콘솔

콘솔을 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 세부 정보를 표시하고자 하는 PostgreSQL DB 클러스터 이름을 선택합니다.
3. 연결성 및 보안(Connectivity & security) 탭에 있는 IAM 역할 관리(Manage IAM roles) 섹션의 이 클러스터에 IAM 역할 추가(Add IAM roles to this cluster/instance)에서 추가할 역할을 선택합니다.
4. 기능에서 s3Import를 선택합니다.
5. [Add role]을 선택합니다.

AWS CLI

CLI를 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

- 다음 명령을 사용해 `my-db-cluster`라는 PostgreSQL DB 클러스터에 역할을 추가합니다. *`your-role-arn`*을 이전 단계에서 기록한 정책 ARN으로 교체합니다. `s3Import` 옵션의 값에 대해 `--feature-name`를 사용합니다.

  Example

  대상 LinuxmacOS, 또는Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

  Windows의 경우:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

RDS API

Amazon RDS API를 사용하여 PostgreSQL DB 클러스터 에 대한 IAM 역할을 추가하려면 AddRoleToDBCluster 작업을 호출합니다.

보안 자격 증명을 사용해 Amazon S3 버킷에 액세스

원할 경우 IAM 역할을 사용하는 대신 보안 자격 증명을 사용해 Amazon S3 버킷에 대한 액세스 권한을 부여여할 수 있습니다. aws_s3.table_import_from_s3 함수 호출에서 `credentials` 파라미터를 지정하면 됩니다.

`credentials` 파라미터는 `aws_commons._aws_credentials_1` 자격 증명을 포함하는 AWS 유형의 구조입니다. 다음과 같이 aws_commons.create_aws_credentials 함수를 사용해 `aws_commons._aws_credentials_1` 구조에서 액세스 키와 비밀 키를 설정하십시오.

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

다음과 같이 `aws_commons._aws_credentials_1` 구조를 생성한 후 aws_s3.table_import_from_s3 함수를 `credentials` 파라미터와 함께 사용해 데이터를 가져옵니다.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

또는 aws_commons.create_aws_credentials 함수 호출 인라인을 `aws_s3.table_import_from_s3` 함수 호출에 포함할 수 있습니다.

```
postgres=> SELECT aws_s3.table_import_from_s3(
```

```
    't', '', '(format csv)',
    :'s3_uri',
    aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Amazon S3 액세스 문제 해결

Amazon S3 파일 데이터를 가져오려 할 때 연결 문제가 발생할 경우 다음 권장 사항을 참조하세요.

- Amazon Aurora 자격 증명 및 액세스 문제 해결
- Amazon Simple Storage Service 사용 설명서의 Amazon S3 문제 해결
- IAM 사용 설명서의 Amazon S3 문제 해결 및 IAM

Amazon S3에서 Aurora PostgreSQL DB 클러스터로 데이터 가져오기

aws_s3 확장의 `table_import_from_s3` 함수를 사용하여 Amazon S3 버킷에서 데이터를 가져옵니다. 참조 정보는 aws_s3.table_import_from_s3 단원을 참조하세요.

> **Note**
>
> 다음 예에서는 Amazon S3 버킷에 대한 액세스 권한을 허용하기 위해 IAM 역할 메서드를 사용합니다. 따라서 `aws_s3.table_import_from_s3` 함수 호출에는 자격 증명 파라미터가 포함되지 않습니다.

다음은 대표적인 예를 보여줍니다.

```
postgres=> SELECT aws_s3.table_import_from_s3(
    't1',
    '',
    '(format csv)',
    :'s3_uri'
);
```

파라미터는 다음과 같습니다.

- `t1` – 데이터를 복사할 PostgreSQL DB 클러스터의 테이블에 지정된 이름입니다.
- `''` – 데이터베이스 테이블의 열 목록입니다(선택 사항). 이 파라미터를 사용해 S3 데이터 중 어떤 열이 어떤 테이블 열에 들어가는지 표시할 수 있습니다. 열을 지정하지 않으면 모든 열이 테이블에 복

사됩니다. 열 목록 사용에 대한 예시는 사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기 단원을 참조하십시오.

- `(format csv)` – PostgreSQL COPY 인수입니다. 복사 프로세스에서는 PostgreSQL COPY 명령의 인수와 형식을 사용하여 데이터를 가져옵니다. 형식에 대한 선택 사항에는 이 예에 표시된 대로 쉼표로 구분된 값(CSV), 텍스트 및 이진법이 있습니다. 기본값은 텍스트입니다.
- `s3_uri` – Amazon S3 파일을 식별하는 정보가 포함된 구조입니다. aws_commons.create_s3_uri 함수를 사용하여 `s3_uri` 구조를 생성하는 예제는 Amazon S3 데이터에서 데이터 가져오기 개요 단원을 참조하세요.

이 함수에 대한 자세한 내용은 aws_s3.table_import_from_s3 단원을 참조하세요.

이 `aws_s3.table_import_from_s3` 함수는 문자를 반환합니다. Amazon S3 버킷에서 가져올 다른 파일 종류를 지정하려면 다음 예 중 하나를 참조하세요.

> **Note**
>
> 0바이트 파일을 가져오면 오류가 발생합니다.

주제

- 사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기
- Amazon S3 압축(gzip) 파일 가져오기
- 인코딩된 Amazon S3 파일 가져오기

사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기

다음 예제에서는 사용자 지정 구분 기호를 사용하는 파일을 가져오는 방법을 보여줍니다. 또한 `column_list` 함수의 aws_s3.table_import_from_s3 파라미터를 사용해 데이터베이스 테이블에서 데이터를 배치할 곳을 제어하는 방법을 보여줍니다.

이 예에서는 다음 정보가 Amazon S3 파일의 파이프로 구분된 열에 정리되어 있다고 가정합니다.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

사용자 지정 구분 기호를 사용하는 파일을 가져오려면

1. 가져온 데이터에 대해 데이터베이스에서 테이블을 생성합니다.

```
postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
```

2. aws_s3.table_import_from_s3 함수의 다음과 같은 형식을 사용해 Amazon S3 파일에서 데이터를 가져옵니다.

   aws_commons.create_s3_uri 함수 호출 인라인을 `aws_s3.table_import_from_s3` 함수 호출에 포함하여 파일을 지정할 수 있습니다.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   'test',
   'a,b,d,e',
   'DELIMITER ''|''',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
east-2')
);
```

이제 데이터는 다음 열의 테이블에 있습니다.

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Amazon S3 압축(gzip) 파일 가져오기

다음 예에서는 gzip으로 압축된 Amazon S3에서 파일을 가져오는 방법을 보여줍니다. 가져오는 파일에 다음과 같은 Amazon S3 메타데이터가 있어야 합니다.

- 키: `Content-Encoding`
- 값: `gzip`

AWS Management Console을 사용하여 파일을 업로드하는 경우 일반적으로 시스템에서 메타데이터를 적용합니다. AWS Management Console, AWS CLI 또는 API를 사용하여 Amazon S3에 파일을 업

로드하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 업로드를 참조하세요.

Amazon S3 메타데이터에 대한 자세한 내용과 시스템 제공 메타데이터에 대한 세부 정보는 Amazon Simple Storage Service 사용 설명서의 Amazon S3 콘솔에서 객체 메타데이터 편집을 참조하세요.

아래와 같이 gzip 파일을 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터로 가져옵니다.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

인코딩된 Amazon S3 파일 가져오기

다음 예에서는 Windows-1252 인코딩이 있는 Amazon S3에서 파일을 가져오는 방법을 보여줍니다.

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

함수 참조

Functions

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Amazon S3 데이터를 Aurora PostgreSQL 테이블로 가져옵니다. `aws_s3` 확장은 `aws_s3.table_import_from_s3` 함수를 제공합니다. 반환 값은 텍스트입니다.

구문

필수 파라미터는 `table_name`, `column_list` 및 `options`입니다. 이 파라미터에서는 데이터베이스 테이블을 식별하고 데이터가 테이블로 복사되는 방식을 지정합니다.

다음 파라미터를 사용할 수도 있습니다.

- `s3_info` 파라미터는 가져올 Amazon S3 파일을 지정합니다. 이 파라미터를 사용하는 경우 IAM 역할에서 PostgreSQL DB 클러스터에 대해 Amazon S3에 액세스할 수 있는 권한을 제공합니다.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- `credentials` 파라미터에서는 Amazon S3에 액세스할 수 있는 자격 증명을 지정합니다. 이 파라미터를 사용할 때는 IAM 역할을 사용하지 마십시오.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

파라미터

table_name

데이터를 가져올 필수 텍스트 문자열로서, PostgreSQL 데이터베이스 테이블의 이름을 포함합니다.

column_list

데이터를 복사할 PostgreSQL 데이터베이스 테이블 열의 목록(선택 사항)을 포함하는 필수 텍스트 문자열입니다. 문자열이 비어 있는 경우 테이블의 모든 열이 사용됩니다. 관련 예시는 사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기 단원을 참조하십시오.

options

PostgreSQL `COPY` 명령에 대한 인수를 포함하는 필수 텍스트 스트링입니다. 이 인수에서는 데이터가 PostgreSQL 테이블에 복사되는 방식을 지정합니다. 자세한 내용은 PostgreSQL COPY 설명서를 참조하십시오.

s3_info

S3 객체에 대한 다음 정보를 포함하는 `aws_commons._s3_uri_1` 복합 키입니다.

- bucket – 파일이 포함된 Amazon S3 버킷의 이름입니다.
- file_path – 파일 경로를 포함한 Amazon S3 파일 이름입니다.
- region - 파일이 위치한 AWS 리전입니다. AWS 리전 이름 및 연결된 값의 목록은 리전 및 가용 영역 섹션을 참조하십시오.

credentials

가져오기 작업에 사용할 다음 자격 증명을 포함하는 aws_commons._aws_credentials_1 복합 유형입니다.

- 액세스 키
- 비밀 키
- 세션 토큰

aws_commons._aws_credentials_1 복합 구조 생성에 대한 자세한 내용은 aws_commons.create_aws_credentials 단원을 참조하십시오.

대체 구문

s3_info 및 credentials 파라미터 대신에 확장 파라미터 집합을 사용하면 테스트에 도움이 됩니다. 다음은 aws_s3.table_import_from_s3 함수에 대한 추가 구문 변형입니다.

- s3_info 파라미터를 사용해 Amazon S3 파일을 식별하는 대신 bucket, file_path 및 region 파라미터의 조합을 사용하십시오. IAM 역할은 이러한 형식의 함수를 통해 PostgreSQL DB 인스턴스에 대해 Amazon S3에 액세스할 수 있는 권한을 제공합니다.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- credentials 파라미터를 사용해 Amazon S3 액세스를 지정하는 대신 access_key, session_key 및 session_token 파라미터의 조합을 사용하십시오.

```
aws_s3.table_import_from_s3 (
   table_name text,
```

```
    column_list text,
    options text,
    bucket text,
    file_path text,
    region text,
    access_key text,
    secret_key text,
    session_token text
)
```

대체 파라미터

bucket

파일이 들어 있는 Amazon S3 버킷의 이름이 포함된 텍스트 문자열입니다.

file_path

파일 경로를 포함한 Amazon S3 파일 이름이 포함된 텍스트 문자열입니다.

region

파일의 AWS 리전 위치를 식별하는 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 리전 및 가용 영역 섹션을 참조하세요.

access_key

가져오기 작업에 사용할 액세스 키가 포함된 텍스트 문자열입니다. 기본값은 NULL입니다.

secret_key

가져오기 작업에 사용할 비밀 키가 포함된 텍스트 문자열입니다. 기본값은 NULL입니다.

session_token

(선택 사항) 가져오기 작업에 사용할 세션 키가 포함된 텍스트 문자열입니다. 기본값은 NULL입니다.

aws_commons.create_s3_uri

Amazon S3 파일 정보를 저장할 `aws_commons._s3_uri_1` 구조를 생성합니다. `aws_commons.create_s3_uri` 함수의 `s3_info` 파라미터에서 aws_s3.table_import_from_s3 함수의 결과를 사용합니다.

구문

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

파라미터

bucket

파일의 Amazon S3 버킷 이름이 포함된 필수 텍스트 문자열입니다.

file_path

파일 경로를 포함한 Amazon S3 파일 이름이 포함된 필수 텍스트 문자열입니다.

region

파일이 위치한 AWS 리전이 포함된 필수 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 리전 및 가용 영역 섹션을 참조하세요.

aws_commons.create_aws_credentials

`aws_commons._aws_credentials_1` 구조에서 액세스 키와 비밀 키를 설정합니다. `aws_commons.create_aws_credentials` 함수의 `credentials` 파라미터에서 aws_s3.table_import_from_s3 함수의 결과를 사용합니다.

구문

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

파라미터

access_key

Amazon S3 파일 가져오기에 사용할 액세스 키가 포함된 필수 텍스트 문자열입니다. 기본값은 NULL입니다.

secret_key

Amazon S3 파일 가져오기에 사용할 비밀 키가 포함된 필수 텍스트 문자열입니다. 기본값은 NULL 입니다.

session_token

Amazon S3 파일 가져오기에 사용할 세션 토큰이 포함된 텍스트 문자열(선택 사항)입니다. 기본값은 NULL입니다. 선택 사항인 `session_token`을 제공하는 경우 임시 자격 증명을 사용할 수 있습니다.

Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기

Aurora PostgreSQL DB 클러스터에서 데이터를 쿼리하여 Amazon S3 버킷에 저장된 파일로 직접 내보낼 수 있습니다. 이 작업을 수행하려면 먼저 Aurora PostgreSQL `aws_s3` 확장을 설치해야 합니다. 이 확장은 쿼리 결과를 Amazon S3로 내보내는 데 사용하는 기능을 제공합니다. 다음에서는 확장 프로그램을 설치하는 방법과 Amazon S3로 데이터를 내보내는 방법을 확인할 수 있습니다.

프로비저닝된 인스턴스 또는 Aurora Serverless v2 DB 인스턴스에서 내보낼 수 있습니다. 이 단계는 Aurora Serverless v1에서 지원되지 않습니다.

> **Note**
>
> Amazon S3로 계정 간 내보내기는 지원되지 않습니다.

현재 사용 가능한 모든 Aurora PostgreSQL 버전은 Amazon Simple Storage Service로 데이터 내보내기를 지원합니다. 자세한 버전 정보는 Aurora PostgreSQL 릴리스 정보에서 [Amazon Aurora PostgreSQL 업데이트](#)를 참조하세요.

내보내기용으로 버킷을 설정하지 않은 경우 Amazon Simple Storage Service 사용 설명서의 다음 주제를 참조하세요.

- [Amazon S3 설정](#)
- [버킷 생성](#)

기본적으로 Aurora PostgreSQL에서 Amazon S3로 내보낸 데이터는 AWS 관리형 키를 통한 서버 측 암호화를 사용합니다. 이미 생성한 고객 관리형 키를 사용할 수도 있습니다. 버킷 암호화를 사용하는

경우 Amazon S3 버킷은 AWS Key Management Service(AWS KMS) 키(SSE-KMS)로 암호화되어야 합니다. 현재 Amazon S3 관리형 키(SSE-S3)로 암호화된 버킷은 지원되지 않습니다.

> **Note**
>
> AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 DB 및 DB 클러스터 스냅샷 데이터를 Amazon S3에 저장할 수 있습니다. 자세한 내용은 [Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기]() 단원을 참조하세요.

주제

- aws_s3 확장 설치
- Amazon S3으로 데이터 내보내기 개요
- 내보낼 Amazon S3 파일 경로 지정
- Amazon S3 버킷에 대한 액세스 권한 설정
- aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기
- 함수 참조
- Amazon S3 액세스 문제 해결

aws_s3 확장 설치

Amazon Simple Storage Service를 Aurora PostgreSQL DB 클러스터 와 함께 사용하려면 먼저 `aws_s3` 확장을 설치해야 합니다. 이 확장은 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에서 Amazon S3 버킷으로 데이터를 내보내는 기능을 제공합니다. Amazon S3에서 데이터를 가져오기 위한 함수도 제공합니다. 자세한 내용은 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터 단원을 참조하세요. `aws_s3` 확장은 필요할 때 자동으로 설치되는 `aws_commons` 확장의 일부 도우미 기능에 따라 다릅니다.

`aws_s3` 확장을 설치하려면

1. psql(또는 PGAdmin)을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 `rds_superuser` 권한을 가진 사용자로 연결합니다. 설정 과정에서 기본 이름을 계속 사용했다면 `postgres`로 연결합니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 다음 명령을 실행하여 확장을 생성합니다.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 확장 프로그램이 설치되었는지 확인하려면 psql \dx 메타 명령을 사용하면 됩니다.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

이제 Amazon S3에서 데이터를 가져오고 Amazon S3로 데이터를 내보내는 기능을 사용할 수 있습니다.

Aurora PostgreSQL 버전에서 Amazon S3로 내보내기를 지원하는지 확인

`describe-db-engine-versions` 명령을 사용하여 Aurora PostgreSQL 버전이 Amazon S3로 내보내기를 지원하는지 확인할 수 있습니다. 다음 예제에서는 버전 10.14를 Amazon S3로 내보낼 수 있는지 확인합니다.

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

출력에 `"s3Export"` 문자열이 포함된 경우 엔진은 Amazon S3 내보내기를 지원합니다. 그렇지 않으면 엔진이 이 내보내기를 지원하지 않습니다.

Amazon S3으로 데이터 내보내기 개요

Aurora PostgreSQL 데이터베이스에 저장된 데이터를 Amazon S3 버킷으로 내보내려면 다음 절차를 따르세요.

Aurora PostgreSQL 데이터를 S3로 내보내려면

1. 데이터를 내보내는 데 사용할 Amazon S3 파일 경로를 식별합니다. 이 프로세스에 대한 자세한 내용은 [내보낼 Amazon S3 파일 경로 지정](#) 단원을 참조하십시오.

2. Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다.

   Amazon S3 파일로 데이터를 내보내려면 Aurora PostgreSQL DB 클러스터에 내보내기 시 스토리지에 사용할 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 여기에는 다음 단계가 포함됩니다.

   1. 내보낼 Amazon S3 버킷에 대한 액세스를 제공하는 IAM 정책을 생성합니다.
   2. IAM 역할 생성.
   3. 생성한 정책을 생성한 역할에 연결하십시오.
   4. 이 IAM 역할을 DB 클러스터 에 추가합니다.

   이 프로세스에 대한 자세한 내용은 Amazon S3 버킷에 대한 액세스 권한 설정 단원을 참조하십시오.

3. 데이터를 가져올 데이터베이스 쿼리를 식별합니다. `aws_s3.query_export_to_s3` 함수를 호출하여 쿼리 데이터를 내보냅니다.

   앞의 준비 작업을 완료한 후 aws_s3.query_export_to_s3 함수를 사용하여 쿼리 결과를 Amazon S3으로 내보냅니다. 이 프로세스에 대한 자세한 내용은 aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기 단원을 참조하세요.

내보낼 Amazon S3 파일 경로 지정

다음 정보를 지정하여 데이터를 내보낼 Amazon S3 위치를 식별합니다.

- 버킷 이름 – 버킷은 Amazon S3 객체 또는 파일을 위한 컨테이너입니다.

  Amazon S3을 이용한 데이터 저장에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Creating a bucket 및 Working with objects를 참조하세요.

- 파일 경로 – 파일 경로는 내보낸 데이터가 Amazon S3 버킷에서 저장되는 위치를 식별합니다. 파일 경로는 다음과 같이 구성됩니다.
  - 가상 폴더 경로를 식별하는 선택적 경로 접두사입니다.
  - 저장할 하나 이상의 파일을 식별하는 파일 접두사입니다. 내보내는 데이터가 클 경우 각각 최대 크기가 약 6GB인 여러 파일에 저장됩니다. 추가 파일 이름의 파일 접두사도 동일하지만 `_part`*XX*가 추가됩니다. *XX*는 2, 3 등을 나타냅니다.

  예를 들어 `exports` 폴더와 `query-1-export` 파일 접두사가 있는 파일 경로는 `/exports/query-1-export`입니다.

- AWS 리전(선택 사항) – Amazon S3 버킷이 위치한 AWS 리전입니다. AWS 리전 값을 지정하지 않으면 Aurora는 DB 클러스터 내보내기와 동일한 AWS 리전의 Amazon S3에 파일을 저장합니다.

> **Note**
>
> 현재 AWS 리전은 내보내는 DB 클러스터의 리전과 동일해야 합니다.

AWS 리전 이름 및 연결된 값의 목록은 리전 및 가용 영역 섹션을 참조하십시오.

내보내는 데이터를 저장할 위치에 대한 Amazon S3 파일 정보를 보관하려면 aws_commons.create_s3_uri 함수를 사용하여 다음과 같이 `aws_commons._s3_uri_1` 복합 구조를 생성할 수 있습니다.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

나중에 이 `s3_uri_1` 값을 aws_s3.query_export_to_s3 함수에 대한 호출의 파라미터로 제공합니다. 예제는 aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기을 참조하세요.

Amazon S3 버킷에 대한 액세스 권한 설정

데이터를 Amazon S3으로 내보내려면 PostgreSQL DB 클러스터에 파일이 저장될 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다.

이렇게 하려면 다음 절차를 따르십시오.

IAM 역할을 통해 PostgreSQL DB 클러스터에 액세스할 수 있는 권한을 Amazon S3에 부여하려면

1. IAM 정책을 생성합니다.

   이 정책은 PostgreSQL DB 클러스터가 Amazon S3에 액세스할 수 있도록 허용하는 권한을 버킷 및 객체에 부여합니다.

   이 정책을 생성하는 과정에서 다음 단계를 수행하십시오.

a. 다음과 같은 필수 작업을 정책에 포함하여 PostgreSQL DB 클러스터에서 Amazon S3 버킷으로 파일 전송을 허용합니다.

   - `s3:PutObject`
   - `s3:AbortMultipartUpload`

b. Amazon S3 버킷과 버킷의 객체를 식별하는 Amazon 리소스 이름(ARN)을 포함합니다. Amazon S3에 액세스하기 위한 ARN 형식은 `arn:aws:s3:::amzn-s3-demo-bucket/*`입니다.

PostgreSQL용 Aurora PostgreSQL에 대한 IAM 정책 생성에 대한 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 단원을 참조하십시오. IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결도 참조하십시오.

다음 AWS CLI 명령은 이 옵션으로 `rds-s3-export-policy`라는 IAM 정책을 만듭니다. *`amzn-s3-demo-bucket`*이라는 버킷에 대한 액세스 권한을 부여합니다.

> **Warning**
>
> 특정 버킷에 액세스하도록 구성된 엔드포인트 정책이 있는 프라이빗 VPC 내에 데이터베이스를 설정하는 것이 좋습니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon S3용 엔드포인트 정책 사용을 참조하십시오.
> 모든 리소스에 액세스할 수 있는 정책은 생성하지 않는 것이 좋습니다. 이러한 액세스 권한은 데이터 보안에 위협이 될 수 있습니다. `S3:PutObject`를 사용하여 모든 리소스에 액세스할 수 있는 권한을 `"Resource":"*"`에 부여하는 정책을 생성하면 내보내기 권한이 있는 사용자가 계정의 모든 버킷으로 데이터를 내보낼 수 있습니다. 또한 사용자는 AWS 리전 내의 공개적으로 쓰기 가능한 버킷으로 데이터를 내보낼 수 있습니다.

정책을 만든 후에 정책의 Amazon 리소스 이름(ARN)을 기록하십시오. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
```

```
            "s3:PutObject*",
            "s3:ListBucket",
            "s3:GetObject*",
            "s3:DeleteObject*",
            "s3:GetBucketLocation",
            "s3:AbortMultipartUpload"
          ],
          "Effect": "Allow",
          "Resource": [
            "arn:aws:s3:::amzn-s3-demo-bucket/*"
          ]
        }
      ]
    }'
```

2. IAM 역할 생성.

   Aurora PostgreSQL이 이 IAM 역할을 수임하여 사용자 대신 Amazon S3 버킷에 액세스할 수 있도록 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자에게 권한을 위임하기 위한 역할 생성을 참조하세요.

   서비스 권한을 특정 리소스로 제한하는 리소스 기반 정책의 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하는 것이 좋습니다. 이는 혼동된 대리자 문제를 방지하는 가장 효과적인 방법입니다.

   두 전역 조건 컨텍스트 키와 계정 ID를 포함한 `aws:SourceArn` 값을 모두 사용하는 경우, `aws:SourceAccount` 값 및 `aws:SourceArn` 값의 계정은 동일한 정책 문에서 사용될 경우 반드시 같은 계정 ID를 사용해야 합니다.

   - 단일 리소스에 대한 교차 서비스 액세스를 원하는 경우 `aws:SourceArn`을 사용하세요.
   - 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 `aws:SourceAccount`를 사용하세요.

   정책에서는 리소스의 전체 ARN이 포함된 `aws:SourceArn` 전역 조건 컨텍스트 키를 사용해야 합니다. 다음 예제에서는 AWS CLI 명령을 사용하여 `rds-s3-export-role`이라는 역할을 생성하는 방법을 보여줍니다.

   Example

   대상 LinuxmacOS, 또는Unix:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
       }
     ]
   }'
```

Windows의 경우:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
```

```
        }
      ]
    }'
```

3. 생성한 IAM 역할에 생성한 IAM 정책을 연결합니다.

   다음 AWS CLI 명령은 앞서 생성한 정책을 `rds-s3-export-role.`라는 역할에 연결합니다. *`your-policy-arn`*을 이전 단계에서 기록한 정책 ARN으로 바꿉니다.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

4. IAM 역할을 DB 클러스터에 추가합니다. 이렇게 하려면 다음에 설명한 대로 AWS Management Console 또는 AWS CLI를 사용합니다.

콘솔

콘솔을 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 세부 정보를 표시하고자 하는 PostgreSQL DB 클러스터 이름을 선택합니다.
3. Connectivity & security(연결성 및 보안) 탭에 있는 Manage IAM roles(IAM 역할 관리) 섹션의 이 인스턴스에 IAM 역할 추가에서 추가할 역할을 선택합니다.
4. 기능에서 s3Export를 선택합니다.
5. [Add role]을 선택합니다.

AWS CLI

CLI를 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

• 다음 명령을 사용해 `my-db-cluster`라는 PostgreSQL DB 클러스터에 역할을 추가합니다. *`your-role-arn`*을 이전 단계에서 기록한 정책 ARN으로 교체합니다. `s3Export` 옵션의 값에 대해 `--feature-name`를 사용합니다.

  Example

  대상 LinuxmacOS, 또는Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

Windows의 경우:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
   --region your-region
```

aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기

aws_s3.query_export_to_s3 함수를 호출하여 PostgreSQL 데이터를 Amazon S3으로 내보냅니다.

주제

- 사전 조건
- aws_s3.query_export_to_s3 호출
- 사용자 지정 구분 기호를 사용하는 CSV 파일로 내보내기
- 인코딩을 사용하여 이진 파일로 내보내기

사전 조건

`aws_s3.query_export_to_s3` 함수를 사용하기 전에 다음 사전 조건을 충족해야 합니다.

- Amazon S3으로 데이터 내보내기 개요에 설명된 대로 필요한 PostgreSQL 확장을 설치합니다.
- 내보낼 Amazon S3 파일 경로 지정에 설명된 대로 데이터를 내보낼 Amazon S3 위치를 결정합니다.
- Amazon S3 버킷에 대한 액세스 권한 설정에 설명된 대로 Amazon S3에 대한 내보내기 액세스 권한이 DB 클러스터 에 있는지 확인합니다.

다음 예제에서는 `sample_table`이라는 데이터베이스 테이블을 사용합니다. 이 예제에서는 *`amzn-s3-demo-bucket`*이라는 버킷으로 데이터를 내보냅니다. 예제 테이블과 데이터는 psql에서 다음 SQL 문을 사용하여 생성됩니다.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

aws_s3.query_export_to_s3 호출

다음은 aws_s3.query_export_to_s3 함수를 호출하는 기본 방법을 보여줍니다.

이 예제에서는 s3_uri_1 변수를 사용하여 Amazon S3 파일을 식별하는 정보가 포함된 구조를 식별합니다. aws_commons.create_s3_uri 함수를 사용하여 구조를 생성합니다.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

파라미터가 다음 두 aws_s3.query_export_to_s3 함수 호출에 따라 달라도 이러한 예제에 대한 결과는 동일합니다. sample_table 표의 모든 행은 *amzn-s3-demo-bucket*이라는 버킷으로 내보내집니다.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

파라미터는 다음과 같이 설명됩니다.

- 'SELECT * FROM sample_table' – 첫 번째 파라미터는 SQL 쿼리를 포함하는 필수 텍스트 문자열입니다. PostgreSQL 엔진은 이 쿼리를 실행합니다. 쿼리 결과는 다른 파라미터에서 식별된 S3 버킷에 복사됩니다.
- :'s3_uri_1' – 이 파라미터는 Amazon S3 파일을 식별하는 구조입니다. 이 예제에서는 변수를 사용하여 이전에 생성된 구조를 식별합니다. 대신 다음과 같이 aws_commons.create_s3_uri 함수 호출 내에 aws_s3.query_export_to_s3 함수 호출을 인라인을 포함시켜 구조를 생성할 수 있습니다.

```
SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
```

```
);
```

• `options :='format text'` – `options` 파라미터는 PostgreSQL COPY 인수를 포함하는 선택적 텍스트 문자열입니다. 복사 프로세스에서는 PostgreSQL COPY 명령의 인수 및 형식을 사용합니다.

지정된 파일이 Amazon S3 버킷에 없으면 생성됩니다. 파일이 이미 있는 경우 파일을 덮어씁니다. Amazon S3에서 내보낸 데이터에 액세스하는 구문은 다음과 같습니다.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

내보내는 데이터가 클 경우 각각 최대 크기가 약 6GB인 여러 파일에 저장됩니다. 추가 파일 이름의 파일 접두사도 동일하지만 `_partXX`가 추가됩니다. *XX*는 2, 3 등을 나타냅니다. 예를 들어 데이터 파일을 저장하는 경로를 다음과 같이 지정한다고 가정합니다.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

내보내기 시 세 개의 데이터 파일을 만들어야 하는 경우 Amazon S3 버킷에 다음 데이터 파일이 포함됩니다.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

이 함수에 대한 전체 참조 및 이 함수를 호출하는 추가 방법은 aws_s3.query_export_to_s3 단원을 참조하세요. Amazon S3에서 파일에 액세스하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 보기를 참조하세요.

사용자 지정 구분 기호를 사용하는 CSV 파일로 내보내기

다음 예제에서는 aws_s3.query_export_to_s3 함수를 호출하여 사용자 지정 구분 기호를 사용하는 파일로 데이터를 내보내는 방법을 보여줍니다. 이 예제에서는 PostgreSQL COPY 명령의 인수를 사용하여 쉼표로 구분된 값(CSV) 형식과 콜론(:) 구분 기호를 지정합니다.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

인코딩을 사용하여 이진 파일로 내보내기

다음 예제에서는 aws_s3.query_export_to_s3 함수를 호출하여 Windows-1253 인코딩이 있는 이진 파일로 데이터를 내보내는 방법을 보여줍니다.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

함수 참조

함수

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

PostgreSQL 쿼리 결과를 Amazon S3 버킷으로 내보냅니다. `aws_s3` 확장은 `aws_s3.query_export_to_s3` 함수를 제공합니다.

두 가지 필수 파라미터는 `query` 및 `s3_info`입니다. 이러한 파라미터는 내보낼 쿼리를 정의하고 내보낼 Amazon S3 버킷을 식별합니다. 다양한 내보내기 파라미터를 정의하기 위해 `options`라는 선택적 파라미터가 제공됩니다. `aws_s3.query_export_to_s3` 함수 사용 예는 aws_s3.query_export_to_s3 함수를 사용하여 쿼리 데이터 내보내기 단원을 참조하십시오.

구문

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

입력 파라미터

query

PostgreSQL 엔진이 실행하는 SQL 쿼리를 포함하는 필수 텍스트 문자열입니다. 이 쿼리의 결과는 `s3_info` 파라미터에서 식별된 S3 버킷에 복사됩니다.

s3_info

S3 객체에 대한 다음 정보를 포함하는 `aws_commons._s3_uri_1` 복합 키입니다.

- `bucket` – 파일을 포함할 Amazon S3 버킷의 이름입니다.

- `file_path` – Amazon S3 파일 이름 및 경로입니다.
- `region` - 버킷이 있는 AWS 리전입니다. AWS 리전 이름 및 연결된 값의 목록은 리전 및 가용 영역 섹션을 참조하십시오.

  현재, 이 값은 내보내는 DB 클러스터 의 AWS 리전과 동일해야 합니다. 기본값은 내보내는 DB 클러스터 의 AWS 리전입니다.

`aws_commons._s3_uri_1` 복합 구조를 생성하려면 aws_commons.create_s3_uri 함수를 참조하십시오.

options

PostgreSQL COPY 명령에 대한 인수를 포함하는 선택적 텍스트 문자열입니다. 이러한 인수는 내보낼 때 데이터를 복사하는 방법을 지정합니다. 자세한 내용은 PostgreSQL COPY 설명서를 참조하십시오.

kms_key text

데이터를 내보낼 S3 버킷의 고객 관리형 KMS 키가 포함된 선택적 텍스트 문자열입니다.

대체 입력 파라미터

`s3_info` 파라미터 대신에 확장 파라미터 세트를 사용하면 테스트에 도움이 됩니다. 다음은 `aws_s3.query_export_to_s3` 함수에 대한 추가 구문 변형입니다.

`s3_info` 파라미터를 사용해 Amazon S3 파일을 식별하는 대신 `bucket`, `file_path` 및 `region` 파라미터의 조합을 사용하십시오.

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

PostgreSQL 엔진이 실행하는 SQL 쿼리를 포함하는 필수 텍스트 문자열입니다. 이 쿼리의 결과는 `s3_info` 파라미터에서 식별된 S3 버킷에 복사됩니다.

bucket

파일이 들어 있는 Amazon S3 버킷의 이름이 포함된 필수 텍스트 문자열입니다.

file_path

파일 경로를 포함한 Amazon S3 파일 이름이 포함된 필수 텍스트 문자열입니다.

region

버킷이 있는 AWS 리전을 포함하는 선택적 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역](#) 섹션을 참조하십시오.

현재, 이 값은 내보내는 DB 클러스터 의 AWS 리전과 동일해야 합니다. 기본값은 내보내는 DB 클러스터 의 AWS 리전입니다.

options

PostgreSQL COPY 명령에 대한 인수를 포함하는 선택적 텍스트 문자열입니다. 이러한 인수는 내보낼 때 데이터를 복사하는 방법을 지정합니다. 자세한 내용은 [PostgreSQL COPY 설명서](#)를 참조하십시오.

kms_key text

데이터를 내보낼 S3 버킷의 고객 관리형 KMS 키가 포함된 선택적 텍스트 문자열입니다.

출력 파라미터

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

지정된 쿼리에 대해 Amazon S3에 성공적으로 업로드된 테이블 행 수입니다.

files_uploaded

Amazon S3에 업로드된 파일 수입니다. 파일은 약 6GB 크기로 생성됩니다. 생성된 각 추가 파일 이름에 `_partXX`가 추가됩니다. *XX*는 2, 3 등을 나타냅니다.

bytes_uploaded

Amazon S3에 업로드된 총 바이트 수입니다.

예시

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Amazon S3 파일 정보를 저장할 `aws_commons._s3_uri_1` 구조를 생성합니다. `aws_commons.create_s3_uri` 함수의 `s3_info` 파라미터에서 [aws_s3.query_export_to_s3](#) 함수의 결과를 사용합니다. `aws_commons.create_s3_uri` 함수 사용 예는 [내보낼 Amazon S3 파일 경로 지정](#) 단원을 참조하십시오.

구문

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

입력 파라미터

bucket

파일의 Amazon S3 버킷 이름이 포함된 필수 텍스트 문자열입니다.

file_path

파일 경로를 포함한 Amazon S3 파일 이름이 포함된 필수 텍스트 문자열입니다.

region

파일이 위치한 AWS 리전이 포함된 필수 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역](#) 섹션을 참조하십시오.

Amazon S3 액세스 문제 해결

데이터를 Amazon S3로 내보내려고 할 때 연결 문제가 발생하면 먼저 DB 인스턴스에 연결된 VPC 보안 그룹의 아웃바운드 액세스 규칙이 네트워크 연결을 허용하는지 확인합니다. 특히 보안 그룹에는 DB

인스턴스가 TCP 트래픽을 포트 443 및 IPv4 주소(0.0.0.0/0)로 보낼 수 있도록 허용하는 규칙이 있어야 합니다. 자세한 내용은 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다.을 참조하세요.

권장 사항은 다음을 참조하세요.

- Amazon Aurora 자격 증명 및 액세스 문제 해결
- Amazon Simple Storage Service 사용 설명서의 Amazon S3 문제 해결
- IAM 사용 설명서의 Amazon S3 문제 해결 및 IAM

Aurora PostgreSQL DB 클러스터에서 AWS Lambda 함수 호출

AWS Lambda는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 이벤트 기반 컴퓨팅 서비스입니다. Aurora PostgreSQL을 비롯한 많은 AWS 서비스에서 사용할 수 있습니다. 예를 들어 Lambda 함수를 사용하여 데이터베이스의 이벤트 알림을 처리하거나 새 파일이 Amazon S3에 업로드될 때마다 파일에서 데이터를 로드할 수 있습니다. Lambda에 대한 자세한 내용은 AWS Lambda 개발자 안내서에서 AWS Lambda란 무엇입니까?를 참조하세요.

> **Note**
>
> AWS Lambda 함수 호출은 Aurora PostgreSQL 11.9 이상(Aurora Serverless v2 포함)에서 지원됩니다.

Lambda 함수와 함께 작동하도록 Aurora PostgreSQL을 설정하는 것은 AWS Lambda, IAM, VPC, Aurora PostgreSQL DB 클러스터를 포괄하는 다단계 프로세스입니다. 다음에서 필요한 단계에 대한 요약을 찾을 수 있습니다.

Lambda 함수에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 시작하기와 AWS Lambda 기본을 참조하세요.

주제

1단계: AWS Lambda에 대한 아웃바운드 연결을 위해 Aurora PostgreSQL DB 클러스터 구성

Lambda 함수는 항상 AWS Lambda 서비스가 소유한 Amazon VPC 내에서 실행됩니다. Lambda는 이 VPC에 네트워크 액세스 및 보안 규칙을 적용하고 VPC를 자동으로 유지 관리 및 모니터링합니다. Aurora PostgreSQL DB 클러스터 는 네트워크 트래픽을 Lambda 서비스의 VPC로 전송합니다. 이를 구성하는 방법은 Aurora DB 클러스터의 프라이머리 DB 인스턴스가 퍼블릭인지 프라이빗인지에 따라 다릅니다.

- 퍼블릭 Aurora PostgreSQL DB 클러스터 – DB 클러스터의 프라이머리 DB 인스턴스는 VPC의 퍼블릭 서브넷에 있고 인스턴스의 "PubliclyAccessible" 속성이 `true`인 경우 퍼블릭입니다. 이 속성의 값을 찾으려면 describe-db-instances AWS CLI 명령을 사용합니다. 또는 AWS Management Console을 사용하여 연결 및 보안 탭을 열고 퍼블릭 액세스 가능이 예인지 확인할 수 있습니다. 인스턴스가 VPC의 퍼블릭 서브넷에 있는지 확인하려면 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

  Lambda에 대한 액세스를 설정하려면 AWS Management Console 또는AWS CLI를 사용하여 VPC의 보안 그룹에 대한 아웃바운드 규칙을 생성합니다. 아웃바운드 규칙은 TCP가 포트 443을 사용하여 IPv4 주소(0.0.0.0/0)로 패킷을 보낼 수 있도록 지정합니다.

- 프라이빗 Aurora PostgreSQL DB 클러스터 – 이 경우 인스턴스의 "PubliclyAccessible" 속성은 `false`이거나 프라이빗 서브넷에 있습니다. 인스턴스가 Lambda와 함께 작동하도록 허용하려면 Network Address Translation)NAT 게이트웨이를 사용할 수 있습니다. 자세한 내용은 NAT 게이트웨이 단원을 참조하세요. 또는 Lambda용 VPC 엔드포인트로 VPC를 구성합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 참조하세요. 엔드포인트는 Aurora PostgreSQL DB 클러스터의 Lambda 함수 호출에 대한 응답을 반환합니다.

이제 VPC가 네트워크 수준에서 AWS Lambda VPC와 상호 작용할 수 있습니다. 다음으로 IAM을 사용하여 권한을 구성합니다.

2단계: Aurora PostgreSQL DB 클러스터 및 AWS Lambda에 대한 IAM 구성

Aurora PostgreSQL DB 클러스터에서 Lambda 함수를 호출하려면 특정 권한이 필요합니다. 필요한 권한을 구성하려면 Lambda 함수 호출을 허용하는 IAM 정책을 생성하고 해당 정책을 역할에 할당한 다음 DB 클러스터에 그 역할을 적용하는 것이 좋습니다. 이 접근 방식은 DB 클러스터에 사용자를 대신하여 지정된 Lambda 함수를 호출할 수 있는 권한을 부여합니다. 다음 단계에서는 AWS CLI를 사용하여 이를 수행하는 방법을 보여줍니다.

Lambda와 함께 클러스터를 사용하기 위한 IAM 권한 구성

1. create-policy AWS CLI 명령을 사용하여 Aurora PostgreSQL DB 클러스터가 지정된 Lambda 함수를 호출하도록 허용하는 IAM 정책을 생성합니다. 문 ID(Sid)는 정책 문에 대한 선택적 설명이며 사용량에 영향을 미치지 않습니다. 이 정책은 Aurora DB 클러스터에 지정된 Lambda 함수를 호출하는 데 필요한 최소 권한을 부여합니다.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

   또는 모든 Lambda 함수를 호출할 수 있도록 미리 정의된 `AWSLambdaRole` 정책을 사용할 수 있습니다. 자세한 내용은 Lambda에 대한 자격 증명 기반 IAM 정책을 참조하세요.

2. create-role AWS CLI 명령을 사용하여 정책이 런타임에 수임할 수 있는 IAM 역할을 생성합니다.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
```

```
    ]
}'
```

3. [attach-role-policy](#) AWS CLI 명령을 사용하여 역할에 정책을 적용합니다.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. [add-role-to-db-cluster](#) AWS CLI 명령을 사용하여 Aurora PostgreSQL DB 클러스터 에 역할을 적용합니다. 이 마지막 단계를 통해 DB 클러스터의 데이터베이스 사용자가 Lambda 함수를 호출할 수 있습니다.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

VPC 및 IAM 구성이 완료되면 이제 `aws_lambda` 확장을 설치할 수 있습니다. 확장은 언제든지 설치할 수 있지만 올바른 VPC 지원 및 IAM 권한을 설정할 때까지 `aws_lambda` 확장은 Aurora PostgreSQL DB 클러스터의 기능에 아무 것도 추가하지 않습니다.

3단계: Aurora PostgreSQL DB 클러스터용 **aws_lambda** 확장 설치

AWS Lambda을 Aurora PostgreSQL DB 클러스터와 사용하려면 Aurora PostgreSQL DB 클러스터에 `aws_lambda` PostgreSQL 확장을 추가합니다. 이 확장은 Aurora PostgreSQL DB 클러스터에 PostgreSQL에서 Lambda 함수를 호출할 수 있는 기능을 제공합니다.

Aurora PostgreSQL DB 클러스터에 **aws_lambda** 확장 설치

PostgreSQL `psql` 명령줄 또는 pgAdmin 도구를 사용하여 Aurora PostgreSQL DB 클러스터 에 연결합니다.

1. `rds_superuser` 권한이 있는 사용자로 Aurora PostgreSQL DB 클러스터 에 연결합니다. 기본 `postgres` 사용자가 예제에 표시됩니다.

```
psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
 5432
```

2. aws_lambda 확장을 설치합니다. aws_commons 확장도 필요합니다. 이 확장은 aws_lambda 및 기타 여러 PostgreSQL용 Aurora 확장에 대한 도우미 함수를 제공합니다. Aurora PostgreSQLDB 클러스터 에 아직 없는 경우 다음과 같이 aws_lambda와 함께 설치됩니다.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

aws_lambda 확장이 Aurora PostgreSQL DB 클러스터의 프라이머리 DB 인스턴스에 설치됩니다. 이제 Lambda 함수를 호출하기 위한 편의 구조를 생성할 수 있습니다.

4단계: Aurora PostgreSQL DB 클러스터와 함께 Lambda 도우미 함수 사용(선택 사항)

aws_commons 확장의 도우미 함수를 사용하여 PostgreSQL에서 보다 쉽게 호출할 수 있는 엔터티를 준비할 수 있습니다. 이렇게 하려면 Lambda 함수에 대한 다음 정보가 필요합니다.

- 함수 이름(Function name) – Lambda 함수의 이름, Amazon 리소스 이름(ARN), 버전 또는 별칭입니다. 2단계: 클러스터 및 Lambda에 대한 IAM 구성에서 생성한 IAM 정책에는 ARN이 필요하므로 함수의 ARN을 사용하는 것이 좋습니다.
- AWS 리전 – (선택 사항) Aurora PostgreSQL DB 클러스터와 동일한 리전에 있지 않은 경우 Lambda 함수가 있는 AWS 리전입니다.

Lambda 함수 이름 정보를 보관하려면 aws_commons.create_lambda_function_arn 함수를 사용합니다. 이 도우미 함수는 호출 함수에 필요한 세부 정보를 사용하여 aws_commons._lambda_function_arn_1 복합 구조를 생성합니다. 다음에서 이 복합 구조를 설정하는 세 가지 대안을 찾을 수 있습니다.

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
```

```
    'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

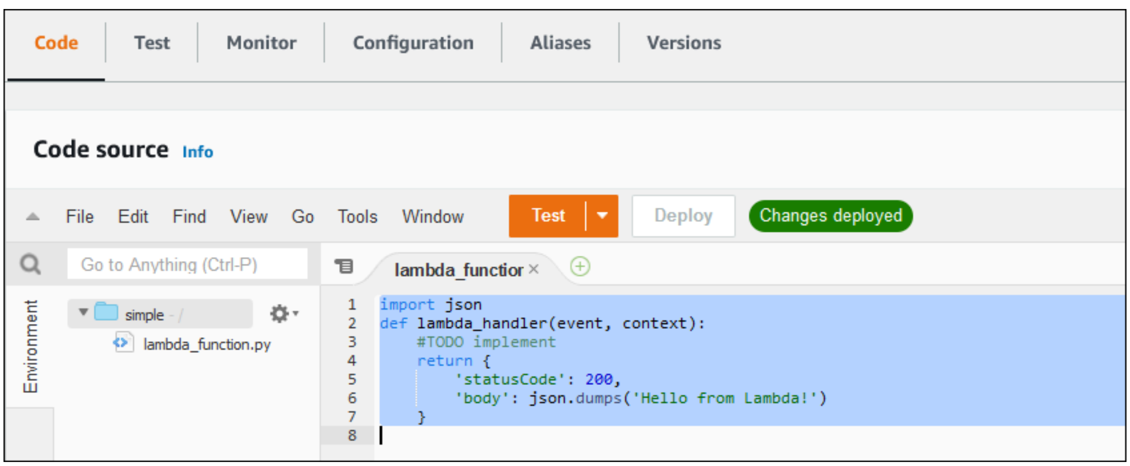
이러한 값은 aws_lambda.invoke 함수 호출에 사용할 수 있습니다. 예를 보려면 5단계: Aurora PostgreSQL DB 클러스터에서 Lambda 함수 호출 섹션을 참조하세요.

5단계: Aurora PostgreSQL DB 클러스터에서 Lambda 함수 호출

`aws_lambda.invoke` 함수는 `invocation_type`에 따라 동기식 또는 비동기식으로 작동합니다. 이 파라미터에 대한 두 가지 대안은 다음과 같이 `RequestResponse`(기본값) 및 `Event`입니다.

- **RequestResponse** - 이 호출 유형은 동기식이며, 호출 유형을 지정하지 않고 호출할 때의 기본 동작입니다. 응답 페이로드에는 `aws_lambda.invoke` 함수의 결과가 포함됩니다. 워크플로가 진행하기 전에 Lambda 함수에서 결과를 수신해야 하는 경우 이 호출 유형을 사용합니다.
- **Event** - 이 호출 유형은 비동기식이며, 응답에 결과가 포함된 페이로드가 포함되지 않습니다. 워크플로에서 처리를 계속하기 위해 Lambda 함수의 결과가 필요하지 않은 경우 이 호출 유형을 사용합니다.

설정에 대한 간단한 테스트로 `psql`을 사용하여 DB 인스턴스에 연결하고 명령줄에서 예제 함수를 호출할 수 있습니다. 다음 스크린샷에 표시된 간단한 Python 함수와 같이 Lambda 서비스에 기본 함수 중 하나가 설정되어 있다고 가정합니다.



예제 함수 호출

1. psql 또는 pgAdmin을 사용하여 프라이머리 DB 인스턴스에 연결합니다.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. ARN을 사용하여 함수를 호출합니다.

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

응답은 다음과 같습니다.

```
status_code |                        payload                        |
 executed_version | log_result
-------------+-------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

호출 시도가 성공하지 못한 경우 Lambda 함수 오류 메시지 섹션을 참조하세요.

6단계: 다른 사용자에게 Lambda 함수를 호출할 수 있는 권한 부여

이 시점에서는 자신만 rds_superuser 권한으로 Lambda 함수를 호출할 수 있습니다. 자신이 생성하는 함수를 다른 사용자가 호출할 수 있도록 허용하려면 권한을 부여해야 합니다.

Lambda 함수를 호출할 수 있는 권한을 부여하는 방법

1. psql 또는 pgAdmin을 사용하여 프라이머리 DB 인스턴스에 연결합니다.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. SQL 명령을 실행합니다.

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

예제: Aurora PostgreSQL DB 클러스터에서 Lambda 함수 호출

다음에서 aws_lambda.invoke 함수를 호출하는 몇 가지 예를 찾을 수 있습니다. 예제 대부분은 4단계: Aurora PostgreSQL DB 클러스터와 함께 Lambda 도우미 함수 사용(선택 사항)에서 생성한 복합 구조 aws_lambda_arn_1을 사용하여 함수 세부 정보 전달을 단순화합니다. 비동기 호출의 예는 예제:

Lambda 함수의 비동기 (이벤트) 호출 섹션을 참조하세요. 나열된 다른 모든 예에서 동기 호출을 사용합니다.

Lambda 호출 유형에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 함수 호출을 참조하세요. aws_lambda_arn_1에 대한 자세한 정보는 aws_commons.create_lambda_function_arn 섹션을 참조하십시오.

예제 목록

- 예제: Lambda 함수의 동기(RequestResponse) 호출
- 예제: Lambda 함수의 비동기 (이벤트) 호출
- 예: 함수 응답에서 Lambda 실행 로그 캡처
- 예제: Lambda 함수에 클라이언트 컨텍스트 포함
- 예제: 특정 버전의 Lambda 함수 호출

예제: Lambda 함수의 동기(RequestResponse) 호출

다음은 동기식 Lambda 함수 호출의 두 가지 예입니다. 이러한 aws_lambda.invoke 함수 호출의 결과는 동일합니다.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

파라미터는 다음과 같이 설명됩니다.

- :'aws_lambda_arn_1' - 이 파라미터는 4단계: Aurora PostgreSQL DB 클러스터와 함께 Lambda 도우미 함수 사용(선택 사항)에서 생성된 복합 구조를 aws_commons.create_lambda_function_arn 도우미 함수와 함께 식별합니다. 다음과 같이 aws_lambda.invoke 호출 내에서 이 구조를 인라인으로 생성할 수도 있습니다.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
 'aws-region'),
'{"body": "Hello from Postgres!"}'::json
);
```

- `'{"body": "Hello from PostgreSQL!"}'::json` – Lambda 함수에 전달할 JSON 페이로드입니다.
- `'RequestResponse'` – Lambda 호출 유형.

예제: Lambda 함수의 비동기 (이벤트) 호출

다음은 비동기 Lambda 함수 호출의 일례입니다. `Event` 호출 유형은 지정된 입력 페이로드를 사용하여 Lambda 함수 호출을 예약하고 즉시 반환합니다. Lambda 함수 결과에 의존하지 않는 특정 워크플로에서 `Event` 호출 유형을 사용합니다.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

예: 함수 응답에서 Lambda 실행 로그 캡처

`aws_lambda.invoke` 함수 호출에서 `log_type` 파라미터를 사용하여 함수 응답에 실행 로그의 마지막 4KB를 포함할 수 있습니다. 기본적으로 이 파라미터는 None으로 설정되지만 다음과 같이 `Tail`을 지정하여 응답에서 Lambda 실행 로그의 결과를 캡처할 수 있습니다.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

응답에 실행 로그를 포함하도록 aws_lambda.invoke 함수의 `log_type` 파라미터를 `Tail`로 설정합니다. `log_type` 파라미터의 기본값은 None입니다.

반환 문자열 `log_result` 은 base64 인코딩된 문자열입니다. `decode` 및 `convert_from` PostgreSQL 함수의 조합을 사용하여 내용을 디코딩할 수 있습니다.

`log_type`에 대한 자세한 정보는 aws_lambda.invoke 섹션을 참조하십시오.

예제: Lambda 함수에 클라이언트 컨텍스트 포함

`aws_lambda.invoke` 함수에는 다음과 같이 페이로드와 별도로 정보를 전달하는 데 사용할 수 있는 `context` 파라미터가 있습니다.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

클라이언트 컨텍스트를 포함하려면 aws_lambda.invoke 함수의 `context` 파라미터에 JSON 객체를 사용하세요.

`context` 파라미터에 대한 자세한 내용은 aws_lambda.invoke 레퍼런스를 참조하세요.

예제: 특정 버전의 Lambda 함수 호출

`aws_lambda.invoke` 호출에 `qualifier` 파라미터를 포함하여 Lambda 함수의 특정 버전을 지정할 수 있습니다. 다음에서 버전의 별칭으로 '*custom_version*'을 사용하여 이를 수행하는 예를 찾을 수 있습니다.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

다음과 같이 대신 함수 이름 세부 정보와 함께 Lambda 함수 한정자를 제공할 수도 있습니다.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

`qualifier` 및 기타 파라미터에 대한 자세한 내용은 aws_lambda.invoke 레퍼런스를 참조하세요.

Lambda 함수 오류 메시지

다음 목록에서 가능한 원인 및 해결 방법을 비롯하여 오류 메시지에 대한 정보를 찾을 수 있습니다.

- VPC 구성 문제

  VPC 구성 문제로 인해 연결을 시도할 때 다음과 같은 오류 메시지가 발생할 수 있습니다.

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  이 오류의 일반적인 원인은 잘못 구성된 VPC 보안 그룹입니다. VPC가 Lambda VPC에 연결할 수 있도록 VPC 보안 그룹의 포트 443에서 TCP에 대한 아웃바운드 규칙이 열려 있는지 확인합니다.

- Lambda 함수를 호출하는 데 필요한 권한 부족

  다음 오류 메시지 중 하나가 표시되면 함수를 호출하는 사용자(역할)에 적절한 권한이 없는 것입니다.

```
ERROR:  permission denied for schema aws_lambda
```

```
ERROR:  permission denied for function invoke
```

Lambda 함수를 호출하려면 사용자(역할)에게 특정 권한을 부여해야 합니다. 자세한 내용은 6단계: 다른 사용자에게 Lambda 함수를 호출할 수 있는 권한 부여을 참조하세요.

- Lambda 함수의 잘못된 오류 처리

  요청 처리 중에 Lambda 함수가 예외를 발생키면, `aws_lambda.invoke` 는 다음과 같은 PostgreSQL 오류와 함께 실패합니다.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

  Lambda 함수 또는 PostgreSQL 애플리케이션의 오류를 처리해야 합니다.

AWS Lambda 함수 및 파라미터 참조

다음은 Aurora PostgreSQL 에서 Lambda 간접 호출에 사용할 함수 및 파라미터에 대한 참조입니다.

함수 및 파라미터

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda 파라미터

aws_lambda.invoke

Aurora PostgreSQL DB 클러스터 에 대해 Lambda 함수를 실행합니다.

Lambda 함수 호출에 대한 자세한 내용은 AWS Lambda 개발자 안내서에서 호출을 참조하세요.

구문

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

입력 파라미터

function_name

Lambda 함수의 식별 이름입니다. 값은 함수 이름, ARN 또는 부분 ARN일 수 있습니다. 가능한 형식 목록은 AWS Lambda 개발자 안내서에서 Lambda 함수 이름 형식을 참조하세요.

payload

Lambda 함수에 대한 입력입니다. 형식은 JSON 또는 JSONB 일 수 있습니다. 자세한 내용은 PostgreSQL 설명서의 JSON 유형을 참조하십시오.

region

(선택 사항) 함수의 Lambda 리전입니다. 기본적으로 Aurora는 `function_name`의 전체 ARN에서 AWS 리전을 확인하거나 Aurora PostgreSQL DB 인스턴스 리전을 사용합니다. 이 리전 값이 `function_name` ARN에 제공된 값과 충돌하면 오류가 발생합니다.

invocation_type

Lambda 함수의 호출 유형입니다. 값은 대소문자를 구분합니다. 가능한 값은 다음을 포함합니다.

- `RequestResponse` – 기본값입니다. Lambda 함수에 대한 이러한 유형의 호출은 동시에 발생하며 결과에 응답 페이로드를 돌려보냅니다. 워크플로가 Lambda 함수 결과 수신에 즉시 의존하는 경우 `RequestResponse` 호출 유형을 사용합니다.
- `Event` – Lambda 함수에 대한 이러한 유형의 호출은 비동기식이며 반환된 페이로드없이 즉시 반환됩니다. 워크플로를 이동하기 전에 Lambda 함수 결과가 필요하지 않은 경우 `Event` 호출 유형을 사용합니다.
- `DryRun` – 이 유형의 호출은 Lambda 함수를 실행하지 않고 액세스를 테스트합니다.

log_type

`log_result` 출력 파라미터에 반환할 Lambda 로그의 유형입니다. 값은 대소문자를 구분합니다. 가능한 값은 다음을 포함합니다.

- – 추적 반환된 `log_result` 출력 파라미터에는 실행 로그의 마지막 4KB가 포함됩니다.
- – 없음 Lambda 로그 정보가 반환되지 않았습니다.

context

JSON 또는 JSONB 형식의 클라이언트 컨텍스트 사용할 필드에는 보다 `custom` 및 `env` 가 포함됩니다.

한정자

호출할 Lambda 함수의 버전을 식별하는 한정자입니다. 이 값이 `function_name` ARN에 제공된 값과 충돌하면 오류가 발생합니다.

출력 파라미터

status_code

HTTP 상태 응답 코드입니다. 자세한 내용은 AWS Lambda 개발자 안내서에서 [Lambda 호출 응답 요소]를 참조하세요.

payload

실행된 Lambda 함수에서 반환된 정보입니다. 형식은 JSON 또는 JSONB입니다.

executed_version

실행된 Lambda 함수 버전입니다.

log_result

Lambda 함수가 호출 될 때 `log_type` 값이 `Tail` 인 경우 반환된 실행 로그 정보입니다. 결과에는 Base64로 인코딩된 실행 로그의 마지막 4KB가 포함됩니다.

aws_commons.create_lambda_function_arn

Lambda 파일 정보를 저장할 `aws_commons._lambda_function_arn_1` 구조를 생성합니다. aws_lambda.invoke `aws_commons.create_lambda_function_arn` 함수의 `function_name` 파라미터에 [aws_lambda.invoke] 함수의 결과를 사용할 수 있습니다.

구문

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

입력 파라미터

function_name

Lambda 함수 이름이 포함된 필수 텍스트 문자열입니다. 값은 함수 이름, 부분 ARN 또는 전체 ARN일 수 있습니다.

region

Lambda 함수가 있는 AWS 리전을 포함하는 선택적 텍스트 문자열입니다. 리전 이름 및 연결된 값의 목록은 섹션을 참조하십시오[리전 및 가용 영역](#)

aws_lambda 파라미터

이 표에서는 `aws_lambda` 함수와 관련된 파라미터를 찾아볼 수 있습니다.

파라미터	설명
`aws_lambda.connect_timeout_ms`	이는 동적 파라미터이며 AWS Lambda에 연결하는 동안 최대 대기 시간을 설정합니다. 기본값은 `1000`입니다. 이 파라미터에 허용되는 값은 1~900,000입니다.
`aws_lambda.request_timeout_ms`	이는 동적 파라미터이며 AWS Lambda의 응답을 기다리는 동안 최대 대기 시간을 설정합니다. 기본값은 `3000`입니다. 이 파라미터에 허용되는 값은 1~900,000입니다.
`aws_lambda.endpoint_override`	AWS Lambda에 연결하는 데 사용할 수 있는 엔드포인트를 지정합니다. 빈 문자열은 해당 리전의 기본 AWS Lambda 엔드포인트를 선택합니다. 이 정적 파라미터 변경을 적용하려면 데이터베이스를 다시 시작해야 합니다.

Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시

Amazon CloudWatch Logs로 로그 데이터를 정기적으로 내보내도록 Aurora PostgreSQL DB 클러스터를 구성할 수 있습니다. 그러면 Aurora PostgreSQL DB 클러스터의 PostgreSQL 로그 이벤트가 Amazon CloudWatch Logs로 Amazon CloudWatch에 자동으로 게시됩니다. CloudWatch의 내보낸 로그 데이터는 Aurora PostgreSQL DB 클러스터의 로그 그룹에서 찾을 수 있습니다. 로그 그룹에는 클러스터에 있는 각 인스턴스의 PostgreSQL 로그 이벤트가 포함된 하나 이상의 로그 스트림이 포함됩니다.

CloudWatch Logs에 로그를 게시하면 클러스터의 PostgreSQL 로그 레코드를 내구성이 뛰어난 스토리지에 보관할 수 있습니다. CloudWatch Logs에서 사용 가능한 로그 데이터를 사용하여 클러스터 작업을 평가하고 개선할 수 있습니다. CloudWatch에서 경보를 생성하고 지표를 볼 수도 있습니다. 자세한 내용은 [Amazon CloudWatch에서 로그 이벤트 모니터링](#) 단원을 참조하세요.

> **Note**
>
> CloudWatch Logs에 PostgreSQL 로그를 게시하면 스토리지가 사용되며 해당 스토리지에 대한 요금이 부과됩니다. 더 이상 필요하지 않은 CloudWatch Logs는 삭제해야 합니다.

기존 Aurora PostgreSQL DB 클러스터에 대한 로그 내보내기 옵션을 꺼도 CloudWatch Logs에 이미 보관된 데이터에는 영향을 미치지 않습니다. 기존 로그는 로그 보관 설정에 따라 CloudWatch Logs에서 계속 사용할 수 있습니다. CloudWatch Logs에 대한 자세한 내용은 Amazon CloudWatch Logs 란?을 참조하세요.

Aurora PostgreSQL은 다음 버전에 대해 CloudWatch Logs에 로그 게시를 지원합니다.

- 16.1 이상의 모든 버전
- 15.2 이상의 15 버전
- 14.3 이상의 14 버전
- 13.3 이상의 13 버전
- 12.8 이상의 12 버전
- 11.9 이상 11 버전

CloudWatch Logs에 로그를 게시하는 옵션을 활성화하고, CloudWatch Logs의 로그 이벤트를 모니터링하고, CloudWatch Logs Insights를 사용하여 로그를 분석하는 방법에 대한 자세한 내용은 다음 주제를 참조하세요.

주제

- Amazon CloudWatch로 로그를 게시하는 옵션 설정
- Amazon CloudWatch에서 로그 이벤트 모니터링
- CloudWatch Logs Insights를 사용하여 PostgreSQL 로그 분석

Amazon CloudWatch로 로그를 게시하는 옵션 설정

Aurora PostgreSQL DB 클러스터의 PostgreSQL 로그를 CloudWatch Logs에 게시하려면 클러스터에 대한 Log export(로그 내보내기) 옵션을 선택합니다. Aurora PostgreSQL DB 클러스터를 생성할 때 로그 내보내기 설정을 선택할 수 있습니다. 클러스터는 나중에 수정해도 됩니다. 기존 클러스터를 수정하면 해당 시점부터 각 인스턴스의 PostgreSQL 로그가 CloudWatch 클러스터에 게시됩니다. Aurora PostgreSQL의 경우 PostgreSQL 로그(`postgresql.log`)는 Amazon CloudWatch에 게시되는 유일한 로그입니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora PostgreSQL DB 클러스터에 대한 로그 내보내기 기능을 설정 수 있습니다.

콘솔

로그 내보내기 옵션을 선택하여 Aurora PostgreSQL DB 클러스터에서 CloudWatch Logs로 PostgreSQL 로그 게시를 시작합니다.

콘솔에서 로그 내보내기 기능을 켜려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. CloudWatch Logs에 게시하려는 로그 데이터가 있는 Aurora PostgreSQL DB 클러스터를 선택합니다.
4. 수정을 선택합니다.
5. Log exports(로그 내보내기) 섹션에서 PostgreSQL log(PostgreSQL 로그를 선택합니다.
6. 계속을 선택한 후, 요약 페이지에서 클러스터 수정을 선택합니다.

AWS CLI

로그 내보내기 옵션을 켜서 AWS CLI를 사용하여 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시를 시작할 수 있습니다. 이를 위해서는 modify-db-cluster AWS CLI 명령을 다음 옵션과 함께 실행해야 합니다.

- `--db-cluster-identifier` - DB 클러스터 식별자입니다.
- `--cloudwatch-logs-export-configuration` - DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 설정할 로그 유형의 구성 설정입니다.

또한 다음 AWS CLI 명령 중 하나를 실행하여 Aurora PostgreSQL 로그를 게시할 수 있습니다.

- [create-db-cluster]
- [restore-db-cluster-from-s3]
- [restore-db-cluster-from-snapshot]
- [restore-db-cluster-to-point-in-time]

다음 옵션으로 AWS CLI 명령 중 하나를 실행합니다.

- `--db-cluster-identifier`—DB 클러스터 식별자입니다.
- `--engine`—데이터베이스 엔진입니다.
- `--enable-cloudwatch-logs-exports`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

실행하는 AWS CLI 명령에 따라 다른 옵션이 필요할 수 있습니다.

Example

다음 명령은 CloudWatch Logs에 로그 파일을 게시하도록 Aurora PostgreSQL DB 클러스터를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
```

```
    --enable-cloudwatch-logs-exports postgresql
```

Example

다음 명령은 CloudWatch Logs에 로그 파일을 게시하도록 기존 Aurora PostgreSQL DB 클러스터를 수정합니다. `--cloudwatch-logs-export-configuration` 값은 JSON 객체입니다. 이 객체의 키는 `EnableLogTypes`이고 그 값은 `postgresql` 및 `instance`입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","instance"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\",
\"instance\"]}'
```

Note

Windows 명령 프롬프트를 사용하는 경우 백슬래시(\)를 접두사로 추가하여 JSON 코드에서 큰따옴표(")를 이스케이프해야 합니다.

Example

다음 예에서는 CloudWatch Logs에 로그 파일 게시를 비활성화하도록 기존 Aurora PostgreSQL DB 클러스터를 수정합니다. `--cloudwatch-logs-export-configuration` 값은 JSON 객체입니다. 이 객체의 키는 `DisableLogTypes`이고 그 값은 `postgresql` 및 `instance`입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":
["postgresql","instance"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\",
\"instance\"]}"
```

> **Note**
>
> Windows 명령 프롬프트를 사용하는 경우 백슬래시(\)를 접두사로 추가하여 JSON 코드에서 큰따옴표(")를 이스케이프해야 합니다.

RDS API

로그 내보내기 옵션을 켜서 AWS CLI를 사용하여 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시를 시작할 수 있습니다. 이를 위해서는 다음 옵션과 함께 ModifyDBCluster 작업을 실행해야 합니다.

- `DBClusterIdentifier` - DB 클러스터 식별자입니다.
- `CloudwatchLogsExportConfiguration` - DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

또한 다음 RDS API 작업 중 하나를 실행하여 RDS API로 Aurora MySQL 로그를 게시할 수 있습니다.

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

다음 파라미터로 RDS API 작업을 실행합니다.

- `DBClusterIdentifier`—DB 클러스터 식별자입니다.
- `Engine`—데이터베이스 엔진입니다.
- `EnableCloudwatchLogsExports`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 로그 유형의 구성 설정입니다.

실행하는 AWS CLI 명령에 따라 다른 파라미터가 필요할 수 있습니다.

Amazon CloudWatch에서 로그 이벤트 모니터링

Aurora PostgreSQL 로그 이벤트를 게시하고 Amazon CloudWatch Logs로 사용할 수 있으므로 Amazon CloudWatch를 사용하여 이벤트를 보고 모니터링할 수 있습니다. 모니터링에 대한 자세한 내용은 [CloudWatch Logs로 전송된 로그 데이터 보기]를 참조하세요.

로그 내보내기를 켜면 다음 패턴과 같이 Aurora PostgreSQL 이름과 로그 유형이 있는 접두사 `/aws/rds/cluster/`를 사용하여 새 로그 그룹이 자동으로 생성됩니다.

```
/aws/rds/cluster/your-cluster-name/postgresql
```

예를 들어 `docs-lab-apg-small`이라는 Aurora PostgreSQL DB 클러스터가 로그를 Amazon CloudWatch Logs로 내보낸다고 가정합니다. Amazon CloudWatch의 로그 그룹 이름은 다음과 같습니다.

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

지정된 이름이 있는 로그 그룹이 존재할 경우 Aurora은 이 로그 그룹을 사용하여 Aurora DB 클러스터의 로그 데이터를 내보냅니다. Aurora PostgreSQL DB 클러스터의 각 DB 인스턴스는 PostgreSQL 로그를 고유한 로그 스트림으로 로그 그룹에 업로드합니다. Amazon CloudWatch에서 사용할 수 있는 다양한 그래픽 및 분석 도구를 사용하여 로그 그룹과 해당 로그 스트림을 검사할 수 있습니다.

예를 들어 Aurora PostgreSQL DB 클러스터에서 로그 이벤트 내 정보를 검색하고 CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 이벤트를 필터링할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 [로그 데이터 검색 및 필터링]을 참조하세요.

기본적으로 새 로그 그룹은 보관 기간 동안 Never expire(만기 없음)를 사용하여 생성됩니다. CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 로그 보관 기간을 변경할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 [CloudWatch Logs에서 로그 데이터 보관 변경]을 참조하세요.

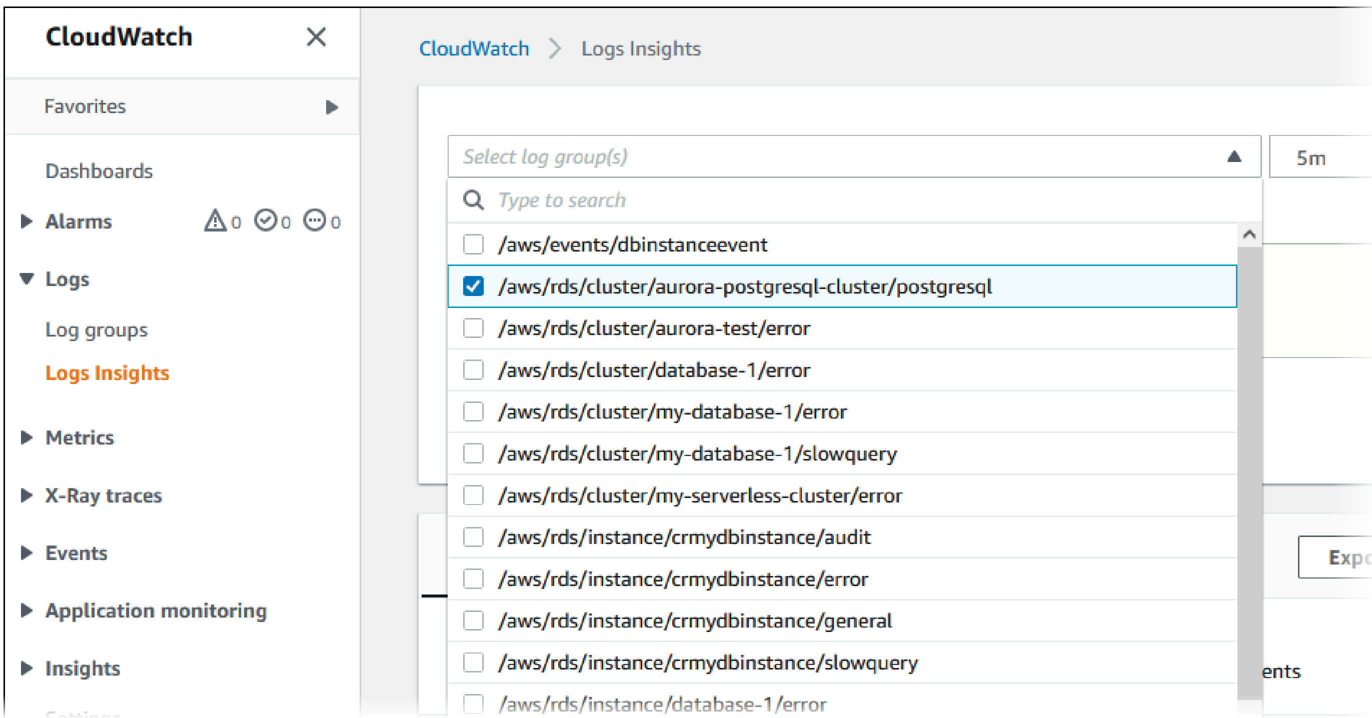
> **Tip**
>
> AWS CloudFormation 같은 자동 구성을 사용하여 미리 정의된 로그 보관 기간, 메트릭 필터 및 액세스 권한이 있는 로그 그룹을 생성할 수 있습니다.

CloudWatch Logs Insights를 사용하여 PostgreSQL 로그 분석

Aurora PostgreSQL DB 클러스터의 PostgreSQL 로그를 CloudWatch Logs로 게시하면 CloudWatch Logs Insights를 사용하여 Amazon CloudWatch Logs의 로그 데이터를 대화식으로 검색하고 분석할 수 있습니다. CloudWatch Logs Insights에는 잠재적 문제를 식별하고 수정 사항을 확인할 수 있도록 로그 데이터를 분석하기 위한 쿼리 언어, 샘플 쿼리 및 기타 도구가 포함되어 있습니다. 자세히 알아보려면 Amazon CloudWatch Logs 사용 설명서의 [CloudWatch Logs Insights에서 로그 데이터 분석](#)을 참조하세요.

CloudWatch Logs 인사이트를 사용하여 PostgreSQL 로그 분석

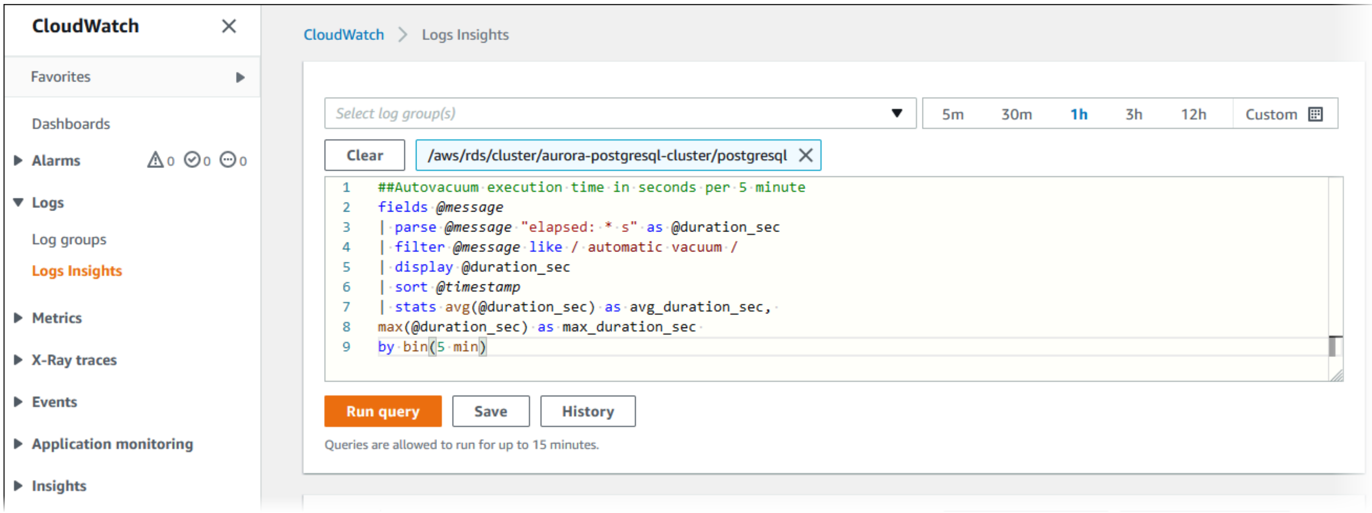
1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창의 로그에서 로그 인사이트를 선택합니다.
3. Select log group(s)(로그 그룹 선택)에서 DB 클러스터의 로그 그룹을 선택합니다.



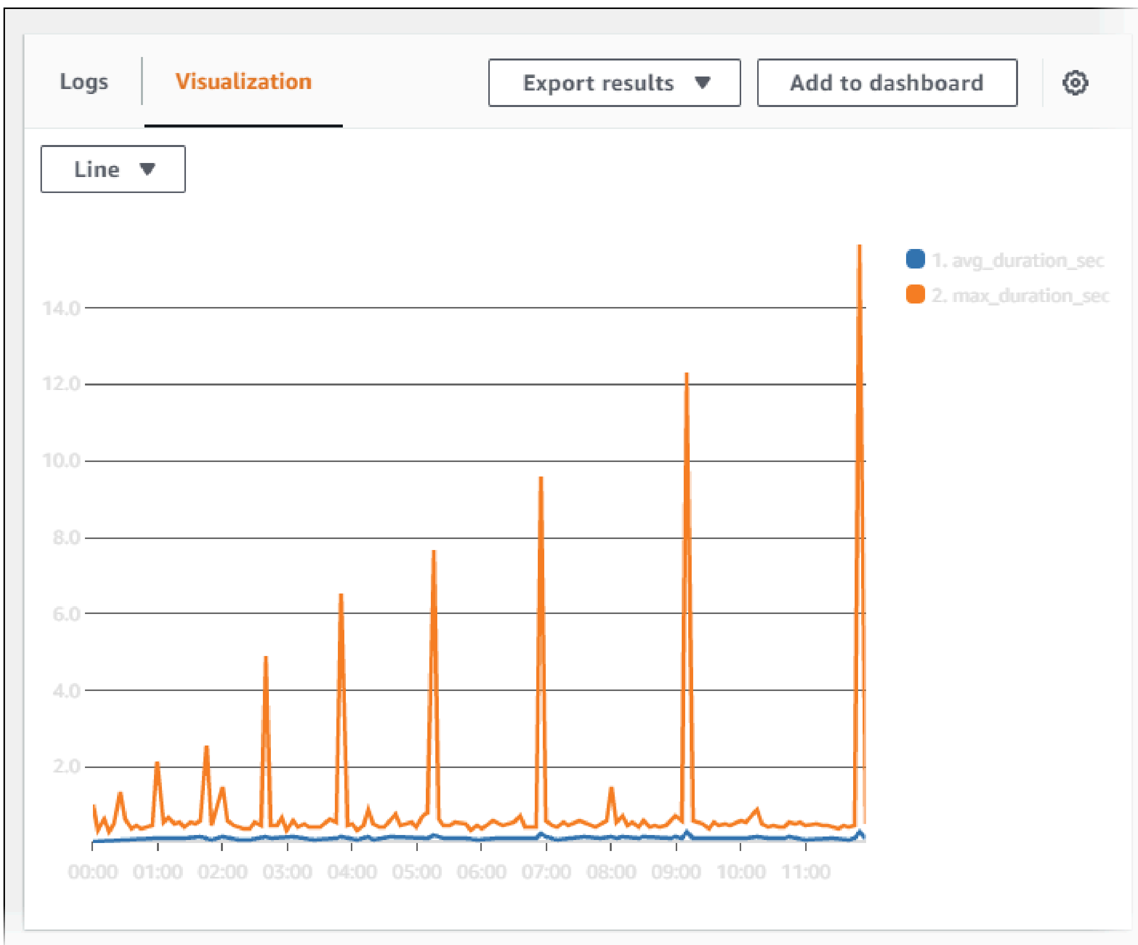
4. 쿼리 편집기에서 현재 표시되어 있는 쿼리를 삭제하고 다음을 입력한 후 쿼리 실행(Run query)을 선택합니다.

```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
```

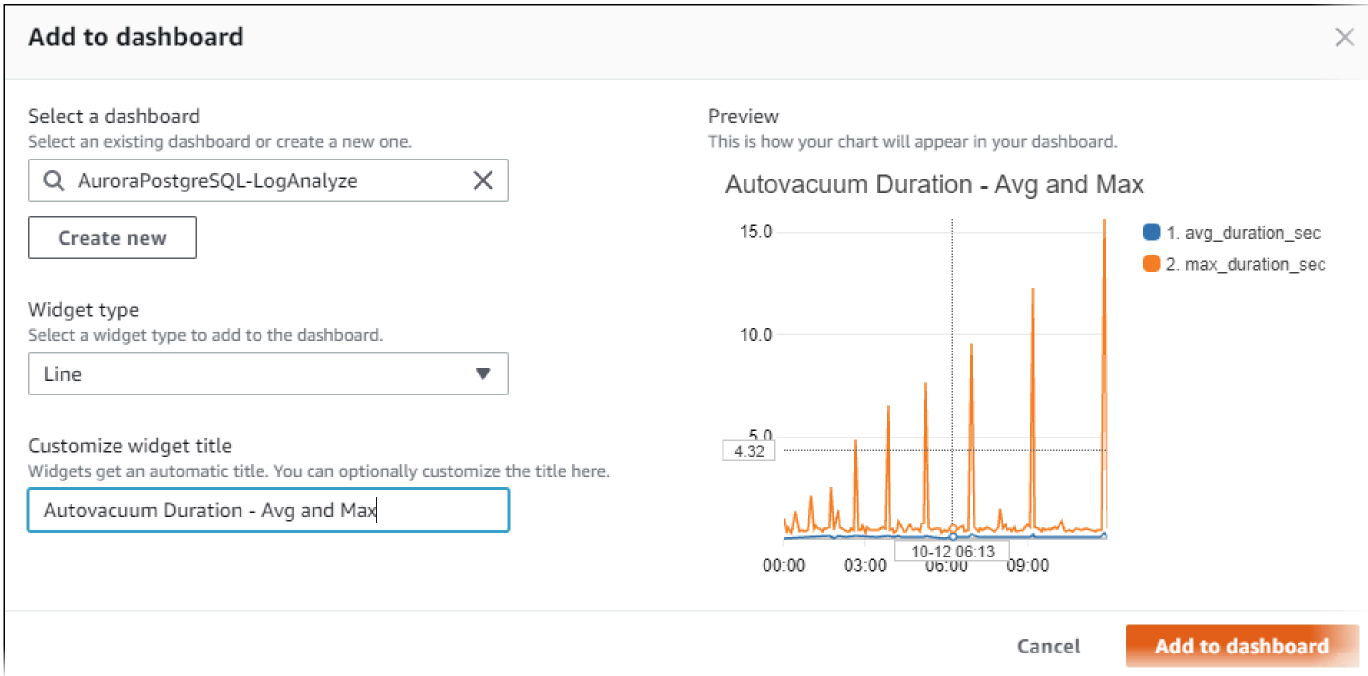
```
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



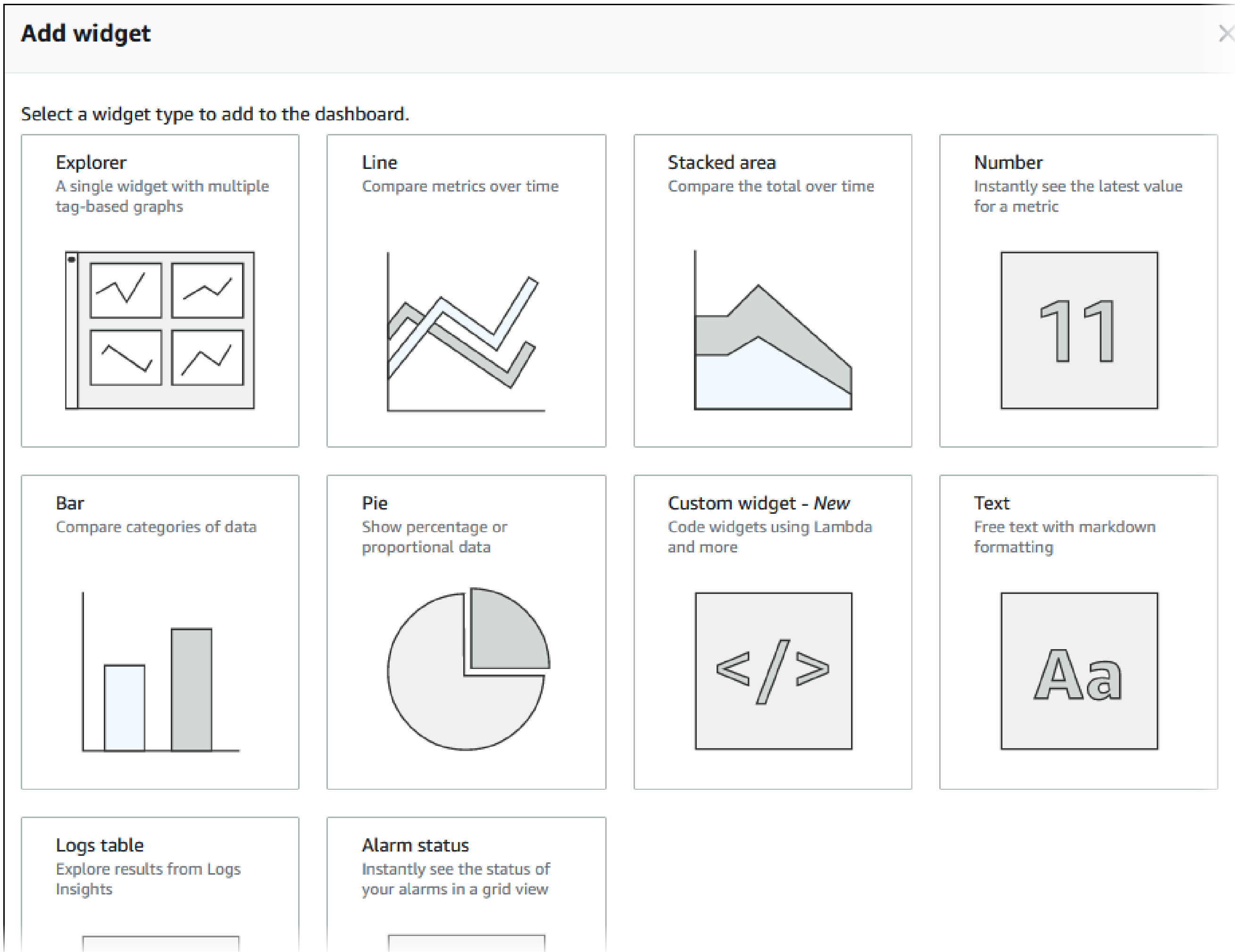
5. 시각화(Visualization) 탭을 선택합니다.



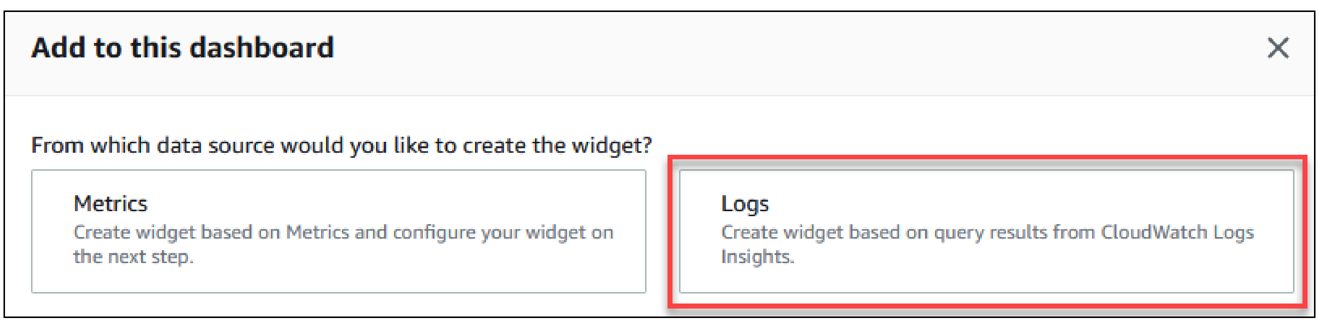
6. 대시보드에 추가(Add to dashboard)를 선택합니다.
7. 대시보드 선택(Select a dachboard)에서 대시보드를 선택하거나 이름을 입력하여 새 대시보드를 만듭니다.
8. 위젯 유형(Widget type)에서 시각화에 사용할 위젯 유형을 선택합니다.



9. (선택 사항) 로그 쿼리 결과에 따라 위젯을 더 추가합니다.
   a. 위젯 추가를 선택합니다.
   b. 위젯 유형을 선택합니다(예:행).



c. 이 대시보드에 추가(Add to this dashboard) 창에서 로그를 선택합니다.

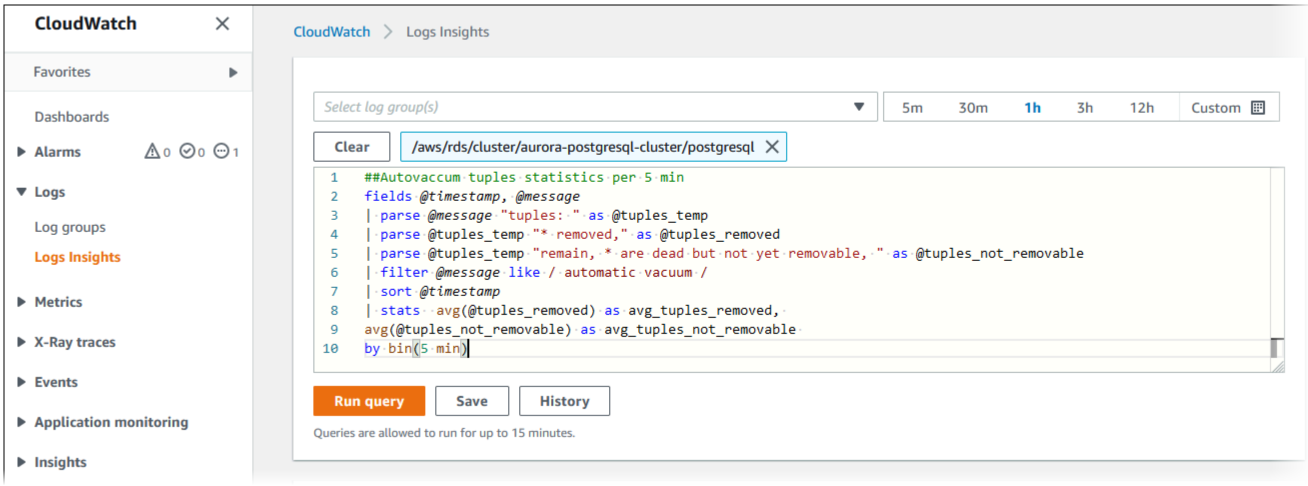


d. 로그 그룹 선택(Select log group(s))에서 DB 클러스터의 로그 그룹을 선택합니다.

e. 쿼리 편집기에서 현재 표시되어 있는 쿼리를 삭제하고 다음을 입력한 후 쿼리 실행(Run query)을 선택합니다.

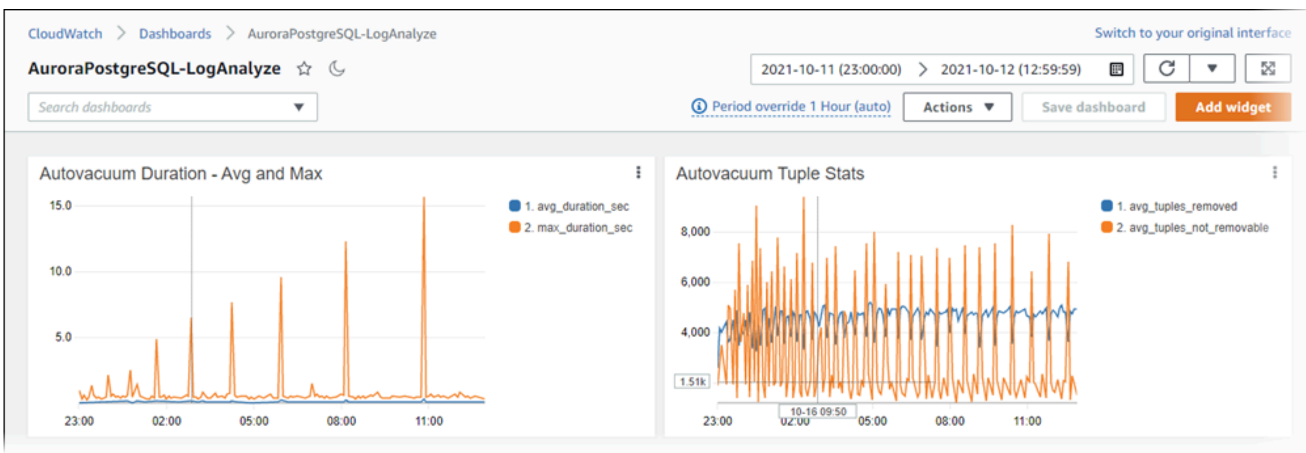
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. 위젯 생성을 선택합니다.

대시보드가 다음 이미지와 비슷해야 합니다.



Aurora PostgreSQL용 쿼리 실행 계획 및 피크 메모리 모니터링

Aurora PostgreSQL DB 인스턴스에서 쿼리 실행 계획을 모니터링하여 현재 데이터베이스 로드에 영향을 미치는 실행 계획을 감지하고, `aurora_compute_plan_id` 파라미터를 사용하여 시간 경과에 따른 실행 계획의 성능 통계를 추적할 수 있습니다. 쿼리가 실행될 때마다 쿼리에서 사용하는 실행 계획에 식별자가 할당되고 동일한 계획의 후속 실행에서 동일한 식별자가 사용됩니다.

`aurora_compute_plan_id`는 Aurora PostgreSQL 버전 14.10, 15.5 이상 버전의 DB 파라미터 그룹에서 기본적으로 `OFF`로 설정되어 있습니다. 계획 식별자를 할당하려면 파라미터 그룹에서 `aurora_compute_plan_id`를 `ON`으로 설정합니다.

이 계획 식별자는 용도가 다른 여러 유틸리티에서 사용됩니다.

DB 인스턴스의 쿼리 피크 메모리 사용량을 모니터링하여 다음 버전에서 데이터베이스 메모리 사용량을 높이는 쿼리를 감지할 수 있습니다.

- 16.3 이상의 모든 버전
- 15.7 이상 버전
- 14.12 이상 버전

쿼리가 실행될 때마다 쿼리에 사용된 피크 메모리가 추적됩니다. 쿼리는 일반적으로 여러 번 실행됩니다. 각 쿼리에 대해 모든 실행의 평균, 최소 및 최대 메모리 사용량 값을 볼 수 있습니다.

주제

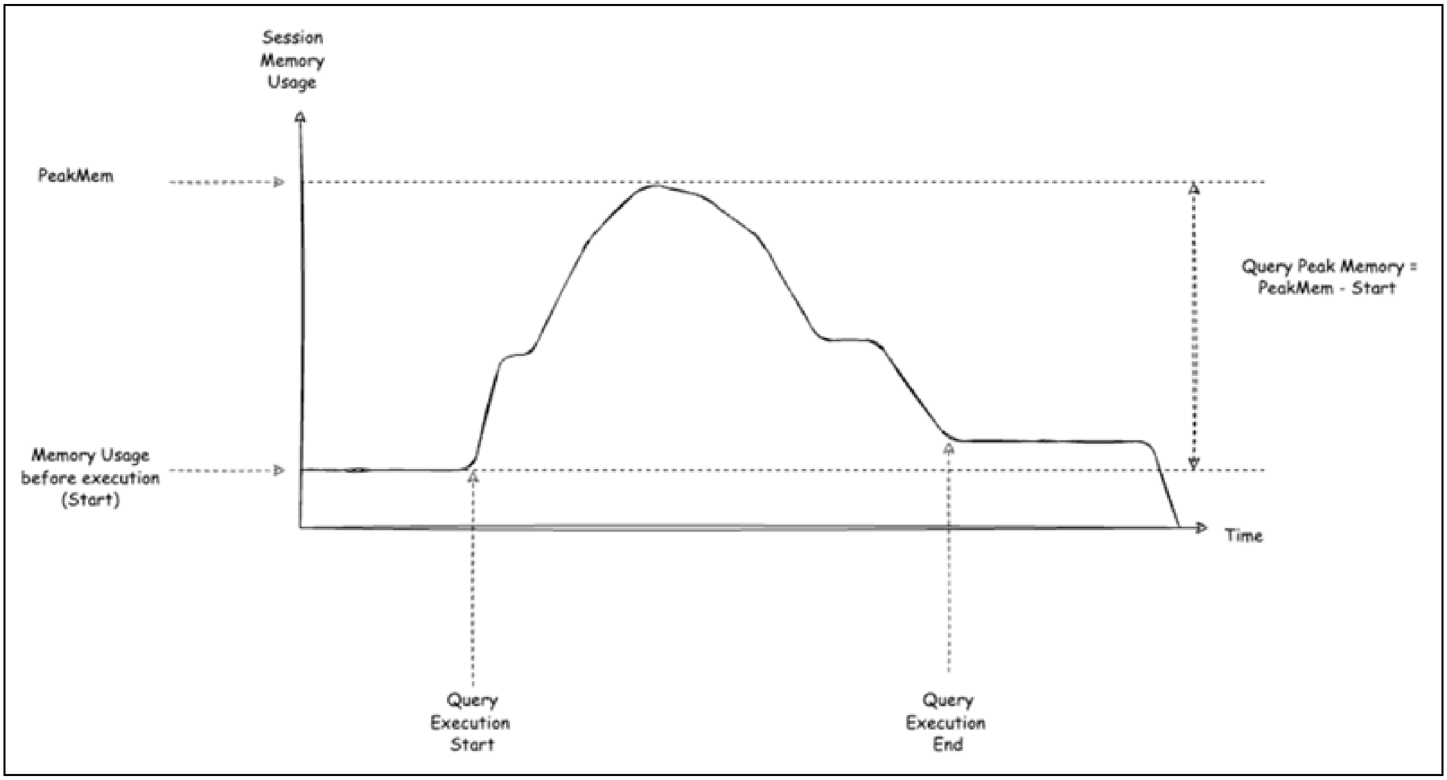
- [Aurora 함수를 사용하여 쿼리 실행 계획 및 피크 메모리에 액세스]
- [Aurora PostgreSQL 쿼리 실행 계획에 대한 파라미터 참조]

Aurora 함수를 사용하여 쿼리 실행 계획 및 피크 메모리에 액세스

`aurora_compute_plan_id`를 사용하면 다음 함수를 사용하여 실행 계획에 액세스할 수 있습니다.

- aurora_stat_activity
- aurora_stat_plans

쿼리 피크 메모리에는 쿼리 처리가 시작되기 전에 할당된 메모리가 포함되지 않습니다. 피크 메모리 사용량은 각 쿼리의 계획 및 실행 단계에서 개별적으로 추적 및 보고됩니다.



다음 함수를 사용하여 쿼리 피크 메모리 통계에 액세스할 수 있습니다.

- aurora_stat_statements
- aurora_stat_plans

이러한 함수에 관한 자세한 내용은 Aurora PostgreSQL 함수 참조 섹션을 참조하세요.

Aurora PostgreSQL 쿼리 실행 계획에 대한 파라미터 참조

DB 파라미터 그룹의 아래 파라미터를 사용하여 쿼리 실행 계획을 모니터링할 수 있습니다.

파라미터

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analyze
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers

- aurora_stat_plans.with_wal
- aurora_stat_plans.with_triggers

> **Note**
>
> aurora_stat_plans.with_* 파라미터 구성은 새로 캡처한 계획에만 적용됩니다.

aurora_compute_plan_id

aurora_compute_plan_id는 쿼리 실행 중에 계획 식별자가 할당되는지를 제어하는 구성 파라미터입니다.

Default	허용된 값	설명
끄기	0(꺼짐)	계획 식별자가 할당되지 않도록 하려면 off로 설정하세요.
	1(켜짐)	계획 식별자를 할당하려면 on으로 설정합니다.

aurora_stat_plans.minutes_until_recapture

계획이 다시 캡처되기까지 걸리는 시간(분)입니다. 기본값은 0이며, 이 경우 계획을 다시 캡처할 수 없습니다. aurora_stat_plans.calls_until_recapture 임계값을 초과한 경우 계획을 다시 캡처할 수 있습니다.

Default	허용된 값	설명
0	0-1073741823	계획이 다시 캡처되기까지 걸리는 시간(분)을 설정합니다.

aurora_stat_plans.calls_until_recapture

플랜이 다시 캡처되기 전 해당 플랜을 호출한 횟수입니다. 기본값은 0이며, 이 경우 해당 횟수만큼 플랜이 호출된 후에는 계획을 다시 캡처할 수 없습니다. aurora_stat_plans.minutes_until_recapture 임계값을 초과한 경우 계획을 다시 캡처할 수 있습니다.

Default	허용된 값	설명
0	0-1073741823	플랜이 다시 캡처되기 전까지 가능한 호출 횟수를 설정합니다.

aurora_stat_plans.with_costs

예상 비용이 포함된 EXPLAIN 계획을 캡처합니다. 허용 값은 on, off입니다. 기본값은 on입니다.

Default	허용된 값	설명
on	0(꺼짐)	각 계획 노드의 예상 비용 및 행을 표시하지 않습니다.
	1(켜짐)	각 계획 노드의 예상 비용과 행을 표시합니다.

aurora_stat_plans.with_analyze

ANALYZE를 사용하여 EXPLAIN 계획을 제어합니다. 이 모드는 계획을 처음 캡처할 때만 사용됩니다. 허용 값은 on, off입니다. 기본값은 off입니다.

Default	허용된 값	설명
끄기	0(꺼짐)	계획의 실제 실행 시간 통계는 포함하지 않습니다.
	1(켜짐)	계획의 실제 실행 시간 통계를 포함합니다.

aurora_stat_plans.with_timing

ANALYZE 사용 시 설명에 계획 타이밍이 캡처됩니다. 기본값은 on입니다.

Default	허용된 값	설명
on	0(꺼짐)	실제 시작 시간 및 각 계획 노드에서 소요된 시간을 포함하지 않습니다.
	1(켜짐)	실제 시작 시간 및 각 계획 노드에서 소요된 시간을 포함합니다.

aurora_stat_plans.with_buffers

ANALYZE 사용 시 설명에 계획 버퍼 사용량 통계가 캡처됩니다. 기본값은 off입니다.

Default	허용된 값	설명
끄기	0(꺼짐)	버퍼 사용량에 대한 정보를 포함하지 않습니다.
	1(켜짐)	버퍼 사용에 대한 정보를 포함합니다.

aurora_stat_plans.with_wal

ANALYZE 사용 시 설명에 계획 WAL 사용량 통계가 캡처됩니다. 기본값은 off입니다.

Default	허용된 값	설명
끄기	0(꺼짐)	WAL 레코드 생성에 대한 정보를 포함하지 않습니다.
	1(켜짐)	WAL 레코드 생성에 대한 정보를 포함합니다.

aurora_stat_plans.with_triggers

ANALYZE 사용 시 설명에 계획 트리거 실행 통계가 캡처됩니다. 기본값은 off입니다.

Default	허용된 값	설명
끄기	0(꺼짐)	트리거 실행 통계를 포함하지 않습니다.
	1(켜짐)	트리거 실행 통계를 포함합니다.

Aurora PostgreSQL용 쿼리 실행 계획 관리

Aurora PostgreSQL 쿼리 계획 관리는 Amazon Aurora PostgreSQL 호환 버전의 DB 클러스터와 함께 사용할 수 있는 선택적 기능입니다. 이 기능은 Aurora PostgreSQL DB 클러스터에 설치할 수 있는 `apg_plan_mgmt` 확장으로 패키징되어 있습니다. 쿼리 계획 관리를 사용하면 SQL 애플리케이션용 최적화 프로그램에서 생성한 쿼리 실행 계획을 관리할 수 있습니다. `apg_plan_mgmt` AWS 확장은 PostgreSQL 데이터베이스 엔진의 기본 쿼리 처리 기능을 기반으로 합니다.

다음에서는 Aurora PostgreSQL 쿼리 계획 관리 기능, 설정 방법 및 Aurora PostgreSQL DB 클러스터에서 이를 사용하는 방법에 대한 정보를 확인할 수 있습니다. 시작하기 전에 Aurora PostgreSQL 버전에 사용할 수 있는 `apg_plan_mgmt` 확장의 특정 버전에 대한 릴리스 노트를 검토하는 것이 좋습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 Aurora PostgreSQL apg_plan_mgmt 확장 버전을 참조하세요.

주제

Aurora PostgreSQL 쿼리 계획 관리 개요

Aurora PostgreSQL 쿼리 계획 관리는 쿼리 계획 회귀를 유발할 수 있는 데이터베이스 변경과 관계없이 계획 안정성을 보장하도록 설계되었습니다. 쿼리 계획 회귀는 최적화 프로그램이 시스템 또는 데이

터베이스가 변경된 후 지정된 SQL 문에 대해 최적이 아닌 계획을 선택할 때 발생합니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩, PostgreSQL 데이터베이스 엔진으로의 업그레이드에 대한 변경 사항을 들 경우 계획 회귀가 발생할 수 있습니다.

Aurora PostgreSQL 쿼리 계획 관리를 사용하면 쿼리 실행 계획을 변경하는 방식과 시점을 제어할 수 있습니다. Aurora PostgreSQL 쿼리 계획 관리의 이점은 다음과 같습니다.

- 강제로 최적화 프로그램이 소수의 알려진 정상 계획 중 하나를 선택하도록 하여 계획의 안정성을 높일 수 있습니다.
- 계획을 중앙에서 최적화한 다음, 최고의 계획을 전역에 배포할 수 있습니다.
- 사용되지 않는 인덱스를 식별하고 인덱스 생성 및 삭제의 영향을 평가할 수 있습니다.
- 최적화 프로그램에서 발견한 새로운 최소 비용 계획을 자동으로 감지할 수 있습니다.
- 성능을 개선하는 계획 변경 사항만 승인하도록 선택할 수 있어 위험이 더 적은 새로운 최적화 프로그램 기능을 사용해 볼 수 있습니다.

쿼리 계획 관리에서 제공하는 도구를 사전에 사용하여 특정 쿼리에 가장 적합한 계획을 지정할 수 있습니다. 또는 쿼리 계획 관리를 사용하여 변화하는 환경에 대응하고 계획 회귀를 방지할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL 쿼리 계획 관리에 대한 모범 사례](#) 섹션을 참조하세요.

주제

지원되는 SQL 문

쿼리 계획 관리는 다음 유형의 SQL 문을 지원합니다.

- 복잡성과 관계없이 SELECT, INSERT, UPDATE 또는 DELETE 문
- 준비된 문. 자세한 내용은 PostgreSQL 설명서의 PREPARE를 참조하세요.

- 동적 문(즉시 실행 모드에서 실행되는 문 포함). 자세한 내용은 PostgreSQL 설명서의 Dynamic SQL(동적 SQL) 및 EXECUTE IMMEDIATE를 참조하세요.
- 내장된 SQL 명령 및 문. 자세한 내용은 PostgreSQL 설명서의 Embedded SQL Commands(내장된 SQL 명령)를 참조하세요.
- 명명된 함수 내의 문. 자세한 내용은 PostgreSQL 설명서에서 CREATE FUNCTION을 참조하세요.
- 임시 테이블을 포함하는 문
- 프로시저 및 DO 블록 내부의 문

쿼리 계획 관리를 수동 모드에서 EXPLAIN와 함께 사용하면 실제로 실행하지 않고도 계획을 캡처할 수 있습니다. 자세한 내용은 최적화 프로그램이 선택한 계획 분석 섹션을 참조하세요. 쿼리 계획 관리 모드(수동, 자동)에 대한 자세한 내용은 Aurora PostgreSQL 실행 계획 캡처 섹션을 참조하세요.

Aurora PostgreSQL 쿼리 계획 관리는 파티셔닝된 테이블, 상속, 행 수준의 보안 및 재귀적인 공통 테이블 표현식(CTE)을 비롯하여 모든 PostgreSQL 언어 기능을 지원합니다. 이러한 PostgreSQL 언어 기능에 대해 자세히 알아보려면 PostgreSQL 설명서의 Table Partitioning(테이블 파티셔닝), Row Security Policies(행 보안 정책), WITH Queries (Common Table Expressions)(WITH 쿼리(공통 테이블 표현식)) 및 기타 주제를 참조하세요.

Aurora PostgreSQL 쿼리 계획 관리 기능의 다양한 버전에 관한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 Aurora PostgreSQL apg_plan_mgmt 확장 버전을 참조하세요.

쿼리 계획 관리의 제한 사항

Aurora PostgreSQL 쿼리 계획 관리의 현재 릴리스에는 다음과 같은 제한이 적용됩니다.

- 시스템 관계를 참조하는 문에 대한 계획은 캡처되지 않음 - 시스템 관계를 참조하는 문(예: pg_class)은 캡처되지 않습니다. 이는 내부적으로 사용되는 대량의 시스템 생성 계획이 캡처되는 것을 방지하기 위해 의도적으로 설계된 것입니다. 보기 내부의 시스템 테이블에도 적용됩니다.
- Aurora PostgreSQL DB 클러스터에 더 큰 DB 인스턴스 클래스가 필요할 수 있음 - 워크로드에 따라 쿼리 계획 관리에 vCPU가 2개 이상인 DB 인스턴스 클래스가 필요할 수 있습니다. max_worker_processes 수는 DB 인스턴스 클래스 크기에 따라 제한됩니다. 2-vCPU DB 인스턴스 클래스(예: db.t3.medium)에서 제공된 max_worker_processes의 수가 지정된 워크로드에 충분하지 않을 수 있습니다. 쿼리 계획 관리를 사용하는 경우 Aurora PostgreSQL DB 클러스터클러스터에 vCPU가 2개 이상인 DB 인스턴스 클래스를 선택하는 것이 좋습니다.

  DB 인스턴스 클래스가 워크로드를 지원하지 못하는 경우, 쿼리 계획 관리에서 다음과 같은 오류 메시지가 발생합니다.

```
WARNING: could not register plan insert background process
HINT: You may need to increase max_worker_processes.
```

이 경우 Aurora PostgreSQL DB 클러스터를 메모리가 더 많은 DB 인스턴스 클래스 크기로 스케일 업해야 합니다. 자세한 내용은 DB 인스턴스 클래스에 지원되는 DB 엔진 섹션을 참조하세요.

- 세션에 이미 저장된 계획은 영향을 받지 않음 - 쿼리 계획 관리는 애플리케이션 코드를 변경하지 않고도 쿼리 계획에 영향을 줄 수 있는 방법을 제공합니다. 그러나 일반 계획이 이미 기존 세션에 저장되어 있는 경우 해당 쿼리 계획을 변경하려면 먼저 DB 클러스터 파라미터 그룹에서 `plan_cache_mode`를 `force_custom_plan`으로 설정해야 합니다.
- `apg_plan_mgmt.dba_plans` 및 `pg_stat_statements`의 `queryid`는 다음과 같은 경우 분기될 수 있습니다.
  - 객체는 apg_plan_mgmt.dba_plans에 저장 후 삭제되고 다시 생성됩니다.
  - `apg_plan_mgmt.plans` 테이블을 다른 클러스터에서 가져왔습니다.

Aurora PostgreSQL 쿼리 계획 관리 기능의 다양한 버전에 관한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 Aurora PostgreSQL apg_plan_mgmt 확장 버전을 참조하세요.

쿼리 계획 관리 용어

이 주제에서 사용되는 용어는 다음과 같습니다.

관리형 문

쿼리 계획 관리 상태에서 최적화 프로그램이 캡처하는 SQL 문입니다. 관리형 문에는`apg_plan_mgmt.dba_plans` 보기에 하나 이상의 쿼리 실행 계획이 저장되어 있습니다.

계획 기준

주어진 관리형 문에 대해 승인된 계획 세트입니다. 즉, `dba_plan` 뷰에서 `status` 열에 '승인됨'이 있는 관리형 문에 대한 모든 계획입니다.

계획 기록

주어진 관리형 문에 대해 캡처된 모든 계획의 세트입니다. 계획 기록에는 상태와 관계없이 해당 문에 대해 캡처된 모든 계획이 포함됩니다.

쿼리 계획 회귀

최적화 프로그램이 새 PostgreSQL 버전이나 통계 변경과 같이 데이터베이스에 특정 변경이 있기 전보다 덜 최적의 계획을 선택하는 경우입니다.

Aurora PostgreSQL의 쿼리 계획 관리 버전

쿼리 계획 관리는 현재 사용 가능한 모든 Aurora PostgreSQL 릴리스에서 지원됩니다. 자세한 정보는 Aurora PostgreSQL 릴리스 정보에서 Amazon Aurora PostgreSQL 업데이트 내용의 목록을 참조하세요.

apg_plan_mgmt 확장을 설치하면 Aurora PostgreSQL DB 클러스터에 쿼리 계획 관리 기능이 추가됩니다. PostgreSQL의 버전마다 지원하는 apg_plan_mgmt 확장 버전이 다릅니다. 사용 중인 Aurora PostgreSQL 버전의 최신 릴리스로 쿼리 계획 관리 확장을 업그레이드하는 것이 좋습니다.

> **Note**
>
> 각 apg_plan_mgmt 확장 버전에 대한 릴리스 정보는 Aurora PostgreSQL 릴리스 정보에서 Aurora PostgreSQL apg_plan_mgmt extension versions(Aurora PostgreSQL apg_plan_mgmt 확장 버전)를 참조하세요.

인스턴스에 연결하고 psql 및 metaccommand \dx를 사용하여 아래와 같이 확장을 나열하면 클러스터에서 실행 중인 버전을 확인할 수 있습니다.

```
labdb=> \dx
                       List of installed extensions
     Name       | Version |     Schema      |                            Description
---------------+---------+---------------
+------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

출력은 이 클러스터가 1.0 버전의 확장을 사용하고 있음을 보여줍니다. 특정 Aurora PostgreSQL 버전에서는 특정 apg_plan_mgmt 버전만 사용할 수 있습니다. 경우에 따라 최신 버전의 쿼리 계획 관리로 업그레이드할 수 있도록 Aurora PostgreSQL DB 클러스터를 새로운 마이너 릴리스로 업그레이드하거나 패치를 적용해야 할 수 있습니다. 출력에 표시된 apg_plan_mgmt 버전 1.0은 Aurora PostgreSQL 버전 10.17 DB 클러스터에서 가져온 것으로, 최신 apg_plan_mgmt 버전은 사용할 수 없습니다. 이 경우 Aurora PostgreSQL DB 클러스터를 더 이후 버전의 PostgreSQL로 업그레이드해야 합니다.

Aurora PostgreSQL DB 클러스터를 새로운 버전의 PostgreSQL로 업그레이드하는 것에 대한 자세한 내용은 Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트 섹션을 참조하세요.

apg_plan_mgmt 확장을 업그레이드하는 방법을 알아보려면 Aurora PostgreSQL의 쿼리 계획 관리 섹션을 참조하세요.

Aurora PostgreSQL 쿼리 계획 관리 활성화

Aurora PostgreSQL DB 클러스터의 쿼리 계획 관리를 설정하려면 확장을 설치하고 여러 DB 클러스터 파라미터 설정을 변경해야 합니다. apg_plan_mgmt 확장을 설치하고 Aurora PostgreSQL DB 클러스터의 기능을 활성화하려면 rds_superuser 권한이 필요합니다.

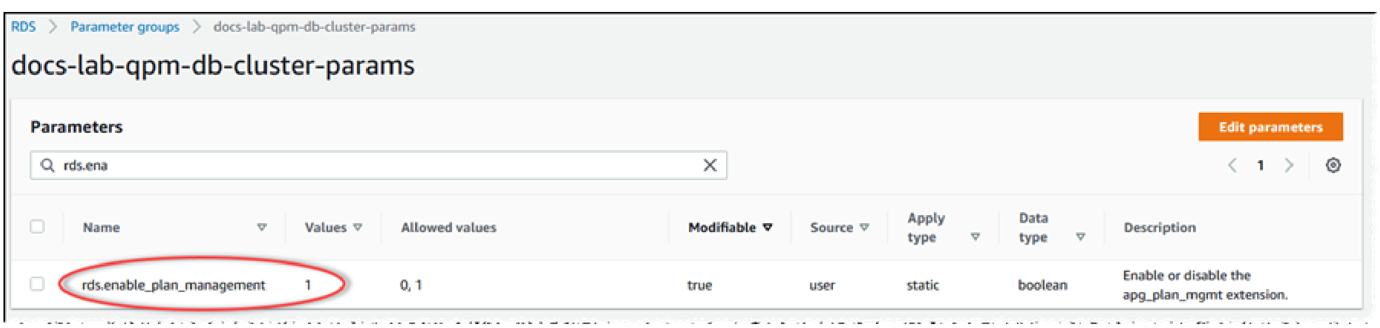
확장을 설치하면 apg_plan_mgmt라는 새 역할이 생성됩니다. 이 역할을 통해 데이터베이스 사용자는 쿼리 계획을 보고, 관리하고, 유지 관리할 수 있습니다. rds_superuser 권한이 있는 관리자는 필요에 따라 데이터베이스 사용자에게 apg_plan_mgmt 역할을 부여해야 합니다.

rds_superuser 역할을 가진 사용자만 다음 절차를 완료할 수 있습니다. rds_superuser 확장 및 해당 apg_plan_mgmt 역할을 생성하는 데 apg_plan_mgmt이(가) 필요합니다. 사용자가 apg_plan_mgmt 확장을 관리하려면 apg_plan_mgmt 역할을 부여받아야 합니다.

Aurora PostgreSQL DB 클러스터에 쿼리 계획 관리를 활성화하는 방법

다음은 Aurora PostgreSQL DB 클러스터로 제출되는 모든 SQL 문에 대해 쿼리 계획 관리를 활성화하는 단계입니다. 이를 자동 모드라고 합니다. 모드 간의 차이에 대한 자세한 내용은 Aurora PostgreSQL 실행 계획 캡처 섹션을 참조하세요.

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Aurora PostgreSQL DB 클러스터에 사용할 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 쿼리 계획 관리를 활성화하고 동작을 설정하려면 특정 파라미터를 변경해야 합니다. 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹 생성 섹션을 참조하세요.
3. 사용자 지정 DB 클러스터 파라미터 그룹을 열고 다음 이미지에 표시된 대로 rds.enable_plan_management 파라미터를 1로 설정합니다.



자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

4. 인스턴스 수준에서 쿼리 계획 파라미터를 설정하는 데 사용할 수 있는 사용자 지정 DB 파라미터 그룹을 생성합니다. 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹 생성 섹션을 참조하세요.
5. Aurora PostgreSQL DB 클러스터의 쓰기 인스턴스를 수정하여 사용자 지정 DB 파라미터 그룹을 사용하도록 합니다. 자세한 내용은 DB 클러스터에서 DB 인스턴스 수정 섹션을 참조하세요.
6. Aurora PostgreSQL DB 클러스터를 수정하여 사용자 지정 DB 파라미터 그룹을 사용하도록 합니다. 자세한 내용은 콘솔, CLI, API를 사용하여 DB 클러스터 수정 섹션을 참조하세요.
7. DB 인스턴스를 재부팅하여 사용자 지정 파라미터 그룹 설정을 활성화합니다.
8. `psql` 또는 `pgAdmin`을 사용하여 Aurora PostgreSQL DB 클러스터의 DB 인스턴스 엔드포인트에 연결합니다. 다음 예에서는 `rds_superuser` 역할에 기본 `postgres` 계정을 사용합니다.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. DB 인스턴스에 다음과 같이 `apg_plan_mgmt` 확장을 생성합니다.

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> 애플리케이션의 템플릿 데이터베이스에 `apg_plan_mgmt` 확장을 설치합니다. 기본 템플릿 데이터베이스의 이름은 `template1`입니다. 자세한 내용은 PostgreSQL 설명서의 Template Databases(템플릿 데이터베이스)를 참조하세요.

10. `apg_plan_mgmt.capture_plan_baselines` 파라미터를 `automatic`으로 변경합니다. 이 설정을 사용하면 옵티마이저는 계획되거나 두 번 이상 실행되는 모든 SQL 문에 대한 계획을 생성합니다.

> **Note**
>
> 쿼리 계획 관리에는 특정 SQL 문에 사용할 수 있는 수동 모드도 있습니다. 자세한 내용은 Aurora PostgreSQL 실행 계획 캡처를 참조하세요.

11. `apg_plan_mgmt.use_plan_baselines` 파라미터의 값을 '켜기'로 변경합니다. 이 파라미터를 사용하면 최적화 프로그램이 계획 기준에서 문에 대한 계획을 선택합니다. 자세한 내용은 Aurora PostgreSQL 관리형 계획 사용를 참조하세요.

> **Note**
>
> 인스턴스를 재부팅하지 않고도 세션의 이러한 동적 파라미터 값을 수정할 수 있습니다.

쿼리 계획 관리 설정이 완료되면 쿼리 계획을 보거나 관리하거나 유지 관리해야 하는 모든 데이터베이스 사용자에게 `apg_plan_mgmt` 역할을 부여해야 합니다.

Aurora PostgreSQL의 쿼리 계획 관리

사용 중인 Aurora PostgreSQL 버전의 최신 릴리스로 쿼리 계획 관리 확장을 업그레이드하는 것이 좋습니다.

1. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 `rds_superuser` 권한을 가진 사용자로 연결합니다. 인스턴스를 설정할 때 기본 이름을 유지했다면 `postgres`로 연결합니다. 이 예에서는 `psql` 사용 방법을 보여주지만 원하는 경우 pgAdmin을 사용할 수도 있습니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 다음 쿼리를 실행하여 확장을 업그레이드합니다.

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
```

3. apg_plan_mgmt.validate_plans 함수를 사용하여 모든 계획의 해시를 업데이트합니다. 최적화 프로그램은 모든 '승인됨', '승인되지 않음, '거부됨' 상태의 계획을 검증하여 확장의 새 버전에서 실행 가능한 계획인지 확인합니다.

```
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
```

   이 함수 사용에 대한 자세한 내용은 계획 검증 섹션을 참조하세요.

4. apg_plan_mgmt.reload 함수를 사용하여 공유 메모리의 모든 계획을 dba_plan 보기에서 검증된 계획으로 새로 고칩니다.

```
SELECT apg_plan_mgmt.reload();
```

쿼리 계획 관리에 사용할 수 있는 모든 함수에 대한 자세한 내용은 Aurora PostgreSQL 쿼리 계획 관리를 위한 함수 참조 섹션을 참조하세요.

Aurora PostgreSQL 쿼리 계획 관리 비활성화

언제든지 `apg_plan_mgmt.use_plan_baselines` 및 `apg_plan_mgmt.capture_plan_baselines`를 비활성화하여 쿼리 계획 관리를 비활성화할 수 있습니다.

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Aurora PostgreSQL 쿼리 계획 관리에 대한 모범 사례

쿼리 계획 관리를 사용하면 쿼리 실행 계획을 변경하는 방식과 변경해야 하는 경우를 제어할 수 있습니다. DBA로서 QPM을 사용할 때의 주요 목표는 데이터베이스에 변경 사항이 있을 때 역행을 방지하고, 옵티마이저의 새 계획 사용 여부를 제어하는 것입니다. 아래에서 쿼리 계획 관리를 사용하는 데 권장되는 모범 사례를 확인할 수 있습니다. 사전 대비형 계획 관리와 사후 대응형 계획 관리는 새 계획의 사용이 승인되는 방식과 시점이 다릅니다.

목차

성능 역행 문제를 방지하기 위한 사전 대비형 계획 관리

계획 성능 회귀가 발생하지 않게 하려면, 새로 검색된 계획의 성능을 승인된 계획의 기존 기준 성능과 비교한 다음 가장 빠른 계획 세트를 새 기준으로 자동 승인하는 프로시저를 실행하여 계획 베이스라인을 개선해야 합니다. 이렇게 하면 시간이 지남에 따라 더 빠른 계획이 검색되어 계획 기준이 개선됩니다.

1. 개발 환경에서 성능 또는 시스템 처리량에 가장 큰 영향을 미치는 SQL 문을 식별합니다. 그런 다음 특정 SQL 문에 대해 계획을 수동으로 캡처 및 계획 자동 캡처에 설명된 대로 이러한 설명문에 대해 계획을 캡처합니다.
2. 개발 환경에서 캡처한 계획을 내보내고 프로덕션 환경으로 가져옵니다. 자세한 내용은 Aurora PostgreSQL용 관리형 계획 내보내기 및 가져오기 섹션을 참조하세요.

3. 프로덕션 단계에서 애플리케이션을 실행하고 승인된 관리형 계획 사용을 적용합니다. 자세한 내용은 Aurora PostgreSQL 관리형 계획 사용 섹션을 참조하세요. 애플리케이션이 실행되는 동안 최적화 프로그램에서 검색되는 새 계획도 추가합니다. 자세한 내용은 계획 자동 캡처 섹션을 참조하세요.
4. 미승인 계획을 분석하고 잘 수행되는 계획을 승인됩니다. 자세한 내용은 계획 성능 평가 섹션을 참조하세요.
5. 애플리케이션이 계속 실행되는 동안 최적화 프로그램은 해당되는 경우 새 계획을 사용하기 시작합니다.

메이저 버전 업그레이드 후 계획 안정성 보장

PostgreSQL의 각 메이저 버전에는 성능 개선을 위해 설계된 쿼리 옵티마이저의 개선 사항 및 변경 사항이 포함되어 있습니다. 하지만 이전 버전에서 옵티마이저가 생성한 쿼리 실행 계획은 새로 업그레이드된 버전에서 성능 회귀를 일으킬 수 있습니다. 쿼리 계획 관리를 사용하여 이 성능 문제를 해결하고 메이저 버전 업그레이드 후 계획 안정성을 보장할 수 있습니다.

최적화 프로그램은 동일한 문에 대해 승인된 계획이 두 개 이상인 경우에도 항상 최소 비용의 승인된 계획을 사용합니다. 업그레이드 후 최적화 프로그램이 새 계획을 발견할 수 있지만 이러한 계획은 승인되지 않은 계획으로 저장됩니다. 이러한 계획은 unapproved_plan_execution_threshold 파라미터와 함께 반응형 계획 관리 스타일을 사용하여 승인된 경우에만 수행됩니다. evolve_plan_baselines 파라미터와 함께 선제적 스타일의 계획 관리를 사용하면 계획 안정성을 극대화할 수 있습니다. 이는 새 계획의 성과를 기존 계획과 비교하여 차선책보다 10% 이상 빠른 계획을 승인하거나 거부합니다.

업그레이드 후 `evolve_plan_baselines` 함수를 사용하여 쿼리 매개 변수 바인딩을 사용한 업그레이드 전후의 계획 성능을 비교할 수 있습니다. 다음 단계에서는 Aurora PostgreSQL 관리형 계획 사용에 설명된 대로 프로덕션 환경에서 승인된 관리형 계획을 사용하고 있다고 가정합니다.

1. 업그레이드하기 전에 쿼리 계획 관리자가 실행 중인 상태에서 애플리케이션을 실행합니다. 애플리케이션이 실행되는 동안 옵티마이저에서 검색되는 새 계획을 추가합니다. 자세한 내용은 계획 자동 캡처 섹션을 참조하세요.
2. 각 계획의 성능을 평가합니다. 자세한 내용은 계획 성능 평가 섹션을 참조하세요.
3. 업그레이드 후 `evolve_plan_baselines` 함수를 사용하여 승인된 계획을 다시 분석합니다. 쿼리 매개 변수 바인딩을 사용하기 전후의 성능을 비교합니다. 새 플랜이 빠른 경우 승인된 계획에 추가하면 됩니다. 동일한 매개 변수 바인딩에 대한 다른 계획보다 빠르면 느린 계획을 거부됨(Rejected)으로 표시할 수 있습니다.

   자세한 내용은 더 나은 계획 승인 섹션을 참조하세요. 이 함수에 대한 자세한 내용은 apg_plan_mgmt.evolve_plan_baselines 단원을 참조하세요.

자세한 내용은 메이저 버전 업그레이드 후 Amazon Aurora PostgreSQL 호환 버전 쿼리 계획 관리를 통해 일관된 성능 보장을 참조하세요.

> **Note**
>
> 논리적 복제 또는 AWS DMS를 사용하여 메이저 버전 업그레이드를 수행하는 경우 기존 계획이 업그레이드된 인스턴스에 복사되도록 `apg_plan_mgmt` 스키마를 복제해야 합니다. 논리적 복제에 대한 자세한 내용은 논리적 복제를 사용하여 Aurora PostgreSQL에 대한 메이저 버전 업그레이드 수행 섹션을 참조하세요.

성능 역행을 찾아내어 복구하기 위한 사후 대응형 계획 관리

애플리케이션이 실행될 때 애플리케이션을 모니터링하여 성능 역행을 발생시키는 계획을 찾아냅니다. 역행을 찾아내면 다음 단계를 따라 좋지 않은 계획을 수동으로 거부하거나 수정합니다.

1. 애플리케이션이 실행되는 동안 관리형 계획의 사용을 적용하고 새로 검색된 계획을 미승인 계획으로 자동 추가합니다. 자세한 내용은 Aurora PostgreSQL 관리형 계획 사용 및 계획 자동 캡처 단원을 참조하십시오.
2. 실행 중인 애플리케이션에서 성능 역행 문제가 있는지 모니터링합니다.
3. 계획 역행 문제를 발견하면 계획의 상태를 `rejected`로 설정합니다. 최적화 프로그램은 다음 번에 SQL 문을 실행할 때 거부된 계획을 자동으로 무시하고 대신에 승인된 다른 계획을 사용합니다. 자세한 내용은 느린 계획 거부 또는 비활성화 섹션을 참조하세요.

   경우에 따라 좋지 않은 계획을 거부, 비활성화 또는 삭제하기 보다는 수정하는 것이 더 나을 수도 있습니다. `pg_hint_plan` 확장을 사용하여 계획 개선을 실습해 봅니다. `pg_hint_plan`을 통해 특별 설명을 사용하여 계획의 정상 생성 방식을 재정의하도록 최적화 프로그램에 지정합니다. 자세한 내용은 pg_hint_plan을 사용하여 계획 수정 섹션을 참조하세요.

Aurora PostgreSQL의 쿼리 계획 관리

Aurora PostgreSQL DB 클러스터에 쿼리 계획 관리를 활성화하면 최적화 프로그램이 두 번 이상 처리하는 모든 SQL 문에 대한 쿼리 실행 계획을 생성하고 저장합니다. 최적화 프로그램은 관리형 문에 대해 생성된 첫 번째 계획의 상태를 항상 `Approved`로 설정하고 `dba_plans` 보기에 저장합니다.

관리형 문에 대해 저장된 승인된 계획의 세트를 계획 기준이라고 합니다. 애플리케이션이 실행될 때 최적화 프로그램은 관리형 문에 대한 추가 계획을 생성할 수 있습니다. 최적화 프로그램은 캡처한 추가 계획을 `Unapproved` 상태로 설정합니다.

이후에 Unapproved 계획이 잘 작동하는지 판단하고 이 계획을 Approved, Rejected 또는 Preferred로 변경할 수 있습니다. 그러기 위해서는 apg_plan_mgmt.evolve_plan_baselines 함수 또는 apg_plan_mgmt.set_plan_status 함수를 사용합니다.

최적화 프로그램이 SQL 문에 대한 계획을 생성할 때 쿼리 계획 관리는 이 계획을 apg_plan_mgmt.plans 테이블에 저장합니다. apg_plan_mgmt 역할이 부여된 데이터베이스 사용자는 apg_plan_mgmt.dba_plans 보기를 쿼리하여 계획 세부 정보를 볼 수 있습니다. 예를 들어, 다음 쿼리는 비프로덕션 Aurora PostgreSQL DB 클러스터의 현재 보기에 있는 계획의 세부 정보를 나열합니다.

- sql_hash - SQL 문의 식별자로, SQL 문이 정규화된 텍스트의 해시 값입니다.
- plan_hash - 계획의 고유 식별자로, sql_hash와 계획의 해시를 조합한 값입니다.
- status – 계획의 상태입니다. 최적화 프로그램은 승인된 계획을 실행할 수 있습니다.
- enabled - 계획이 사용할 준비가 되었는지(true) 아니면 사용할 준비가 되지 않았는지(false)를 나타냅니다.
- plan_outline - 실제 실행 계획을 다시 생성하는 데 사용되는 계획을 표현합니다. 트리 구조의 연산자는 EXPLAIN 출력의 연산자에 매핑됩니다.

이 apg_plan_mgmt.dba_plans 보기에는 계획의 모든 세부 정보(예: 계획이 마지막으로 사용된 날짜)가 포함된 더 많은 열이 있습니다. 자세한 내용은 [Aurora PostgreSQL 호환 에디션의 apg_plan_mgmt.dba_plans 뷰에 대한 참조](#) 섹션을 참조하세요.

정규화 및 SQL 해시

apg_plan_mgmt.dba_plans 보기에서 SQL 해시 값을 사용하여 관리형 문을 식별할 수 있습니다. SQL 해시는 차이(예: 리터럴 값)를 없앤 SQL 문의 정규화된 표현에서 계산됩니다.

각 SQL 문의 정규화 프로세스는 공백과 대/소문자를 보존하므로 SQL 문의 요점을 읽고 이해할 수 있습니다. 정규화는 다음 항목을 제거하거나 대체합니다.

- 주요 블록 설명
- EXPLAIN 키워드와 EXPLAIN 옵션 및 EXPLAIN ANALYZE
- 후행 공백
- 모든 리터럴

다음 문을 예로 들 수 있습니다.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

쿼리 계획 관리는 이 문을 다음과 같이 정규화합니다.

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

정규화는 리터럴 또는 파라미터 값만 다를 수 있는 유사 SQL 문에 대해 동일한 SQL 해시를 사용할 수 있게 합니다. 즉, 동일한 SQL 해시에 대해 서로 다른 조건에서 최적 상태인 서로 다른 계획이 여러 개 있을 수 있습니다.

> **Note**
> 서로 다른 스키마와 함께 사용되는 단일 SQL 문은 런타임에 특정 스키마에 바인딩되므로 다른 계획을 사용합니다. 플래너는 스키마 바인딩에 대한 통계를 사용하여 최적의 계획을 선택합니다.

최적화 프로그램이 계획을 선택하는 방법에 대한 자세한 내용은 Aurora PostgreSQL 관리형 계획 사용 섹션을 참조하세요. 이 섹션에서는 계획을 실제로 사용하기 전에 `EXPLAIN` 및 `EXPLAIN ANALYZE`를 사용하여 계획을 미리 보는 방법을 배울 수 있습니다. 자세한 내용은 최적화 프로그램이 선택한 계획 분석을 참조하세요. 계획 선택 프로세스를 설명하는 이미지는 최적화 프로그램에서 실행할 계획을 선택하는 방식. 섹션을 참조하세요.

Aurora PostgreSQL 실행 계획 캡처

Aurora PostgreSQL 쿼리 계획 관리에서는 쿼리 실행 계획을 캡처하기 위한 두 가지 모드, 자동 또는 수동을 제공합니다. `apg_plan_mgmt.capture_plans_baselines`의 값을 `automatic` 또는 `manual`로 설정하여 모드를 선택합니다. 수동 계획 캡처를 사용해 특정 SQL 문에 대한 실행 계획을 캡처할 수 있습니다. 또는 자동 계획 캡처를 사용하여 애플리케이션이 실행될 때 두 번 이상 실행되는 모든 (또는 가장 느린) 계획을 캡처할 수 있습니다.

계획을 캡처하면 최적화 프로그램이 관리형 설명문에 대해 캡처된 첫 번째 계획의 상태를 `approved`로 설정합니다. 최적화 프로그램은 관리형 설명문에 대해 캡처된 추가 계획의 상태를 항상 `unapproved`로 설정합니다. 그러나 두 가지 이상의 계획이 `approved` 상태로 저장되는 경우가 가끔 있을 수 있습니다. 이런 일은 설명문에 대해 여러 계획이 병렬적으로 생성될 때와 설명문의 첫 번째 계획이 커밋되기 전에 발생할 수 있습니다.

`dba_plans` 보기에 캡처 및 저장할 수 있는 최대 계획 수를 제어하려면 DB 인스턴스 수준의 파라미터 그룹에서 `apg_plan_mgmt.max_plans` 파라미터를 설정합니다. `apg_plan_mgmt.max_plans`

파라미터를 변경하는 경우 새 값을 적용하려면 DB 인스턴스를 재부팅해야 합니다. 자세한 내용은 apg_plan_mgmt.max_plans 파라미터를 참조하세요.

특정 SQL 문에 대해 계획을 수동으로 캡처

관리해야 할 SQL 문 세트를 알고 있는 경우 이러한 설명문을 SQL 스크립트 파일에 추가한 후 계획을 수동으로 캡처합니다. 다음은 SQL 문 세트에 대해 쿼리 계획을 수동으로 캡처하는 방법을 보여주는 psql 예제입니다.

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

각 SQL 문에 대한 계획을 캡처하면 최적화 프로그램이 새 행을 `apg_plan_mgmt.dba_plans` 보기에 추가합니다.

SQL 스크립트 파일에서 EXPLAIN 또는 EXPLAIN EXECUTE 문을 사용하는 것이 좋습니다. 관심이 있는 계획을 모두 캡처하려면 파라미터 값에 충분한 변형을 포함해야 합니다.

최적화 프로그램의 최소 비용 계획보다 더 나은 계획을 알고 있으면 최적화 프로그램에서 더 나은 계획을 사용하도록 강제할 수 있습니다. 이를 위해서는 최적화 프로그램 힌트를 하나 이상 지정해야 합니다. 자세한 내용은 pg_hint_plan을 사용하여 계획 수정 섹션을 참조하세요. `unapproved` 및 `approved` 계획의 성능을 비교하고 이러한 계획을 승인, 거부 또는 삭제하려면 계획 성능 평가 단원을 참조하십시오.

계획 자동 캡처

다음과 같은 상황에서는 자동 계획 캡처를 사용합니다.

- 관리할 특정 SQL 문을 모르는 경우
- 관리할 SQL 문이 수백 개 또는 수천 개인 경우
- 애플리케이션에서 클라이언트 API를 사용하는 경우 예를 들면, JDBC는 psql로 표현할 수 없는 이름이 지정되지 않은 준비된 명령문이나 대량 모드 명령문을 사용합니다.

계획을 자동으로 캡처하려면

1. DB 인스턴스 수준의 파라미터 그룹에서 `apg_plan_mgmt.capture_plan_baselines`를 `automatic`으로 설정하여 자동 계획 캡처를 켭니다. 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

2. 애플리케이션이 실행되면 최적화 프로그램이 두 번 이상 실행되는 각 SQL 문에 대해 계획을 캡처합니다.

   애플리케이션이 기본 쿼리 계획 관리 파라미터 설정으로 실행되면 최적화 프로그램이 두 번 이상 실행되는 각 SQL 문에 대해 계획을 캡처합니다. 기본값 사용 중 모든 계획을 캡처하는 작업은 실행 시간 오버헤드가 거의 발생하지 않으며 프로덕션 상태에서 활성화할 수 있습니다.

자동 계획 캡처를 끄려면

- DB 인스턴스 수준의 파라미터 그룹에서 `apg_plan_mgmt.capture_plan_baselines` 파라미터를 `off`로 설정합니다.

미승인 계획의 성능을 측정하고 이러한 계획을 승인, 거부 또는 삭제하려면 계획 성능 평가 단원을 참조하세요.

Aurora PostgreSQL 관리형 계획 사용

최적화 프로그램에서 관리형 설명문에 대해 캡처한 계획을 사용하려면 파라미터 apg_plan_mgmt.use_plan_baselines를 true로 설정합니다. 다음은 로컬 인스턴스 예제입니다.

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

애플리케이션이 실행되는 동안 이 설정을 사용하면 최적화 프로그램이 각각의 관리형 문에 대해 유효하고 활성화된 최소 비용, 기본 또는 승인된 계획을 사용합니다.

최적화 프로그램이 선택한 계획 분석

apg_plan_mgmt.use_plan_baselines 파라미터가 true로 설정된 경우 EXPLAIN ANALYZE SQL 문을 사용하여 최적화 프로그램에서 해당 설명문을 실행해야 하는 경우에 사용할 계획을 표시할 수 있습니다. 다음은 예입니다.

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                    QUERY PLAN
--------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

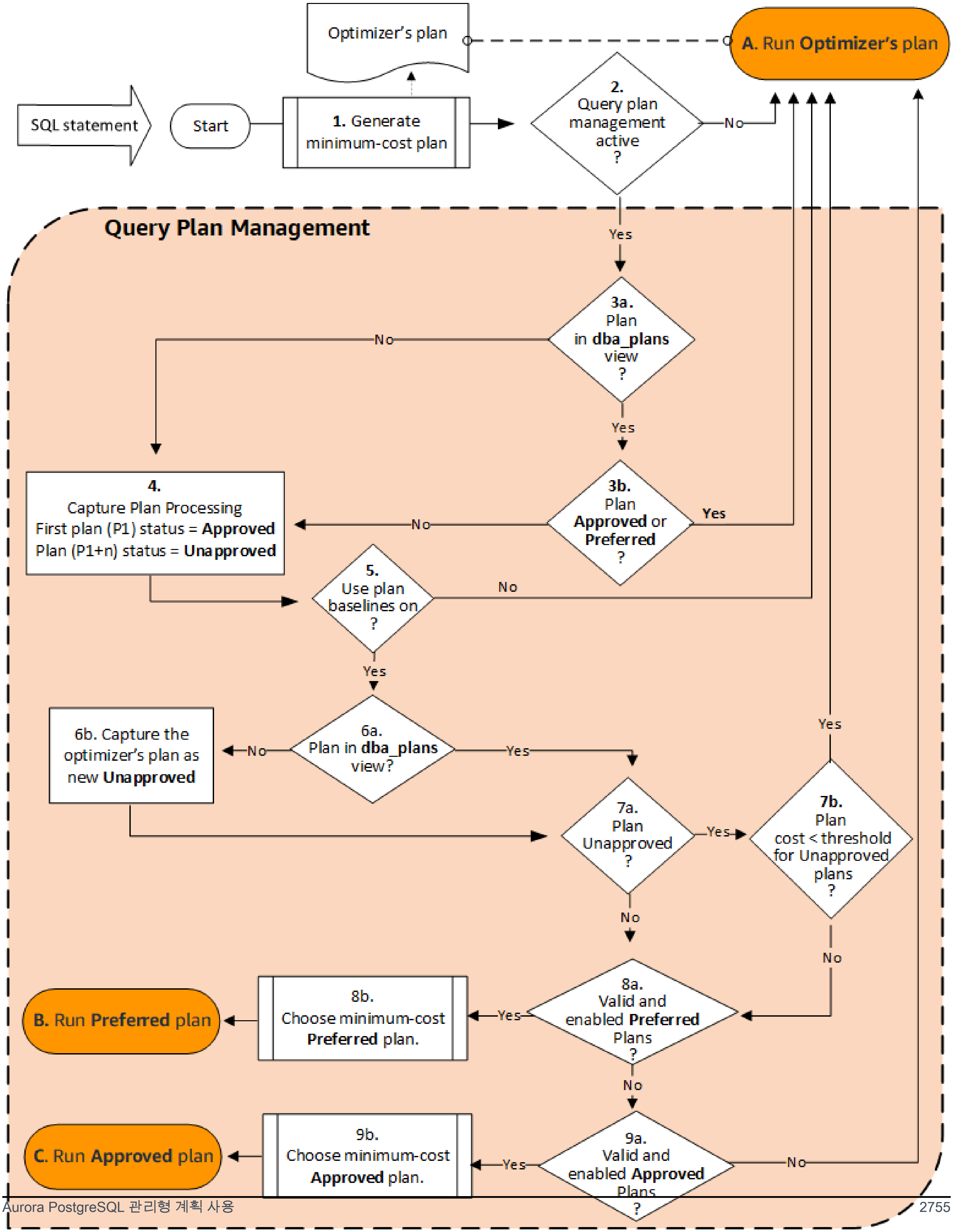
출력에는 실행할 기준의 승인된 계획이 표시됩니다. 그러나 출력에 비용이 더 저렴한 계획을 찾았다고 나와 있습니다. 이 경우, 계획 자동 캡처에 설명된 바와 같이 자동 계획 캡처를 켜서 이 새로운 최소 비용 계획을 캡처합니다.

새 계획은 항상 최적화 프로그램에 Unapproved로 캡처됩니다. apg_plan_mgmt.evolve_plan_baselines 함수를 사용하여 계획을 비교하고 승인됨, 거부됨 또는 비활성화됨으로 변경할 수 있습니다. 자세한 내용은 계획 성능 평가 섹션을 참조하세요.

최적화 프로그램에서 실행할 계획을 선택하는 방식.

실행 계획의 비용은 최적화 프로그램에서 서로 다른 계획을 비교할 때 계산하는 추정치입니다. 계획의 비용을 계산할 때 최적화 프로그램은 해당 계획에 필요한 CPU 및 I/O 작업과 같은 요소를 포함합니다. PostgreSQL 쿼리 플래너 비용 견적에 대해 자세히 알아보려면 쿼리 계획을 참조하세요.

다음 이미지는 쿼리 계획 관리가 활성 상태일 때와 그렇지 않을 때 주어진 SQL 문에 대해 계획이 선택되는 방법을 보여줍니다.


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Yes
Query Plan Management
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
Yes
No
5. Use plan baselines on ?
No
Yes
6b. Capture the optimizer's plan as new Unapproved
6a. Plan in dba_plans view?
No
Yes
7a. Plan Unapproved ?
Yes
No
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
B. Run Preferred plan
8b. Choose minimum-cost Preferred plan.
8a. Valid and enabled Preferred Plans ?
Yes
No
C. Run Approved plan
9b. Choose minimum-cost Approved plan.
9a. Valid and enabled Approved Plans ?
Yes
No

이 흐름은 다음과 같습니다.

1. 최적화 프로그램은 SQL 문에 대한 최소 비용 계획을 생성합니다.
2. 쿼리 계획 관리가 활성화되어 있지 않으면 최적화 프로그램의 계획(A. Run Optimizer's plan)이 즉시 실행됩니다. 쿼리 계획 관리는 및 매개변수가 모두 기본 설정(각각 "off" 및 "false")일 때 비활성화됩니다.

   그렇지 않으면 쿼리 계획 관리가 활성화됩니다. 이 경우 SQL 문과 이에 대한 최적화 프로그램의 계획은 계획이 선택되기 전에 추가로 평가됩니다.

   > **Tip**
   >
   > `apg_plan_mgmt` 역할이 있는 데이터베이스 사용자는 사전에 계획을 비교하고, 계획의 상태를 변경하고, 필요에 따라 특정 계획을 강제로 사용할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL 쿼리 계획 개선](#) 섹션을 참조하세요.

3. SQL 문에는 이전에 쿼리 계획 관리에 의해 저장되었던 계획이 이미 있을 수 있습니다. 계획은 계획을 생성하는 데 사용된 SQL 문에 대한 정보와 함께 `apg_plan_mgmt.dba_plans`에 저장됩니다. 계획에 대한 정보에는 해당 상태가 포함됩니다. 계획 상태에 따라 다음과 같이 사용 여부가 결정될 수 있습니다.
   a. 계획이 SQL 문에 대해 저장된 계획에 없으면 주어진 SQL 문에 대한 최적화 프로그램에서 이 특정 계획이 처음 생성되었음을 의미합니다. 계획이 캡처 계획 처리 (4)로 전송됩니다.
   b. 저장된 계획 중 해당 계획이 승인됨 또는 선호되면 계획(A. Run Optimizer's plan)이 실행됩니다.

      계획이 저장된 계획에 있지만 승인 또는 선호도가 아닌 경우 계획은 캡처 계획 처리(4)로 전송됩니다.
4. 주어진 SQL 문에 대해 처음으로 계획을 캡처할 때 계획의 상태는 항상 승인됨(P1)으로 설정됩니다. 최적화 프로그램은 이후에 동일한 SQL 문에 대해 동일한 계획을 생성하면 해당 계획의 상태가 미승인(P1+n)으로 변경됩니다.

   계획이 캡처되고 상태가 업데이트되면 평가가 다음 단계(5)에서 계속됩니다.
5. 계획의 기준선은 SQL 문의 기록과 다양한 상태에서의 계획으로 구성됩니다. 쿼리 계획 관리에서는 계획을 선택할 때 다음과 같이 계획 기준선 사용 옵션의 설정 여부에 따라 기준을 고려할 수 있습니다.
   - `apg_plan_mgmt.use_plan_baselines` 파라미터가 기본값(`false`)으로 설정된 경우 계획 기준선 사용은 'off'입니다. 계획은 실행되기 전에 기준선과 비교되지 않습니다(A. Run Optimizer의 계획).

- `apg_plan_mgmt.use_plan_baselines` 파라미터가 `true`로 설정된 경우 계획 기준선 사용이 'on'입니다. 계획은 기준선(6)을 사용하여 추가로 평가됩니다.

6. 계획은 기준선의 명령문에 대한 다른 계획과 비교됩니다.

   a. 최적화 프로그램의 계획이 베이스라인에 있는 계획들 중이라면 그 상태를 체크한다(7a).

   b. 최적화 프로그램의 계획이 기준선의 계획에 없는 경우 해당 계획은 명령문의 계획에 새 `Unapproved` 계획으로 추가됩니다.

7. 계획의 상태는 승인되지 않은 경우에만 확인하기 위해 확인됩니다.

   a. 계획의 상태가 미승인인 경우 계획의 예상 비용은 승인되지 않은 실행 계획 임계값에 대해 지정된 예상 비용과 비교됩니다.

      - 계획의 예상 비용이 임계값 미만인 경우 미승인 계획(A. Run Optimizer의 계획)이라도 최적화 프로그램은 이를 사용합니다. 일반적으로 최적화 프로그램은 미승인 계획을 실행하지 않습니다. 그러나 파라미터가 비용 임계값을 지정하면 최적화 프로그램은 승인되지 않은 계획의 비용을 임계값과 비교합니다. 예상 비용이 임계값보다 작으면 최적화 프로그램이 계획을 실행합니다. 자세한 내용은 apg_plan_mgmt.unapproved_plan_execution_threshold 섹션을 참조하세요.
      - 계획의 예상 비용이 임계값 미만이 아니면 계획의 다른 속성이 확인됩니다(8a).

   b. 계획의 상태가 미승인 상태가 아닌 경우 다른 속성이 확인됩니다(8a).

8. 최적화 프로그램은 비활성화된 계획을 사용하지 않습니다. 즉, `enable` 속성이 'f'(false)로 설정된 계획입니다. 또한 최적화 프로그램은 상태가 거부됨인 계획을 사용하지 않습니다.

   최적화 프로그램은 유효하지 않은 계획을 사용할 수 없습니다. 인덱스 및 테이블 파티션과 같이 계획이 의존하는 개체가 제거되거나 삭제되면 계획이 유효하지 않게 될 수 있습니다.

   a. 명령문에 활성화되고 유효한 선호 계획이 있는 경우 최적화 프로그램은 이 SQL 문에 대해 저장된 선호 계획 중에서 최소 비용 계획을 선택합니다. 그런 다음 최적화 프로그램은 최소 비용 선호 계획을 실행합니다.

   b. 명령문에 활성화되고 유효한 기본 계획이 없으면 다음 단계(9)에서 평가됩니다.

9. 명령문에 활성화되고 유효한 승인됨 계획이 있는 경우 최적화 프로그램은 이 SQL 문에 대해 저장된 승인됨 계획 중에서 최소 비용 계획을 선택합니다. 그런 다음 최적화 프로그램은 최소 비용 승인된 계획을 실행합니다.

   명령문에 유효하고 활성화된 승인된 계획이 없으면 최적화 프로그램은 최소 비용 계획을 사용합니다(A. 옵티마이저의 계획 실행).

dba_plans 보기에서 Aurora PostgreSQL 쿼리 계획 검사

apg_plan_mgmt 역할이 부여된 데이터베이스 사용자 및 관리자는 apg_plan_mgmt.dba_plans에 저장된 계획을 보고 관리할 수 있습니다. Aurora PostgreSQL DB 클러스터의 관리자 (rds_superuser 권한이 있는 사람)는 쿼리 계획 관리를 사용해야 하는 데이터베이스 사용자에게 이 역할을 명시적으로 부여해야 합니다.

apg_plan_mgmt 보기에는 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 있는 모든 데이터베이스의 모든 관리형 SQL 문에 대한 계획 기록이 포함됩니다. 이 보기를 통해 계획, 상태, 마지막 사용 시점 및 기타 모든 관련 세부 정보를 검토할 수 있습니다.

정규화 및 SQL 해시에서 설명한 대로 각 관리형 계획은 SQL 해시 값과 계획 해시 값의 조합으로 식별됩니다. 이러한 식별자를 사용하면 Amazon RDS 성능 개선 도우미와 같은 도구를 사용하여 개별 계획 성능을 추적할 수 있습니다. 성능 개선 도우미에 대한 자세한 내용은 Amazon RDS 성능 개선 도우미 사용 단원을 참조하세요.

관리형 계획 나열

관리형 계획을 나열하려면 apg_plan_mgmt.dba_plans 보기에서 SELECT 문을 사용합니다. 다음 예제는 dba_plans(승인 및 미승인 계획을 나타냄)와 같은 status 보기의 일부 열을 보여줍니다.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash    | plan_hash |   status    | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

읽기 쉽게 하기 위해, 표시된 쿼리와 출력에는 dba_plans 보기에 있는 몇 개의 열만 나열되어 있습니다. 전체 정보는 Aurora PostgreSQL 호환 에디션의 apg_plan_mgmt.dba_plans 뷰에 대한 참조 섹션을 참조하세요.

Aurora PostgreSQL 쿼리 계획 개선

계획 성과를 평가하고 계획을 수정하여 쿼리 계획 관리를 개선합니다. 쿼리 계획을 개선하는 방법에 대한 자세한 내용은 다음 주제를 참조하세요.

주제

- [계획 성능 평가]
- [pg_hint_plan을 사용하여 계획 수정]

계획 성능 평가

최적화 프로그램이 계획을 미승인 계획으로 캡처한 후 apg_plan_mgmt.evolve_plan_baselines 함수를 사용하여 실제 성능에 따라 계획을 비교합니다. 성능 실험의 결과에 따라 승인되지 않음에서 승인됨 또는 거부됨으로 계획의 상태를 변경할 수 있습니다. 대신에 apg_plan_mgmt.evolve_plan_baselines 함수를 사용하여 요구 사항을 충족하지 않을 경우 계획을 일시적으로 비활성화하도록 결정할 수 있습니다.

더 나은 계획 승인

다음 예제에서는 apg_plan_mgmt.evolve_plan_baselines 함수를 사용하여 관리형 계획의 상태를 승인됨으로 변경하는 방법을 보여줍니다.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
-----------------------
0
(1 row)
```

출력에 파라미터 바인딩이 1과 10,000인 rangequery 문에 대한 성능 보고서가 표시됩니다. 새로운 미승인 계획(Baseline+1)이 이전에 승인된 계획 최상의 계획(Baseline)보다 더 낫습니다. 새 계획이 Approved 상태인지 확인하려면 apg_plan_mgmt.dba_plans 보기를 확인합니다.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash   | plan_hash |  status   | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

이제 관리형 계획에 설명문의 계획 기준인 승인된 계획 두 개가 포함됩니다. 또한 `apg_plan_mgmt.set_plan_status` 함수를 호출하여 계획의 상태 필드를 `'Approved'`, `'Rejected'`, `'Unapproved'` 또는 `'Preferred'`로 직접 설정할 수도 있습니다.

느린 계획 거부 또는 비활성화

계획을 거부하거나 비활성화하려면 `'reject'` 또는 `'disable'` 을 `apg_plan_mgmt.evolve_plan_baselines` 함수에 작업 파라미터로 전달합니다. 이 예제에서는 해당 문에 대한 최상의 `Unapproved` 계획보다 최소 10% 더 느린 캡처된 `Approved` 계획을 비활성화 합니다.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

또한 계획을 거부됨 또는 비활성화됨으로 즉시 설정할 수도 있습니다. 계획의 활성화 필드를 `true` 또는 `false`로 즉시 설정하려면 `apg_plan_mgmt.set_plan_enabled` 함수를 호출합니다. 계획의 상태 필드를 `'Approved'`, `'Rejected'`, `'Unapproved'` 또는 `'Preferred'`로 즉시 설정하려면 `apg_plan_mgmt.set_plan_status` 함수를 호출합니다.

현재 유효하지 않고 앞으로도 유효하지 않을 것으로 예상되는 계획을 삭제하려면 `apg_plan_mgmt.validate_plans` 함수를 사용하세요. 이 함수를 사용하면 유효하지 않은 계획을 삭제하거나 비활성화할 수 있습니다. 자세한 내용은 [계획 검증](#) 섹션을 참조하세요.

pg_hint_plan을 사용하여 계획 수정

쿼리 최적화 프로그램은 모든 설명문에 대한 최적 계획을 찾도록 설계되었으며, 대부분의 경우 최적화 프로그램은 좋은 계획을 찾아냅니다. 하지만 경우에 따라 최적화 프로그램에서 생성된 것보다 훨씬 더 나은 계획이 존재한다는 것을 알게 될 수도 있습니다. 최적화 프로그램을 통해 원하는 계획을 생

성하기 위한 두 가지 권장 방법은 PostgreSQL에서 pg_hint_plan 확장을 사용하거나 Grand Unified Configuration(GUC) 변수를 설정하는 것입니다.

- pg_hint_plan 확장 – PostgreSQL의 pg_hint_plan 확장을 사용하여 플래너의 작동 방식을 수정하려면 "힌트(hint)"를 지정합니다. pg_hint_plan 확장을 설치하고 사용하는 방법은 pg_hint_plan 설명서를 참조하십시오.
- GUC 변수 – 하나 이상의 비용 모델 파라미터 또는 다른 최적화 프로그램 파라미터(예: from_collapse_limit 또는 GEQO_threshold)를 재정의합니다.

이러한 기술 중 하나를 사용하여 쿼리 최적화 프로그램에서 계획을 사용하도록 지정할 경우 쿼리 계획 관리를 사용하여 새 계획을 캡처하고 사용하도록 설정할 수도 있습니다.

pg_hint_plan 확장을 사용하여 SQL 문의 조인 순서, 조인 방법 또는 액세스 경로를 변경할 수 있습니다. 특수 pg_hint_plan 구문과 함께 SQL 설명을 사용하여 최적화 프로그램에서 계획을 생성하는 방식을 수정할 수 있습니다. 예를 들어 문제의 SQL 문에 양방향 조인이 있다고 가정해 보겠습니다.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

또한 최적화 프로그램에서는 조인 순서(t1, t2)를 선택하지만 조인 순서(t2, t1)가 더 빠름을 알고 있다고 가정해 보겠습니다. 다음 힌트는 최적화 프로그램에서 더 빠른 조인 순서(t2, t1)를 사용하도록 지정합니다. 최적화 프로그램에서 SQL 문에 대한 계획을 생성하되 해당 문을 실행하지 않도록 EXPLAIN을 포함합니다. (출력이 표시되지 않음)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

다음 단계에서는 pg_hint_plan을 사용하는 방법을 보여줍니다.

최적화 프로그램의 생성된 계획을 수정하고 pg_hint_plan을 사용하여 계획을 캡처하려면

1. 수동 캡처 모드를 켭니다.

```
SET apg_plan_mgmt.capture_plan_baselines = manual;
```

2. 관심 SQL 문에 대한 힌트를 지정합니다.

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

이 실행 후 최적화 프로그램이 apg_plan_mgmt.dba_plans 보기에서 해당 계획을 캡처합니다. 캡처한 계획은 특수 pg_hint_plan 설명 구문을 포함하지 않습니다. 쿼리 계획 관리 기능이 선행 설명을 제거하여 설명문을 정규화하기 때문입니다.

3. apg_plan_mgmt.dba_plans 보기를 사용하여 관리형 계획을 확인합니다.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. 계획의 상태를 Preferred로 설정합니다. 그러면 최적화 프로그램에서 최소 비용 계획이 아직 Approved 또는 Preferred가 아닐 때 승인된 계획 세트에서 선택하는 대신 해당 계획을 실행합니다.

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. 수동 계획 캡처를 끄고 관리형 계획을 사용하도록 설정합니다.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

이제는 원본 SQL 문이 실행될 때 최적화 프로그램이 Approved 또는 Preferred 계획을 선택합니다. 최소 비용 계획이 Approved 또는 Preferred가 아니면 최적화 프로그램은 Preferred 계획을 선택합니다.

Aurora PostgreSQL 쿼리 계획 삭제

사용하지 않는 실행 계획 또는 유효하지 않은 계획을 삭제합니다. 계획 삭제에 대한 자세한 내용은 다음 섹션을 참조하세요.

주제

- 계획 삭제
- 계획 검증

계획 삭제

한 달 이상, 구체적으로 32일 동안 사용하지 않은 계획은 자동으로 삭제됩니다. 이것이 apg_plan_mgmt.plan_retention_period 파라미터의 기본 설정입니다. 값을 1부터 시작해서 계획 보존 기간을 더 긴 기간으로 변경하거나 더 짧은 기간으로 변경할 수 있습니다. 현재 날짜에서 last_used 날짜를 빼서 계획의 마지막으로 사용된 이후 일수를 계산합니다. last_used 날짜는 최적화 프로그램이 계획을 최소 비용 계획으로 선택했거나 계획이 실행된 가장 최근 날짜입니다. 계획의 날짜가 apg_plan_mgmt.dba_plans 보기에 저장됩니다.

장시간 사용하지 않았거나 유용하지 않은 계획을 삭제하는 것이 좋습니다. 최적화 프로그램이 계획을 실행하거나 해당 계획을 문에 대한 최소 비용 계획으로 선택할 때마다 최적화 프로그램이 업데이트하는 last_used 날짜가 각 계획에 지정되어 있습니다. 안전하게 삭제할 수 있는 계획을 확인하려면 마지막 last_used 날짜를 확인하세요.

다음 쿼리는 총 계획 수, 삭제에 실패한 계획 및 성공적으로 삭제된 계획이 포함된 3열 테이블을 반환합니다. 여기에는 apg_plan_mgmt.delete_plan 함수를 사용하여 지난 31일 동안 최소 비용 계획으로 선택되지 않았으며 Rejected 상태가 아닌 모든 계획을 삭제하는 방법의 예인 중첩 쿼리가 있습니다.

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```

 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

자세한 내용은 apg_plan_mgmt.delete_plan 섹션을 참조하세요.

현재 유효하지 않고 앞으로도 유효하지 않을 것으로 예상되는 계획을 삭제하려면 apg_plan_mgmt.validate_plans 함수를 사용하세요. 이 함수를 사용하면 유효하지 않은 계획을 삭제하거나 비활성화할 수 있습니다. 자세한 내용은 계획 검증 섹션을 참조하세요.

> **Important**
>
> 유효하지 않은 계획을 정리하지 않을 경우 결국에는 쿼리 계획 관리용으로 별도 보관되어 있는 공유 메모리가 부족해질 수 있습니다. 관리형 계획에 사용할 수 있는 메모리의 양을 제어하려면 `apg_plan_mgmt.max_plans` 파라미터를 사용합니다. 사용자 지정 DB 파라미터 그룹에서 이 파라미터를 설정하고 DB 인스턴스를 재부팅하여 변경 내용을 적용합니다. 자세한 내용은 [apg_plan_mgmt.max_plans](#) 파라미터를 참조하세요.

계획 검증

`apg_plan_mgmt.validate_plans` 함수를 사용하여 유효하지 않은 계획을 삭제하거나 비활성화할 수 있습니다.

인덱스나 테이블 같은 종속된 객체가 제거되면 계획이 유효하지 않게 되거나 기한 경과 상태가 될 수 있습니다. 하지만 제거된 객체가 다시 생성되는 경우에는 계획이 실시적으로 잘못될 수 있습니다. 유효하지 않은 계획이 나중에 유효 상태가 될 수 있는 경우 유효하지 않은 계획을 삭제하기 보다는 비활성화하거나 아무 작업도 하지 않는 것이 더 나을 수 있습니다.

유효하지 않고 지난 주에 사용한 적이 없는 모든 계획을 찾아서 삭제하려면 다음과 같이 `apg_plan_mgmt.validate_plans` 함수를 사용합니다.

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

계획을 직접 활성화하거나 비활성화하려면 `apg_plan_mgmt.set_plan_enabled` 함수를 사용합니다.

Aurora PostgreSQL용 관리형 계획 내보내기 및 가져오기

관리형 계획을 내보내고 다른 DB 인스턴스로 가져올 수 있습니다.

관리형 계획을 내보내려면

권한 있는 사용자는 `apg_plan_mgmt.plans` 테이블의 일부를 다른 테이블에 복사한 후, pg_dump 명령을 사용하여 저장할 수 있습니다. 다음은 예제입니다.

```
CREATE TABLE plans_copy AS SELECT *
```

```
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

관리형 계획을 가져오려면

1. 내보낸 관리형 계획의 .tar 파일을 계획이 복원되어야 할 시스템으로 복사합니다.
2. `pg_restore` 명령을 사용하여 tar 파일을 새 테이블로 복사합니다.

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```

3. 다음 예제와 같이 `plans_copy` 테이블과 `apg_plan_mgmt.plans` 테이블을 병합합니다.

   > **Note**
   >
   > 어떤 경우에는 한 버전의 `apg_plan_mgmt` 확장에서 덤프하여 다른 버전으로 복원할 수 있습니다. 이 경우 계획 테이블의 열은 다를 수 있습니다. 다른 경우에는 SELECT *를 사용하는 대신에 열에 명시적으로 이름을 지정합니다.

   ```
   INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
    ON CONFLICT ON CONSTRAINT plans_pkey
    DO UPDATE SET
    status = EXCLUDED.status,
    enabled = EXCLUDED.enabled,
    -- Save the most recent last_used date
    --
    last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
    THEN EXCLUDED.last_used ELSE plans.last_used END,
    -- Save statistics gathered by evolve_plan_baselines, if it ran:
    --
    estimated_startup_cost = EXCLUDED.estimated_startup_cost,
    estimated_total_cost = EXCLUDED.estimated_total_cost,
    planning_time_ms = EXCLUDED.planning_time_ms,
    execution_time_ms = EXCLUDED.execution_time_ms,
    total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
    execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
   ```

4. 관리형 계획을 공유 메모리에 다시 로드하고 임시 계획 테이블을 제거합니다.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Aurora PostgreSQL 쿼리 계획 관리를 위한 파라미터 참조

이 섹션에 나열된 파라미터를 사용하여 `apg_plan_mgmt` 확장에 대한 기본 설정을 지정할 수 있습니다 사용자 지정 DB 클러스터 파라미터 및 Aurora PostgreSQL DB 클러스터와 연결된 DB 파라미터 그룹에서 사용할 수 있습니다. 이러한 파라미터는 쿼리 계획 관리 기능의 동작과 이것이 최적화 프로그램에 미치는 영향을 제어합니다. 쿼리 계획 관리를 설정하는 방법에 대한 자세한 내용은 Aurora PostgreSQL 쿼리 계획 관리 활성화 단원을 참조하세요. `apg_plan_mgmt` 확장이 해당 섹션에 설명된 대로 설정되지 않은 경우 다음 파라미터를 변경해도 효과가 없습니다. 파라미터 수정에 대한 자세한 정보는 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 및 Amazon Aurora DB 인스턴스용 DB 파라미터 그룹 단원을 참조하세요.

파라미터

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

각 SQL 문에 대해 최적화 프로그램에서 생성한 쿼리 실행 계획을 캡처하여 `dba_plans` 뷰에 저장합니다. 기본적으로 저장할 수 있는 최대 계획 수는 `apg_plan_mgmt.max_plans` 파라미터에서 지정한 대로 10,000개입니다. 참조 정보는 apg_plan_mgmt.max_plans 단원을 참조하세요.

사용자 지정 DB 클러스터 파라미터 그룹 또는 사용자 지정 DB 파라미터 그룹에서 이 파라미터를 설정할 수 있습니다. 이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
끄기	automatic	세션 수준 또는 파라미터 그룹에서 이 설정을 적용하여 두 번 이상 사용되는 계획을 캡처합니다.
	수동	세션 수준 또는 파라미터 그룹에서 이 설정을 적용하여 한 번 이상 사용되는 계획을 캡처합니다.
	끄기	계획 캡처를 끕니다.

자세한 내용은 Aurora PostgreSQL 실행 계획 캡처 섹션을 참조하세요.

apg_plan_mgmt.plan_capture_threshold

쿼리 실행 계획의 총 비용이 임계값 미만인 경우 계획이 `apg_plan_mgmt.dba_plans` 뷰에 캡처되지 않도록 임계값을 지정합니다.

이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
0	0 - 1.79769e+308	캡처 계획을 위한 `apg_plan_mgmt` 쿼리 계획 총 실행 비용의 임계값을 설정합니다.

자세한 내용은 dba_plans 보기에서 Aurora PostgreSQL 쿼리 계획 검사 섹션을 참조하세요.

apg_plan_mgmt.explain_hashes

`EXPLAIN [ANALYZE]`에서 출력 끝에 `sql_hash` 및 `plan_hash`를 표시할지 지정합니다. 이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
0	0(꺼짐)	EXPLAIN에는 해시 참 옵션이 없는 sql_hash 및 plan_hash가 표시되지 않습니다.

Default	허용된 값	설명
	1(켜짐)	EXPLAIN에는 해시 참 옵션이 없는 sql_hash 및 plan_hash가 표시됩니다.

apg_plan_mgmt.log_plan_enforcement_result

QPM 관리형 계획이 제대로 사용되는지 확인하기 위해 결과를 기록해야 할지 지정합니다. 저장된 일반 계획을 사용하면 로그 파일에 레코드가 기록되지 않습니다. 이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
없음	없음	로그 파일에 계획 시행 결과를 표시하지 않습니다.
	on_error	QPM이 관리형 계획을 사용하지 못하는 경우에만 계획 시행 결과를 로그 파일에 표시합니다.
	모두	성공 및 실패를 포함하여 모든 계획 시행 결과를 로그 파일에 표시합니다.

apg_plan_mgmt.max_databases

쿼리 계획 관리를 사용할 수 있는 Aurora PostgreSQL DB 클러스터의 작성기 인스턴스에 있는 최대 데이터베이스 수를 지정합니다. 기본적으로 최대 10개의 데이터베이스에서 쿼리 계획 관리를 사용할 수 있습니다. 인스턴스에 10개 이상의 데이터베이스가 있는 경우 이 설정의 값을 변경할 수 있습니다. 특정 인스턴스에 있는 데이터베이스 수를 확인하려면 `psql`을 사용하여 인스턴스에 연결합니다. 그런 다음 psql 메타 명령, `\l`을 사용하여 데이터베이스를 나열합니다.

이 파라미터의 값을 변경하려면 설정을 적용하려면 인스턴스를 재부팅해야 합니다.

Default	허용된 값	설명
10	10-2147483647	인스턴스에서 쿼리 계획 관리를 사용할 수 있는 최대 데이터베이스 수입니다.

사용자 지정 DB 클러스터 파라미터 그룹 또는 사용자 지정 DB 파라미터 그룹에서 이 파라미터를 설정할 수 있습니다.

apg_plan_mgmt.max_plans

쿼리 계획 관리자가 `apg_plan_mgmt.dba_plans` 보기에서 유지할 수 있는 최대 SQL 문 수를 설정합니다. 모든 Aurora PostgreSQL 버전에서 이 파라미터를 `10000` 또는 그 이상으로 설정하는 것이 좋습니다.

사용자 지정 DB 클러스터 파라미터 그룹 또는 사용자 지정 DB 파라미터 그룹에서 이 파라미터를 설정할 수 있습니다. 이 파라미터의 값을 변경하려면 설정을 적용하려면 인스턴스를 재부팅해야 합니다.

Default	허용된 값	설명
10000	10-2147483647	`apg_plan_mgmt.dba_plans` 보기에 저장할 수 있는 최대 계획 수입니다. Aurora PostgreSQL 버전 10 및 이전 버전의 기본값은 1000입니다.

자세한 내용은 dba_plans 보기에서 Aurora PostgreSQL 쿼리 계획 검사 섹션을 참조하세요.

apg_plan_mgmt.plan_hash_version

plan_hash 계산이 처리하도록 설계된 사용 사례를 지정합니다. 상위 `apg_plan_mgmt.plan_hash_version` 버전에는 하위 버전의 모든 기능이 포함됩니다. 예를 들어, 버전 3은 버전 2에서 지원하는 사용 사례를 지원합니다.

이 파라미터의 값을 변경한 다음 `apg_plan_mgmt.validate_plans('update_plan_hash')`로 호출해야 합니다. apg_plan_mgmt가 설치되어 있고 계획 테이블에 항목이 있는 각 데이터베이스의 plan_hash 값을 업데이트합니다. 자세한 내용은 계획 검증 섹션을 참조하세요.

Default	허용된 값	설명
1	1	기본 plan_hash 계산입니다.
	2	plan_hash 계산이 다중 스키마 지원을 위해 수정되었습니다.

Default	허용된 값	설명
	3	plan_hash 계산이 다중 스키마 및 파티셔닝된 테이블 지원을 위해 수정되었습니다.
	4	plan_hash 계산이 병렬 연산자 및 구체화 노드를 지원하도록 수정되었습니다.

apg_plan_mgmt.plan_retention_period

`apg_plan_mgmt.dba_plans` 보기에서 계획을 유지할 일 수를 지정하고 이후에는 자동으로 삭제됩니다. 기본적으로 계획이 마지막으로 사용된 후 32일이 경과하면 계획이 삭제됩니다(`apg_plan_mgmt.dba_plans` 보기의 `last_used` 열). 이 설정을 1 이상의 숫자로 변경할 수 있습니다.

이 파라미터의 값을 변경하려면 설정을 적용하려면 인스턴스를 재부팅해야 합니다.

Default	허용된 값	설명
32	1-2147483647	계획이 삭제되기 전에 마지막으로 사용된 이후 최대 일수입니다.

자세한 내용은 dba_plans 보기에서 Aurora PostgreSQL 쿼리 계획 검사 섹션을 참조하세요.

apg_plan_mgmt.unapproved_plan_execution_threshold

최적화 프로그램에서 승인되지 않은 계획을 사용할 수 있는 비용 임계값을 지정합니다. 기본적으로 임계값은 0이므로 최적화 프로그램은 미승인 계획을 실행하지 않습니다. 이 파라미터를 매우 낮은 임계값(예: 100)으로 설정하면 사소한 계획에 대한 계획 적용 오버헤드를 피할 수 있습니다. 반응형 계획 관리 스타일을 사용하여 이 파라미터를 10000000과 같은 매우 큰 값으로 설정할 수도 있습니다. 이렇게 하면 최적화 프로그램이 계획 적용 오버헤드 없이 선택한 모든 계획을 사용할 수 있습니다. 하지만 잘못된 계획이 발견되면 다음 번에 사용되지 않도록 수동으로 '거부됨'으로 표시할 수 있습니다.

이 파라미터의 값은 주어진 계획을 실행하기 위한 예상 비용을 나타냅니다. 승인되지 않은 계획이 예상 비용보다 낮으면 최적화 프로그램은 SQL 문에 이를 사용합니다. 캡처된 계획과 해당 상태(승인됨, 승인되지 않음)를 `dba_plans` 보기에서 볼 수 있습니다. 자세한 내용은 dba_plans 보기에서 Aurora PostgreSQL 쿼리 계획 검사를 참조하세요.

이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
0	0-2147483647	승인되지 않은 계획이 사용되는 아래의 예상 계획 비용입니다.

자세한 내용은 Aurora PostgreSQL 관리형 계획 사용 섹션을 참조하세요.

apg_plan_mgmt.use_plan_baselines

최적화 프로그램이 `apg_plan_mgmt.dba_plans` 보기에 캡처되고 저장된 승인된 계획 중 하나를 사용해야 함을 지정합니다. 기본적으로 이 파라미터는 꺼져 있으므로(거짓) 최적화 프로그램은 추가 평가 없이 생성한 최소 비용 계획을 사용합니다. 이 파라미터를 켜면(true로 설정) 최적화 프로그램이 계획 기준선에서 명령문에 대한 쿼리 실행 계획을 선택하도록 합니다. 자세한 내용은 Aurora PostgreSQL 관리형 계획 사용 섹션을 참조하세요. 이 프로세스를 자세히 설명하는 이미지를 찾으려면 최적화 프로그램에서 실행할 계획을 선택하는 방식. 단원을 참조하세요.

사용자 지정 DB 클러스터 파라미터 그룹 또는 사용자 지정 DB 파라미터 그룹에서 이 파라미터를 설정할 수 있습니다. 이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
false	true	`apg_plan_mgmt.dba_plans` 에서 승인, 선호 또는 승인되지 않은 계획을 사용합니다. 이들 중 어느 것도 최적화 프로그램에 대한 모든 평가 기준을 충족하지 않으면 자체 생성된 최소 비용 계획을 사용할 수 있습니다. 자세한 내용은 최적화 프로그램에서 실행할 계획을 선택하는 방식. 섹션을 참조하세요.
	false	옵티마이저에서 생성한 최소 비용 계획을 사용합니다.

필요에 따라 다른 캡처된 계획의 응답 시간을 평가하고 계획 상태를 변경할 수 있습니다. 자세한 내용은 Aurora PostgreSQL 쿼리 계획 개선 섹션을 참조하세요.

auto_explain.hashes

auto_explain 출력에 sql_hash 및 plan_hash가 표시되는지 지정합니다. 이 파라미터의 값을 변경하면 재부팅이 필요하지 않습니다.

Default	허용된 값	설명
0(꺼짐)	0(꺼짐)	auto_explain 결과는 sql_hash 및 plan_hash 를 표시하지 않습니다.
	1(켜짐)	auto_explain 결과는 sql_hash 및 plan_hash 를 표시합니다.

Aurora PostgreSQL 쿼리 계획 관리를 위한 함수 참조

apg_plan_mgmt 확장은 다음 함수를 제공합니다.

함수

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

지정된 SQL 계획 해시 및 계획 아웃라인을 대상 SQL 계획 해시 및 아웃라인에 복사하여 대상의 계획 해시 및 아웃라인을 덮어씁니다. 이 함수는 apg_plan_mgmt 2.3 이상 릴리스에서 사용할 수 있습니다.

구문

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
```

```
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

반환 값

복사가 성공하면 0을 반환합니다. 입력이 잘못된 경우 예외를 발생시킵니다.

파라미터

파라미터	설명
source_sql_hash	대상 쿼리에 복사할 plan_hash 와 관련된 sql_hash ID입니다.
source_plan_hash	대상 쿼리에 복사할 plan_hash ID입니다.
target_sql_hash	소스 플랜 해시 및 아웃라인으로 업데이트할 쿼리의 sql_hash ID입니다.
target_plan_hash	소스 플랜 해시 및 아웃라인으로 업데이트할 쿼리의 plan_hash ID입니다.
force_update_target_plan_hash	(선택 사항) target_sql_hash 에 대한 소스 계획을 재현할 수 없는 경우에도 쿼리의 target_plan_hash ID가 업데이트됩니다. true로 설정하면 함수를 사용하여 관계 이름과 열이 일치하는 스키마 간에 계획을 복사할 수 있습니다.

사용 노트

이 함수를 사용하면 힌트를 사용하는 계획 해시 및 계획 개요를 다른 유사한 문에 복사할 수 있으므로, 대상 문에서 발생할 때마다 인라인 힌트 문을 사용하지 않아도 됩니다. 업데이트된 대상 쿼리 때문에 잘못된 계획이 생성되는 경우, 이 함수는 오류를 발생시키고 시도한 업데이트를 롤백합니다.

apg_plan_mgmt.delete_plan

관리형 계획을 삭제합니다.

구문

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

반환 값

삭제가 성공한 경우 0을 반환하고, 실패한 경우 -1을 반환합니다.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID.

apg_plan_mgmt.evolve_plan_baselines

이미 승인된 계획이 더 빠르지 여부 또는 쿼리 최적화 프로그램에서 최소 비용 계획으로 식별된 계획이 더 빠르지 여부를 확인합니다.

구문

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

반환 값

승인된 최상의 계획보다 빠르지 않은 계획 수.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID. sql_hash ID 값이 동일한 모든 계획의 평균을 구하려면 NULL을 사용합니다.
min_speed up_factor	최소 속도 향상 인수는 계획이 승인되려면 이미 승인된 계획 중 최상의 계획보다 몇 배 더 빨라야 하는지를 지정합니다. 또는 거부되거나 비활성화되려면 몇 배 더 느려야 하는지를 지정할 수도 있습니다. 이 값은 양수 부동 값입니다.
action	함수가 수행해야 할 작업입니다. 유효 값에는 다음이 포함됩니다. 대/소문자를 구분하지 않습니다. • 'disable' – 최소 속도 향상 인수를 충족하지 않는 각각의 일치하는 계획을 비활성화합니다. • 'approve' – 최소 속도 향상 인수를 충족하는 각각의 일치하는 계획을 활성화하고 상태를 로 설정합니다.approved • 'reject' – 최소 속도 향상 인수를 충족하지 않는 각각의 일치하는 계획의 경우, 상태를 로 설정합니다.rejected • NULL – 이 함수는 단순히 최소 속도 향상 인수를 충족하지 않아서 성능 이점을 없는 계획 수만 반환합니다.

사용 노트

계획 + 실행 시간이 설정한 인수만큼 최상의 승인된 계획보다 더 빠른지 여부에 따라 지정된 계획을 승인됨, 거부됨 또는 비활성화됨으로 설정합니다. 이 작업 파라미터를 'approve' 또는 'reject'로 설정하여 성능 기준을 충족하는 계획을 자동으로 승인하거나 거부할 수 있습니다. 또는 ''(빈 문자열)로 설정하여 성능 실험을 수행한 후 보고서만 생성하고 아무런 작업도 취하지 않을 수 있습니다.

최근에 실행된 계획에 대해 apg_plan_mgmt.evolve_plan_baselines 함수를 무의미하게 다시 실행하는 일을 방지할 수 있습니다. 이를 위해서는 최근에 생성한 미승인 계획까지만 계획을 제한하십시오. 또는 최근 apg_plan_mgmt.evolve_plan_baselines 타임스탬프가 있는 승인된 계획에서만 last_verified 함수를 실행하지 않도록 할 수 있습니다.

성능 실험을 수행하여 기준 내 다른 계획과 각 계획의 계획 + 실행 시간을 비교합니다. 경우에 따라서는 설명문에 대해 계획 하나만 있고 해당 계획이 승인된 계획일 수 있습니다. 이러한 경우에는 계획의 계획 + 실행 시간과 아무런 계획도 사용하지 않을 때의 계획 + 실행 시간을 비교합니다.

각 계획의 증분형 이점(또는 단점)이 apg_plan_mgmt.dba_plans 열의 total_time_benefit_ms 보기에 기록됩니다. 이 값이 양수이면 기준에 이 계획을 포함할 경우 주목할 만한 성능 개선이 있는 것입니다.

각 후보 계획의 계획 + 실행 시간을 수집할 뿐만 아니라, last_verified 보기의 apg_plan_mgmt.dba_plans 열이 current_timestamp로 업데이트됩니다. last_verified 타임스탬프를 사용하면 최근에 성능이 확인된 계획에 대해 이 함수가 다시 실행되지 않도록 할 수 있습니다.

apg_plan_mgmt.get_explain_plan

지정한 SQL 문에 대한 EXPLAIN 문의 텍스트를 생성합니다.

구문

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

반환 값

지정한 SQL 문에 대한 런타임 통계를 반환합니다. 간단한 explainOptionList 계획을 반환하려면 EXPLAIN 없이 사용합니다.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID.
explainOptionList	설명 옵션을 쉼표로 구분한 목록입니다. 유효한 값으로는 'analyze' , 'verbose' , 'buffers' , 'hashes' 및 'format

파라미터	설명
	json'이 있습니다. explainOptionList 가 NULL이거나 빈 문자열('')인 경우 이 함수는 통계 데이터 없이 EXPLAIN 문을 생성합니다.

사용 노트

explainOptionList의 경우 EXPLAIN 문에 사용하는 것과 동일한 옵션 중 하나를 사용할 수 있습니다. Aurora PostgreSQL 옵티마이저는 사용자가 EXPLAIN 문에 제공항하는 옵션 목록을 연결합니다.

apg_plan_mgmt.plan_last_used

공유 메모리에서 지정된 계획의 last_used 날짜를 반환합니다.

> **Note**
> 공유 메모리의 값은 DB 클러스터의 기본 DB 인스턴스에서 항상 최신 상태입니다. 값은 apg_plan_mgmt.dba_plans 뷰의 last_used 열에 주기적으로 플러시됩니다.

구문

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

반환 값

last_used 날짜를 반환합니다.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID.

apg_plan_mgmt.reload

계획을 apg_plan_mgmt.dba_plans 보기에서 공유 메모리로 다시 로드합니다.

구문

```
apg_plan_mgmt.reload()
```

반환 값

없음.

파라미터

없음.

사용 노트

다음 상황의 경우 reload를 호출합니다.

- 새 계획이 복제본으로 전파될 때까지 기다리기 보다는 이 함수를 사용하여 읽기 전용 복제본의 공유 메모리를 즉시 새로 고칩니다.
- 관리형 계획을 가져온 후에 사용합니다.

apg_plan_mgmt.set_plan_enabled

관리형 계획을 활성화하거나 비활성화합니다.

구문

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

반환 값

설정이 성공한 경우 0을 반환하고, 실패한 경우 -1을 반환합니다.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID.
enabled	부울 값(true 또는 false): • true 값은 계획을 활성화합니다. • false 값은 계획을 비활성화합니다.

apg_plan_mgmt.set_plan_status

관리된 계획의 상태를 Approved, Unapproved, Rejected 또는 Preferred로 설정합니다.

구문

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

반환 값

설정이 성공한 경우 0을 반환하고, 실패한 경우 -1을 반환합니다.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID.
status	다음 값 중 하나를 가진 문자열:

파라미터	설명
	• `'Approved'` • `'Unapproved'` • `'Rejected'` • `'Preferred'` 대문자를 사용하든 소문자를 사용하든 `apg_plan_mgmt.dba_plans` 보기에서 상태 값은 초기 대문자로 설정됩니다. 이러한 값에 대한 자세한 내용은 `status`의 [Aurora PostgreSQL 호환 에디션의 apg_plan_mgmt.dba_plans 뷰에 대한 참조](#)를 참조하세요.

apg_plan_mgmt.update_plans_last_used

공유 메모리에 저장된 `last_used` 날짜로 계획 테이블을 즉시 업데이트합니다.

구문

```
apg_plan_mgmt.update_plans_last_used()
```

반환 값

없음.

파라미터

없음.

사용 노트

`dba_plans.last_used` 열에 대항하는 쿼리가 가장 최신 정보를 사용하도록 `update_plans_last_used`를 호출합니다. 만약 `last_used` 날짜가 즉시 업데이트되지 않는다면, 백그라운드 프로세스가 계획 테이블을 `last_used` 날짜로 1시간에 한 번씩(기본값) 업데이트합니다.

예를 들어, 특정 `sql_hash`가 있는 명령문이 느리게 실행되기 시작하면, 성능 회귀 분석이 시작된 이후 해당 명령문에 대해 어떤 계획이 실행되었는지 확인할 수 있습니다. 이렇게 하려면 먼저 공유 메모리의 데이터를 디스크로 플러시하여 `last_used` 날짜가 현재 날짜이고, 그런 다음 성능 회귀가 있는 명령문의 `sql_hash`의 모든 계획을 쿼리합니다. 쿼리에서 `last_used` 날짜가 성능 회귀 분석이 시작

된 날짜보다 크거나 같은지 확인하세요. 쿼리는 성과 회귀를 담당할 수 있는 계획 또는 계획 집합을 식별합니다. apg_plan_mgmt.get_explain_plan과 explainOptionList를 사용하여 verbose, hashes로 설정할 수 있습니다. 또한 apg_plan_mgmt.evolve_plan_baselines를 통해 계획과 더 나은 성능을 발휘할 수 있는 대체 계획을 분석할 수 있습니다.

update_plans_last_used 함수는 DB 클러스터의 기본 DB 인스턴스에만 영향을 줍니다.

apg_plan_mgmt.validate_plans

최적화 프로그램에서 계획을 여전히 다시 생성할 수 있는지 검증합니다. 최적화 프로그램은 계획의 활성 또는 비활성 여부와 상관없이 Approved, Unapproved 및 Preferred 계획을 검증합니다. Rejected 계획은 검증되지 않습니다. 원할 경우 apg_plan_mgmt.validate_plans 함수를 사용하여 유효하지 않은 계획을 삭제하거나 비활성화할 수 있습니다.

구문

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
    action)
```

반환 값

잘못된 계획 수.

파라미터

파라미터	설명
sql_hash	계획의 관리형 SQL 문의 sql_hash ID.
plan_hash	관리형 계획의 plan_hash ID. 동일한 sql_hash ID 값에 대한 모든 계획의 평균을 구하려면 NULL을 사용합니다.
action	함수에서 잘못된 계획에 대해 수행할 작업. 유효한 문자열 값에는 다음이 포함됩니다. 대/소문자를 구분하지 않습니다. • 'disable' – 각각의 잘못된 계획이 비활성화됩니다.

파라미터	설명
	• 'delete' – 각각의 잘못된 계획이 삭제됩니다. • –'update_plan_hash' 정확하게 재현할 수 없는 계획에 대한 plan_hash ID를 업데이트합니다. SQL을 다시 작성하여 계획을 수정할 수도 있습니다. 그런 다음 원본 SQL에 대해 좋은 계획을 Approved 계획으로 등록할 수 있습니다. • NULL – 함수가 잘못된 계획 수만 반환합니다. 다른 작업이 수행되지 않습니다. • '' – 빈 문자열을 지정하면 유효한 계획 수와 잘못된 계획 수를 둘 다 표시하는 메시지가 생성됩니다. 그 밖의 값은 빈 문자열로 간주됩니다.

사용 노트

전체 validate_plans(action) 보기의 모든 관리형 설명문에 대해 모든 관리형 계획을 검증하려면 apg_plan_mgmt.dba_plans 형태를 사용합니다.

validate_plans(sql_hash, plan_hash, action)가 지정된 관리형 설명문의 경우 plan_hash가 지정된 관리형 계획을 검증하려면 sql_hash 형태를 사용합니다.

validate_plans(sql_hash, NULL, action)가 지정된 관리형 설명문에 대해 모든 관리형 계획을 검증하려면 sql_hash 형태를 사용합니다.

Aurora PostgreSQL 호환 에디션의 apg_plan_mgmt.dba_plans 뷰에 대한 참조

apg_plan_mgmt.dba_plans 보기의 계획 정보 열에는 다음이 포함됩니다.

dba_plans 열	설명
cardinality_error	예상 카디널리티와 실제 카디널리티 간의 오차를 측정합니다. 카디널리티는 계획에서 처리할 테이블 행 개수입니다. 카디널리티 오차가 크면 계획이 최적 상태가 아닐 가능성이 높습니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 작성됩니다.

dba_plans 열	설명
compatibility_level	이 파라미터는 쿼리 계획이 마지막으로 검증된 시기를 보여줍니다. Aurora PostgreSQL 버전 12.19, 13.15, 14.12, 15.7, 16.3 이상에서는 Aurora 버전 번호가 표시됩니다. 이전 버전의 경우 기능별 버전 번호가 표시됩니다. Note 이 파라미터 값을 기본 설정으로 유지합니다. Aurora PostgreSQL은 이 값을 자동으로 설정하고 업데이트합니다.
created_by	계획을 생성한 인증된 사용자(session_user)입니다.
enabled	계획의 활성화/비활성화 여부를 나타내는 지표입니다. 모든 계획은 기본적으로 활성화되어 있습니다. 계획을 비활성화하여 최적화 프로그램에서 사용되지 않도록 할 수 있습니다. 이 값을 수정하려면 apg_plan_mgmt.set_plan_enabled 함수를 사용합니다.
environment_variables	최적화 프로그램이 계획이 캡처될 때 재정의한 PostgreSQL Grand Unified Configuration(GUC) 파라미터와 값입니다.
estimated_startup_cost	최적화 프로그램에서 테이블의 행을 전송하기 전 최적화 프로그램 설정 예상 비용입니다.
estimated_total_cost	최종 테이블 행을 전송하는 데 드는 최적화 프로그램 예상 비용입니다.
execution_time_benefit_ms	계획 활성화 시 실행 시간 편익(밀리초)입니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 작성됩니다.
execution_time_ms	계획이 실행될 예상 시간(밀리초)입니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 작성됩니다.
has_side_effects	SQL 문이 데이터 조작 언어(DML) 문이거나 VOLATILE 함수를 포함하는 SELECT 문임을 나타내는 값입니다.

dba_plans 열	설명
last_used	이 값은 계획이 실행될 때마다 또는 계획이 쿼리 최적화 프로그램의 최소 비용 계획일 경우 현재 날짜로 업데이트됩니다. 이 값은 공유 메모리에 저장되고 정기적으로 디스크로 플러시됩니다. 최신 값을 가져오려면 apg_plan_mgmt.plan_last_used(sql_hash, plan_hash) 값을 읽는 대신 last_used 함수를 호출하여 공유 메모리에서 날짜를 읽습니다. 자세한 내용은 apg_plan_mgmt.plan_retention_period 파라미터를 참조하십시오.
last_validated	apg_plan_mgmt.validate_plans 함수 또는 apg_plan_mgmt.evolve_plan_baselines 함수로 계획을 다시 생성할 수 있음이 확인된 최근 날짜 및 시간입니다.
last_verified	계획이 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 지정된 파라미터에 대한 최적 수행 계획인 것으로 확인된 최근 날짜 및 시간입니다.
origin	apg_plan_mgmt.capture_plan_baselines 파라미터를 사용하여 계획을 캡처한 방법입니다. 유효한 값은 다음과 같습니다. M – 수동 계획 캡처 기능을 사용하여 계획을 캡처했습니다. A – 자동 계획 캡처 기능을 사용하여 계획을 캡처했습니다.
param_list	준비된 설명문인 경우 문으로 전달된 파라미터 값입니다.
plan_created	계획이 생성된 날짜 및 시간입니다.
plan_hash	계획 식별자입니다. plan_hash 및 sql_hash의 조합은 특정 계획을 고유하게 식별합니다.
plan_outline	실제 실행 계획을 다시 생성하는 데 사용되고 데이터베이스에 독립적인 계획을 표현합니다. EXPLAIN 출력에 나타나는 연산자에 해당하는 트리의 연산자입니다.
planning_time_ms	플래너를 실행할 실제 시간(밀리초)입니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 작성됩니다.

dba_plans 열	설명
queryId	pg_stat_statements 확장을 통해 계산된 설명문 해시입니다. 이 식별자는 객체 식별자(OID)에 종속되므로 안정형 또는 데이터베이스 독립형 식별자가 아닙니다. 쿼리 계획을 캡처할 때 compute_query_id 가 off면 값이 0이 됩니다.
sql_hash	정규화된(리터럴이 제거됨), SQL 문 텍스트의 해시 값입니다.
sql_text	SQL 문의 전체 텍스트입니다.
status	최적화 프로그램에서 계획을 사용하는 방법을 결정하는 계획의 상태입니다. 유효 값에는 다음이 포함됩니다. • Approved – 최적화 프로그램에서 실행하기로 선택할 수 있는 사용 가능한 계획입니다. 최적화 프로그램은 관리형 설명문의 승인된 계획(기준) 세트에서 최소 비용 계획을 실행합니다. 계획을 승인됨 상태로 재설정하려면 apg_plan_mgmt.evolve_plan_baselines 함수를 사용합니다. • Unapproved – 사용할 수 있는지 확인되지 않은 캡처된 계획입니다. 자세한 내용은 계획 성능 평가 섹션을 참조하세요. • Rejected – 최적화 프로그램에서 사용할 수 없는 계획입니다. 자세한 내용은 느린 계획 거부 또는 비활성화 섹션을 참조하세요. • Preferred – 결정한 계획이 관리형 설명문에 사용할 기본 계획입니다. 최적화 프로그램의 최소 비용 계획이 승인된 또는 기본 설정된 계획이 아닌 경우 계획 시행 오버헤드를 줄일 수 있습니다. 이를 위해 승인된 계획의 하위 집합을 Preferred 로 설정하십시오. 최적화 프로그램의 최소 비용이 Approved 계획이 아니면 Preferred 계획이 Approved 계획에 앞서 선택됩니다. 계획을 Preferred 로 재설정하려면 apg_plan_mgmt.set_plan_status 함수를 사용합니다.

dba_plans 열	설명
stmt_name	PREPARE 문 안에 있는 SQL 문의 이름입니다. 이름이 지정되지 않은 준비된 설명문의 경우 이 값이 빈 문자열입니다. 준비되지 않은 설명문의 경우 이 값이 NULL입니다.
total_time_benefit_ms	이 계획 활성화 시 총 시간 편익(밀리초)입니다. 이 값은 계획 시간 및 실행 시간을 모두 고려합니다. 이 값이 음수이면 이 계획을 활성화하는 것이 불리합니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines 함수에 의해 작성됩니다.

쿼리 계획 관리의 고급 기능

아래에서 Aurora PostgreSQL 쿼리 계획 관리(QPM)의 고급 기능에 대한 정보를 찾을 수 있습니다.

주제

- 복제본에서 Aurora PostgreSQL 실행 계획 캡처
- 테이블 파티션 지원

복제본에서 Aurora PostgreSQL 실행 계획 캡처

쿼리 계획 관리(QPM)를 사용하면 Aurora 복제본에서 생성된 쿼리 계획을 캡처하여 Aurora DB 클러스터의 기본 DB 인스턴스에 저장할 수 있습니다. 모든 Aurora 복제본에서 쿼리 계획을 수집하고 기본 인스턴스의 중앙 영구 테이블에서 최적의 계획 집합을 유지 관리할 수 있습니다. 그런 다음 필요할 때 이 계획을 다른 복제본에 적용할 수 있습니다. 이를 통해 실행 계획의 안정성을 유지하고 DB 클러스터 및 엔진 버전 전반의 쿼리 성능을 개선할 수 있습니다.

주제

- 사전 조건
- Aurora 복제본의 계획 캡처 관리
- 문제 해결

사전 조건

Aurora 복제본에서 **capture_plan_baselines parameter** 켜기 - Aurora 복제본에서 계획을 캡처하려면 capture_plan_baselines 파라미터를 자동 또는 수동으로 설정합니다. 자세한 내용은 apg_plan_mgmt.capture_plan_baselines 단원을 참조하십시오.

postgres_fdw 확장 설치 - Aurora 복제본에서 계획을 캡처하려면 postgres_fdw 외부 데이터 래퍼 확장을 설치해야 합니다. 각 데이터베이스에서 다음 명령을 실행하여 확장을 설치합니다.

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Aurora 복제본의 계획 캡처 관리

Aurora 복제본의 계획 캡처 켜기

Aurora 복제본에서 계획 캡처를 생성하거나 제거하려면 rds_superuser 권한이 있어야 합니다. 사용자 역할 및 권한에 대한 자세한 내용은 PostgreSQL 역할 및 권한 이해를 참조하세요.

계획을 캡처하려면 다음과 같이 라이터 DB 인스턴스에서 apg_plan_mgmt.create_replica_plan_capture 함수를 호출하세요.

```
postgres=> CALL apg_plan_mgmt.create_replica_plan_capture('endpoint', 'password');
```

- 엔드포인트 - Aurora Global Database 라이터 엔드포인트 또는 cluster_endpoint는 Aurora 복제본의 계획 캡처에 대한 장애 조치 지원을 제공합니다.

  Aurora Global Database 라이터 엔드포인트에 대한 자세한 내용은 Amazon Aurora Global Database의 엔드포인트 보기 섹션을 참조하시기 바랍니다.

  클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora용 클러스터 엔드포인트 섹션을 참조하세요.
- 암호 - 보안을 강화하려면 암호를 생성할 때 아래 지침을 따르는 것이 좋습니다.
  - 최소 8개의 문자를 포함해야 합니다.
  - 최소한 대문자 1개, 소문자 1개 및 숫자 1개를 포함해야 합니다.
  - 하나 이상의 특수 문자(?, !, #, <, >, * 등)가 있어야 합니다.

> **Note**
>
> 엔드포인트, 암호 또는 포트 번호를 변경하는 경우 엔드포인트 및 암호를 사용하여 `apg_plan_mgmt.create_replica_plan_capture()`를 다시 실행하여 계획 캡처를 다시 초기화해야 합니다. 이렇게 하지 않으면 Aurora 복제본에서 계획을 캡처하는 데 실패합니다.

Aurora 복제본의 계획 캡처 끄기

파라미터 그룹에서 값을 `off`로 설정하여 Aurora 복제본의 `capture_plan_baselines` 파라미터를 끌 수 있습니다.

Aurora 복제본의 계획 캡처 제거

Aurora 복제본에서 계획 캡처를 완전히 제거할 수 있지만 먼저 확인할 것이 있습니다. 계획 캡처를 제거하려면 다음과 같이 `apg_plan_mgmt.remove_replica_plan_capture`를 호출합니다.

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

Aurora 복제본에서 계획 캡처를 설정하려면 엔드포인트와 암호를 사용하여 apg_plan_mgmt.create_replica_plan_capture()를 다시 호출해야 합니다.

문제 해결

Aurora 복제본에서 계획이 예상대로 캡처되지 않는 경우 아래에서 문제 해결 아이디어와 해결 방법을 찾을 수 있습니다.

- 파라미터 설정 - `capture_plan_baselines` 파라미터가 계획 캡처를 켜도록 적절한 값으로 설정되어 있는지 확인합니다.
- **`postgres_fdw`** 확장 설치 - 다음 쿼리를 사용하여 `postgres_fdw` 설치 여부를 확인합니다.

```
postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
```

- create_replica_plan_capture()가 호출됨 - 다음 명령을 사용하여 사용자 매핑이 종료되는지 확인합니다. 그렇지 않으면 `create_replica_plan_capture()`를 호출하여 기능을 초기화합니다.

```
postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- 엔드포인트 및 포트 번호 - 해당하는 경우 엔드포인트 및 포트 번호를 확인합니다. 값이 올바르지 않으면 오류 메시지가 표시되지 않습니다.

  다음 명령을 사용하여 create()에 엔드포인트가 사용되었는지 확인하고 엔드포인트가 있는 데이터베이스를 확인합니다.

  ```
  postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- reload() - 삭제 함수를 유효하게 하려면 Aurora 복제본에서 apg_plan_mgmt.delete_plan()을 호출한 후 apg_plan_mgmt.reload()를 호출해야 합니다. 이렇게 하면 변경 사항을 성공적으로 구현할 수 있습니다.
- 암호 - 앞서 언급한 지침에 따라 create_replica_plan_capture()에 암호를 입력해야 합니다. 이렇게 하지 않으면 오류 메시지가 수신됩니다. 자세한 내용은 Aurora 복제본의 계획 캡처 관리 섹션을 참조하세요. 요구 사항에 맞는 다른 암호를 사용하세요.
- 리전 간 연결 - Aurora 복제본의 계획 캡처는 Aurora Global Database에서도 지원됩니다. Aurora Global Database에서는 라이터 인스턴스와 Aurora 복제본이 서로 다른 리전에 있을 수 있습니다. Aurora Global Database 라이터 엔드포인트를 사용하여 장애 조치 또는 전환 이벤트 후 연결을 유지해야 합니다. Aurora Global Database 엔드포인트에 대한 자세한 내용은 Amazon Aurora Global Database의 엔드포인트 보기를 참조하시기 바랍니다. 라이터 인스턴스와 리전 간 복제본은 VPC 피어링을 사용하여 통신할 수 있어야 합니다. 자세한 내용은 VPC 피어링을 참조하세요. 리전 간 장애 조치가 발생하는 경우 엔드포인트를 새 기본 DB 클러스터 엔드포인트로 재구성해야 합니다.

> **Note**
>
> Aurora Global Database 라이터 엔드포인트 대신 클러스터 엔드포인트를 사용하는 경우 전역 장애 조치 또는 전환 작업을 수행한 후 클러스터 엔드포인트를 업데이트해야 합니다.

테이블 파티션 지원

Aurora PostgreSQL 쿼리 계획 관리(QPM)는 다음 버전에서 선언적 테이블 파티셔닝을 지원합니다.

- 15.3 이상의 15 버전
- 14.8 이상의 14 버전
- 13.11 이상의 13 버전

자세한 내용은 테이블 파티셔닝을 참조하세요.

주제

- 테이블 파티션 설정
- 테이블 파티션에 대한 계획 캡처
- 테이블 파티션 계획 적용
- 이름 지정 규칙

테이블 파티션 설정

Aurora PostgreSQL QPM에서 테이블 파티션을 설정하려면 다음 작업을 수행합니다.

1. DB 클러스터 파라미터 그룹에서 `apg_plan_mgmt.plan_hash_version`을 3 이상으로 설정합니다.
2. 쿼리 계획 관리를 사용하고 `apg_plan_mgmt.dba_plans` 보기에 항목이 있는 데이터베이스로 이동합니다.
3. `apg_plan_mgmt.validate_plans('update_plan_hash')`를 호출하여 계획 테이블의 `plan_hash` 값을 업데이트합니다.
4. `apg_plan_mgmt.dba_plans` 보기에 항목이 있으며 쿼리 계획 관리가 활성화된 모든 데이터베이스에 대해 2~3단계를 반복합니다.

이런 파라미터에 대한 자세한 내용은 Aurora PostgreSQL 쿼리 계획 관리를 위한 파라미터 참조 섹션을 참조하세요.

테이블 파티션에 대한 계획 캡처

QPM에서는 다양한 계획이 `plan_hash` 값으로 구분됩니다. `plan_hash`가 어떻게 변하는지 이해하려면 먼저 비슷한 종류의 계획을 이해해야 합니다.

계획이 동일한 것으로 간주되려면 Append 노드 수준에서 누적된 액세스 방법, 숫자 제거 인덱스 이름 및 숫자 제거 파티션 이름의 조합이 일정해야 합니다. 계획에서 액세스하는 특정 파티션은 중요하지 않습니다. 다음 예제에서는 파티션이 4개인 `tbl_a` 테이블이 생성됩니다.

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
```

```
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

다음 계획은 쿼리에서 조회하는 파티션 수에 관계없이 단일 스캔 방법을 사용하여 tbl_a를 스캔하기 때문에 동일한 것으로 간주됩니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
-------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;
```

```
                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

상위 수준에서 액세스 방법, 숫자 제거 인덱스 이름 및 숫자 제거 파티션 이름이 `SeqScan tbl_a`, `IndexScan (i_idx) tbl_a`이므로 다음 3가지 계획도 동일한 것으로 간주됩니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
```

```
        Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

하위 파티션의 순서 및 발생 횟수에 관계없이 액세스 방법, 숫자 제거 인덱스 이름 및 숫자 제거 파티션 이름은 위의 각 계획에 대해 상위 수준에서 일정합니다.

그러나 다음 조건 중 하나라도 충족되면 계획이 다른 것으로 간주됩니다.

- 계획에 추가 액세스 방법이 사용됩니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
```

```
                  Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

• 계획의 모든 액세스 방법이 더 이상 사용되지 않습니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

• 인덱스 방법과 연결된 인덱스가 변경됩니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                             QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

테이블 파티션 계획 적용

파티셔닝된 테이블에 대한 승인된 계획은 위치 대응과 함께 적용됩니다. 계획은 파티션에만 국한되지 않으며, 원래 쿼리에서 참조된 계획이 아닌 다른 파티션에도 적용할 수 있습니다. 또한 계획에는 원래 승인된 개요와 다른 수의 파티션에 액세스하는 쿼리에 적용할 수 있는 기능이 있습니다.

예를 들어 승인된 개요가 아래 계획에 대한 것이라면 다음과 같습니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

그런 다음 2개, 4개 이상의 파티션을 참조하는 SQL 쿼리에도 이 계획을 적용할 수 있습니다. 2개 및 4개 파티션 액세스에 대한 이러한 시나리오에서 발생할 수 있는 계획은 다음과 같습니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-----------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
```

```
        Index Cond: ((i >= 990) AND (i <= 3100))
        Filter: ((j < 9910) AND (k > 50))
  ->  Seq Scan on tbl_a2 tbl_a_2
        Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
  ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
        Index Cond: ((i >= 990) AND (i <= 3100))
        Filter: ((j < 9910) AND (k > 50))
  ->  Seq Scan on tbl_a4 tbl_a_4
        Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
----------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

각 파티션마다 다른 액세스 방법을 사용하는 승인된 다른 계획을 고려해 보세요.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
```

```
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

이 경우 두 파티션에서 읽어 들이는 계획은 실행되지 않습니다. 승인된 계획의 모든 조합(액세스 방법, 인덱스 이름)을 사용할 수 없으면 계획을 적용할 수 없습니다. 예를 들어 다음 계획에는 서로 다른 계획 해시가 있으며 이러한 경우에는 승인된 계획을 적용할 수 없습니다.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

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

```
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

이름 지정 규칙

QPM에서 선언적 파티셔닝된 테이블을 사용하여 계획을 적용하려면 상위 테이블, 테이블 파티션 및 인덱스에 대한 특정 이름 지정 규칙을 따라야 합니다.

- 상위 테이블 이름 - 이 이름은 숫자만이 아니라 알파벳이나 특수 문자로 구분해야 합니다. 예를 들어 tA, tB 및 tC는 개별 상위 테이블에 사용할 수 있는 이름이지만, t1, t2 및 t3는 사용할 수 없습니다.
- 개별 파티션 테이블 이름 - 동일한 상위 파티션의 파티션은 숫자만 달라야 합니다. 예를 들어 tA의 허용 가능한 파티션 이름은 tA1, tA2 또는 t1A, t2A나 여러 자리 숫자일 수 있습니다.

  문자, 특수 문자로 차이를 두면 계획 적용이 보장되지 않습니다.
- 인덱스 이름 - 파티션 테이블 계층 구조에서 모든 인덱스의 이름이 고유하도록 해야 합니다. 즉, 이름에서 숫자가 아닌 부분이 달라야 합니다. 예를 들어 이름이 `tA`인 파티셔닝 테이블에 인덱스 이름이 `tA_col1_idx1`인 인덱스가 있는 경우 이름이 `tA_col1_idx2`인 다른 인덱스를 포함할 수 없습니다. 하지만 이름이 `tA_a_col1_idx2`인 인덱스는 포함할 수 있습니다. 이름에서 숫자가 아닌 부분이 고유하기 때문입니다. 이 규칙은 상위 테이블과 개별 파티션 테이블 모두에 생성된 인덱스에 적용됩니다.

위의 이름 지정 규칙을 준수하지 않을 경우 승인된 계획 적용이 실패할 수 있습니다. 다음은 실패한 적용의 예를 보여줍니다.

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;
```

```
                            QUERY PLAN
--------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                            QUERY PLAN
-------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

두 계획이 동일하게 보일 수도 있지만 하위 테이블의 이름 때문에 `Plan Hash` 값이 다릅니다. 테이블 이름이 숫자만이 아닌 영문자에 따라 다양하므로 적용이 실패합니다.

확장 및 외부 데이터 래퍼 작업

Aurora PostgreSQL 호환 버전 DB 클러스터의 기능을 확장하기 위해 다양한 PostgreSQL 확장을 설치하고 사용할 수 있습니다. 예를 들어 사용 사례에서 매우 큰 테이블에 집중적인 데이터 입력이 필요한 경우 `pg_partman` 확장을 설치하여 데이터를 분할하고 워크로드를 분산할 수 있습니다.

> **Note**
>
> Aurora PostgreSQL 14.5부터 Aurora PostgreSQL은 PostgreSQL용 신뢰할 수 있는 언어 확장을 지원합니다. 이 기능은 Aurora PostgreSQL에 추가할 수 있는 `pg_tle` 확장으로 구현됩니다. 이 확장을 사용하면 개발자는 설정 및 구성 요구 사항을 간소화하고 새 확장에 대한 많은 예비 테스트를 줄일 수 있는 안전한 환경에서 자체 PostgreSQL 확장을 만들 수 있습니다. 자세한 내용은 PostgreSQL용 신뢰할 수 있는 언어 확장 작업 단원을 참조하십시오.

경우에 따라 확장을 설치하는 대신 Aurora PostgreSQL DB 클러스터의 사용자 지정 DB 클러스터 파라미터 그룹의 `shared_preload_libraries` 목록에 특정 모듈을 추가할 수 있습니다. 일반적으로 기본 DB 클러스터 파라미터 그룹은 `pg_stat_statements`만 로드하지만 목록에 추가할 수 있는 다른 모듈도 몇 개 있습니다. 예를 들어, PostgreSQL pg_cron 확장을 사용하여 유지 관리 예약에 자세히 설명된 대로 `pg_cron` 모듈을 추가하여 스케줄링 기능을 추가할 수 있습니다. 또 다른 예로, `auto_explain` 모듈을 로드하여 쿼리 실행 계획을 로깅할 수 있습니다. 자세히 알아보려면 AWS 지식 센터에서 쿼리 실행 계획 로깅을 참조하세요.

외부 데이터에 대한 액세스를 제공하는 확장을 구체적으로 외부 데이터 래퍼(FDW)라고 합니다. 하나의 예로는 `oracle_fdw` 확장을 사용하면 Aurora PostgreSQL DB 클러스터가 Oracle 데이터베이스와 함께 작동할 수 있는 경우가 있습니다.

또한 Aurora PostgreSQL DB 인스턴스에 설치할 수 있는 확장 프로그램을 `rds.allowed_extensions` 파라미터에 나열하여 정확하게 지정할 수 있습니다. 자세한 내용은 PostgreSQL 확장 프로그램의 설치 제한을 참조하세요.

아래에서 Aurora PostgreSQL에 사용할 수 있는 일부 확장, 모듈 및 FDW 설정과 사용에 대한 정보를 확인할 수 있습니다. 간단히 이 모든 것을 '확장'이라고 합니다. 현재 사용 가능한 Aurora PostgreSQL 버전에서 사용할 수 있는 확장 목록은 Aurora PostgreSQL 릴리스 정보에서 Extension versions for Amazon Aurora PostgreSQL(Aurora PostgreSQL 확장 버전)을 참조하세요.

- lo 모듈을 사용하여 대형 객체 관리
- PostGIS 확장을 사용하여 공간 데이터 관리

- pg_partman 확장자를 사용하여 PostgreSQL 파티션 관리하기
- PostgreSQL pg_cron 확장을 사용하여 유지 관리 예약
- pgAudit를 사용하여 데이터베이스 활동 로깅
- pglogical을 사용하여 인스턴스 간 데이터 동기화
- oracle_fdw 확장을 사용하여 Oracle 데이터베이스 작업
- tds_fdw 확장을 사용하여 SQL Server 데이터베이스 작업

PostgreSQL에 대한 Amazon Aurora 위임 확장 지원 사용

PostgreSQL에 대한 Amazon Aurora 위임 확장 지원을 사용하면 `rds_superuser` 역할이 아닌 사용자에게도 확장 관리를 위임할 수 있습니다. 이 위임 확장 지원을 통해 `rds_extension`이라는 새 역할이 생성되며, 이 역할이 할당되어야 해당 사용자가 다른 확장을 관리할 수 있습니다. 이 역할은 확장을 생성하고, 업데이트하고, 삭제할 수 있습니다.

Aurora PostgreSQL DB 인스턴스에 설치할 수 있는 확장을 `rds.allowed_extensions` 파라미터에 등록하여 지정할 수 있습니다. 자세한 내용은 Amazon RDS for PostgreSQL로 PostgreSQL 확장 사용을 참조하세요.

`rds.allowed_delegated_extensions` 파라미터를 사용하여 `rds_extension` 역할이 있는 사용자가 관리할 수 있는 가용 확장 목록을 제한할 수 있습니다.

위임 확장 지원은 다음 버전에서 사용할 수 있습니다.

- 모든 상위 버전
- 15.5 이상의 15 버전
- 14.10 이상의 14 버전
- 13.13 이상의 13 버전
- 12.17 이상의 12 버전

주제

- 사용자에게 위임 확장 지원 활성화
- Aurora 위임 확장 지원에서 PostgreSQL에 대해 사용되는 구성
- 위임 확장에 대한 지원 비활성화
- Amazon Aurora 위임 확장 지원 사용의 이점

- [PostgreSQL에 대한 Aurora 위임 확장 지원 제한]
- [특정 확장에 필요한 권한]
- [보안 고려 사항]
- [확장 삭제 캐스케이드 비활성화]
- [위임 확장 지원을 사용하여 추가할 수 있는 확장의 예]

사용자에게 위임 확장 지원 활성화

사용자에게 위임 확장 지원을 활성화하려면 다음을 수행해야 합니다.

1. 사용자에게 **rds_extension** 역할 부여 - `rds_superuser` 권한으로 데이터베이스에 연결하고 다음 명령을 실행합니다.

   ```
   Postgres => grant rds_extension to user_name;
   ```

2. 위임된 사용자가 관리할 수 있는 확장 목록 설정 - `rds.allowed_delegated_extensions`를 통해 DB 클러스터 파라미터에서 `rds.allowed_extensions`를 사용하여 가용 확장의 하위 집합을 지정할 수 있습니다. 다음 수준 중 하나에서 이 작업을 수행할 수 있습니다.

   - 클러스터 또는 인스턴스 파라미터 그룹에서 AWS Management Console 또는 API를 통해 가능합니다. 자세한 내용은 [Amazon Aurora의 파라미터 그룹] 단원을 참조하십시오.
   - 데이터베이스 수준에서 다음 명령을 사용하세요.

   ```
   alter database database_name set rds.allowed_delegated_extensions =
    'extension_name_1,
                        extension_name_2,...extension_name_n';
   ```

   - 사용자 수준에서 다음 명령을 사용하세요.

   ```
   alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                        extension_name_2,...extension_name_n';
   ```

> **Note**
>
> `rds.allowed_delegated_extensions` 동적 파라미터를 변경한 후에는 데이터베이스를 다시 시작할 필요가 없습니다.

3. 확장 생성 프로세스 중에 생성된 객체에 대한 위임된 사용자 액세스 허용 - 특정 확장에서는 추가 권한을 부여해야 `rds_extension` 역할을 가진 사용자가 해당 개체에 액세스할 수 있는 개체를 만들 수 있습니다. `rds_superuser`는 위임된 사용자에게 해당 객체에 대한 액세스 권한을 부여해야 합니다. 이벤트 트리거를 사용하여 위임된 사용자에게 권한을 자동으로 부여하는 것도 한 가지 방법입니다. 자세한 정보는 위임 확장에 대한 지원 비활성화에서 이벤트 트리거 예제를 참조하세요.

Aurora 위임 확장 지원에서 PostgreSQL에 대해 사용되는 구성

구성 이름	설명	기본 값	참고	권한을 수정하거나 부여할 수 있는 사람
`rds.a ed_de ted_e sions`	이 파라미터는 rds_extension 역할이 데이터베이스에서 관리할 수 있는 확장을 제한합니다. rds.allowed_extensions의 하위 집합이어야 합니다.	빈 문자열	• 기본적으로 이 파라미터는 빈 문자열이므로 `rds_extension` 을 사용하는 사용자에게 확장이 위임되지 않았음을 의미합니다. • 사용자에게 권한이 있는 경우 모든 지원되는 확장을 추가할 수 있습니다. 이렇게 하려면 `rds.allowed_delegated_extensions` 파라미터를 쉼표로 구분된 확장 이름(문자열)으로 설정합니다. 이 파라미터에 확장 목록을 추가하면 `rds_exten`	rds_superuser

구성 이름	설명	기본 값	참고	권한을 수정하거나 부여할 수 있는 사람
			sion 역할의 사용자가 설치할 수 있는 확장을 명시적으로 파악할 수 있습니다. • *로 설정하면 rds_allowed_extensions 에 나열된 모든 확장이 rds_extension 역할의 사용자에게 위임된다는 의미입니다. 이 파라미터 설정에 대한 자세한 내용은 사용자에게 위임 확장 지원 활성화 섹션을 참조하세요.	

구성 이름	설명	기본 값	참고	권한을 수정하거나 부여할 수 있는 사람
`rds.a ed_ex ions`	이 파라미터를 사용하면 고객이 Aurora PostgreSQL DB 인스턴스에 설치할 수 있는 확장을 제한할 수 있습니다. 자세한 내용은 [PostgreSQL 확장 설치 제한](#)을 참조하세요.	"*"	기본적으로 이 파라미터는 '*'로 설정되어 있습니다. 즉, RDS for PostgreSQL과 Aurora PostgreSQL에서 지원되는 모든 확장은 필요한 권한이 있는 사용자가 만들 수 있습니다. 이 값이 비어 있으면 Aurora PostgreSQL DB 인스턴스에 확장을 설치할 수 없습니다.	관리자

구성 이름	설명	기본 값	참고	권한을 수정하거나 부여할 수 있는 사람
rds-deleg ated_ nsion ow_dr ascac	이 파라미터는 rds_extension 권한이 있는 사용자가 캐스케이드 옵션을 사용하여 확장을 삭제할 수 있는 기능을 제어합니다.	꺼	기본적으로 rds-delegated_extension_allow_drop_cascade 는 off로 설정됩니다. 즉, rds_extension 사용자는 캐스케이드 옵션을 사용하여 확장 프로그램을 삭제할 수 없습니다. 이 기능을 부여하려면 rds.delegated_extension_allow_drop_cascade 파라미터를 on으로 설정해야 합니다.	rds_superuser

위임 확장에 대한 지원 비활성화

부분적으로 비활성화

위임된 사용자는 새 확장을 만들 수는 없지만 기존 확장을 계속 업데이트할 수는 있습니다.

- DB 클러스터 파라미터 그룹에서 rds.allowed_delegated_extensions를 기본값으로 재설정합니다.
- 데이터베이스 수준에서 다음 명령을 사용하세요.

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- 사용자 수준에서 다음 명령을 사용하세요.

```
alter user user_name reset rds.allowed_delegated_extensions;
```

전체적으로 비활성화

사용자로부터 rds_extension 역할을 취소하면 사용자는 표준 권한으로 되돌아갑니다. 해당 사용자는 더 이상 확장을 생성, 업데이트 또는 삭제할 수 없습니다.

```
postgres => revoke rds_extension from user_name;
```

이벤트 트리거 예제

확장 생성 시 만들어진 객체에 대한 권한 설정이 필요한 확장을 rds_extension 권한이 있는 위임된 사용자가 사용할 수 있도록 하려면, 아래 이벤트 트리거 예제를 사용자 지정하고 위임된 사용자가 전체 기능에 액세스할 수 있도록 하려는 확장만 추가하면 됩니다. 이 이벤트 트리거는 template1(기본 템플릿)에서 만들 수 있으므로, template1에서 만든 모든 데이터베이스에는 해당 이벤트 트리거가 있습니다. 위임된 사용자가 확장을 설치하면 이 트리거는 확장에서 생성된 객체에 대한 소유권을 자동으로 부여합니다.

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';
```

```
        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %, 
schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user, 
schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_gaz TO 
%I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_lex TO 
%I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_rules 
TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE format('ALTER TEXT SEARCH DICTIONARY %I.intdict OWNER TO %I;', 
schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE 
%I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE 
%I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE 
%I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN 
SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA topology TO 
%I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA topology TO %I 
WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE ON SCHEMA topology TO %I WITH GRANT OPTION;', 
databaseowner);
        END IF;
    END LOOP;
```

```
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE create_ext();
```

Amazon Aurora 위임 확장 지원 사용의 이점

PostgreSQL에 대한 Amazon Aurora 위임 확장 지원을 사용하면 `rds_superuser` 역할이 아닌 사용자에게도 확장 관리를 안전하게 위임할 수 있습니다. 이 기능에는 다음과 같은 이점이 있습니다.

- 원하는 사용자에게 확장 관리를 쉽게 위임할 수 있습니다.
- 이 작업에는 `rds_superuser` 역할이 필요하지 않습니다.
- 동일한 DB 클러스터의 여러 데이터베이스에 대해 서로 다른 확장 세트를 지원하는 기능을 제공합니다.

PostgreSQL에 대한 Aurora 위임 확장 지원 제한

- 확장 생성 프로세스 중에 생성되는 객체에서 확장이 제대로 작동하려면 추가 권한이 필요할 수 있습니다.

특정 확장에 필요한 권한

다음과 같은 확장을 생성, 사용 또는 업데이트하려면 위임된 사용자에게 다음과 같은 함수, 테이블 및 스키마에 대해 필요한 권한이 있어야 합니다.

소유권이나 권한이 필요한 확장	함수	표	스키마	텍스트 검색 사전	설명
addre tanda er_da s		us_gaz, us_lex, us_lex, I.us_rules			
amch	bt_index_check, bt_index_ parent_check				
dict_i				intdict	
pg_p n		custom_ti me_partitions, part_config, part_config_sub			
pg_s taten					
Post(	st_tileenvelope	spatial_ref_sys			
postç aster					

소유권이나 권한이 필요한 확장	함수	표	스키마	텍스트 검색 사전	설명
postg opolo		topology, layer	topology		위임된 사용자는 데이터베이스 소유자여야 함
log_f	create_foreign_table_for_log_file				
rds_t	role_password_encryption_type				
postg iger_ oder		geocode_settings_default, geocode_settings	tiger		
pg_fr acem	pg_freespace				
pg_v lity	pg_visibility				

보안 고려 사항

rds_extension 역할이 있는 사용자는 접속 권한이 있는 모든 데이터베이스의 확장을 관리할 수 있다는 점을 기억하세요. 위임된 사용자가 단일 데이터베이스의 확장을 관리하도록 하려는 경우, 각 데이터베이스의 모든 공개 권한을 취소한 다음 해당 특정 데이터베이스의 연결 권한을 위임 사용자에게 명시적으로 부여하는 것이 좋습니다.

사용자가 여러 데이터베이스의 정보에 액세스할 수 있도록 하는 확장에는 여러 가지가 있습니다. 이러한 확장을 rds.allowed_delegated_extensions에 추가하기 전에 rds_extension 권한을 부여한 사용자가 데이터베이스 간 기능을 사용할 수 있는지 확인하세요. 예를 들어, postgres_fdw 및 dblink는 동일한 인스턴스 또는 원격 인스턴스의 여러 데이터베이스를 쿼리할 수 있는 기능을 제공합니다. log_fdw는 인스턴스의 모든 데이터베이스에 대한 postgres 엔진 로그 파일을 읽습니다. 이 로그 파일에는 여러 데이터베이스의 느린 쿼리나 오류 메시지가 포함될 수 있습니다. pg_cron은 예약된 백그라운드 작업을 DB 인스턴스에서 실행할 수 있도록 하고 작업이 다른 데이터베이스에서 실행되도록 구성할 수 있습니다.

확장 삭제 캐스케이드 비활성화

rds_extension 역할의 사용자가 캐스케이드 옵션을 사용하여 확장을 삭제하는 기능은 rds.delegated_extension_allow_drop_cascade 파라미터에 의해 제어됩니다. 기본적으로 rds-delegated_extension_allow_drop_cascade는 off로 설정됩니다. 즉, rds_extension 역할의 사용자는 아래 쿼리에 나와 있는 캐스케이드 옵션으로 확장을 삭제할 수 없습니다.

```
DROP EXTENSION CASCADE;
```

이렇게 하면 확장에 종속된 객체와 해당 객체에 종속된 모든 객체가 자동으로 삭제되기 때문입니다. 캐스케이드 옵션을 사용하려고 시도하면 오류가 발생합니다.

이 기능을 부여하려면 rds.delegated_extension_allow_drop_cascade 파라미터를 on으로 설정해야 합니다.

rds.delegated_extension_allow_drop_cascade 동적 파라미터를 변경하기 위해 데이터베이스를 다시 시작할 필요가 없습니다. 다음 수준 중 하나에서 이 작업을 수행할 수 있습니다.

- 클러스터 또는 인스턴스 파라미터 그룹에서 AWS Management Console 또는 API를 통해 가능합니다.
- 데이터베이스 수준에서 다음 명령을 사용하세요.

```
alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
```

- 사용자 수준에서 다음 명령을 사용하세요.

```
alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
```

위임 확장 지원을 사용하여 추가할 수 있는 확장의 예

- rds_tools

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

lo 모듈을 사용하여 대형 객체 관리

lo 모듈(확장)은 JDBC 또는 ODBC 드라이버를 통해 PostgreSQL 데이터베이스로 작업하는 데이터베이스 사용자 및 개발자를 위한 것입니다. JDBC와 ODBC는 모두 참조가 변경될 때 데이터베이스에서 대형 객체의 삭제를 처리할 것으로 예상합니다. 그러나 PostgreSQL은 이런 식으로 작동하지 않습니다. PostgreSQL 참조가 변경될 때 객체를 삭제해야 한다고 가정하지 않습니다. 따라서 객체는 참조되지 않고 디스크에 남아 있게 됩니다. lo 확장에는 필요한 경우 객체를 삭제하기 위해 참조 변경 사항을 트리거하는 데 사용하는 함수가 포함되어 있습니다.

> **Tip**
>
> 데이터베이스가 lo 확장의 이점을 누릴 수 있는지 확인하려면 `vacuumlo` 유틸리티를 사용하여 분리되어 있는 대형 객체가 있는지 확인합니다. 작업을 수행하지 않고 분리되어 있는 대형 객체 수를 확인하려면 -n 옵션(no-op)과 함께 유틸리티를 실행하면 됩니다. 자세한 방법은 다음 [vacuumlo utility](#) 섹션을 참조하세요.

lo 모듈은 Aurora PostgreSQL 13.7, 12.11, 11.16, 10.21 이상 마이너 버전에서 사용할 수 있습니다.

모듈(확장)을 설치하려면 `rds_superuser` 권한이 필요합니다. lo 확장을 설치하면 데이터베이스에 다음이 추가됩니다.

- `lo` - 바이너리 대형 객체(BLOB) 및 기타 대형 객체에 사용할 수 있는 대형 객체(lo) 데이터 유형입니다. `lo` 데이터 유형은 `oid` 데이터 유형의 영역에 있습니다. 바꿔 말하면 선택적 제약 조건이 포함된 객체 식별자입니다. 자세한 내용은 PostgreSQL 설명서의 [객체 식별자](#)를 참조하세요. 간단히 말해,

lo 데이터 유형을 사용하여 대형 객체 참조가 들어 있는 데이터베이스 열을 다른 객체 식별자(OID)와 구별할 수 있습니다.

- `lo_manage` - 대형 객체 참조가 포함된 테이블 열의 트리거에 사용할 수 있는 함수입니다. 대형 객체를 참조하는 값을 삭제하거나 수정할 때마다 트리거는 객체(`lo_unlink`)를 참조에서 분리합니다. 열이 대형 객체에 대한 유일한 데이터베이스 참조인 경우에만 열에 트리거를 사용합니다.

대형 객체 모듈에 대한 자세한 내용은 PostgreSQL 설명서의 lo를 참조하세요.

lo 확장 설치

lo 확장을 설치하려면 `rds_superuser` 권한이 필요합니다.

lo 확장 설치

1. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 프라이머리 DB 인스턴스에 연결합니다.

```
psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

   메시지가 표시되면 암호를 입력합니다. `psql` 클라이언트는 기본 관리 연결 데이터베이스인 `postgres=>`를 연결하고 프롬프트로 표시합니다.

2. 다음과 같이 확장을 설치합니다.

```
postgres=> CREATE EXTENSION lo;
CREATE EXTENSION
```

이제 `lo` 데이터 유형을 사용하여 테이블의 열을 정의할 수 있습니다. 예를 들어, 래스터 이미지 데이터가 들어 있는 테이블(`images`)을 생성할 수 있습니다. 테이블을 생성하는 다음 예와 같이 열 `raster`에 `lo` 데이터 유형을 사용할 수 있습니다.

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

lo_manage 트리거 함수를 사용하여 객체 삭제

`lo`의 트리거 또는 다른 대형 객체 열에서 `lo_manage` 함수를 사용하여 `lo`가 업데이트되거나 삭제될 때 분리된 객체를 지우거나 발생하지 않도록 방지할 수 있습니다.

대형 객체를 참조하는 열에 트리거 설정

- 다음 중 하나를 수행합니다.
  - 인수의 열 이름을 사용하여 대형 객체에 대한 고유한 참조를 포함하도록 각 열에 BEFORE UPDATE OR DELETE 트리거를 생성합니다.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

  - 열이 업데이트되는 경우에만 트리거를 적용합니다.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

`lo_manage` 트리거 함수는 트리거를 정의하는 방법에 따라 열 데이터를 삽입하거나 삭제하는 맥락에서만 작동합니다. 데이터베이스에서 `DROP` 또는 `TRUNCATE` 작업을 수행할 때는 영향을 미치지 않습니다. 즉, 삭제하기 전에 테이블에서 객체 열을 삭제하여 분리된 객체가 생성되지 않도록 해야 합니다.

예를 들어, `images` 테이블을 포함하는 데이터베이스를 삭제한다고 가정해 봅니다. 다음과 같이 열을 삭제합니다.

```
postgres=> DELETE FROM images COLUMN raster
```

해당 열에 삭제를 처리하는 `lo_manage` 함수가 정의되어 있다는 가정하에 이제 테이블을 안전하게 삭제할 수 있습니다.

vacuumlo를 사용하여 고립된 대형 객체 제거

`vacuumlo` 유틸리티는 데이터베이스에서 고립된 대형 객체를 식별하고 제거할 수 있습니다. 이 유틸리티는 PostgreSQL 9.1.24부터 지원됩니다. 데이터베이스 사용자가 대형 객체로 자주 작업하는 경우 가끔 `vacuumlo`를 실행하여 분리되어 있는 대형 객체를 정리하는 것이 좋습니다.

lo 확장을 설치하기 전에 `vacuumlo`를 사용하여 Aurora PostgreSQL DB 클러스터의 이점을 얻을 수 있는지를 평가할 수 있습니다. 이렇게 하려면 다음과 같이 -n 옵션(no-op)과 함께 `vacuumlo`를 실행하여 제거할 항목을 표시합니다.

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
```

```
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

출력에서 알 수 있듯이 분리된 대형 객체는 이 특정 데이터베이스에서는 문제가 되지 않습니다.

유틸리티에 대한 자세한 내용은 PostgreSQL 설명서의 vacuumlo 섹션을 참조하세요.

vacuumlo 작동 방식 이해

vacuumlo 명령은 사용자 테이블에 영향을 미치거나 충돌하지 않고 PostgreSQL 데이터베이스에서 고립된 대형 객체(LO)를 제거합니다.

명령은 다음과 같이 작동합니다.

1. vacuumlo는 데이터베이스에 있는 대형 객체의 모든 객체 ID(OID)가 포함된 임시 테이블을 만드는 것으로 시작합니다.
2. 그런 다음 vacuumlo는 데이터 유형 oid 또는 lo를 사용하는 데이터베이스의 모든 열을 스캔합니다. vacuumlo가 이러한 열에서 일치하는 OID를 찾으면 임시 테이블에서 OID를 제거합니다. vacuumlo는 이러한 유형을 기반으로 하는 도메인이 아니라 이름이 oid 또는 lo인 열만 검사합니다.
3. 임시 테이블의 나머지 항목은 고립된 LO를 나타내며, vacuumlo가 안전하게 제거합니다.

vacuumlo 성능 개선

-l 옵션을 사용하여 배치 크기를 늘려 vacuumlo의 성능을 잠재적으로 개선할 수 있습니다. 이렇게 하면 vacuumlo가 한 번에 더 많은 LO를 처리할 수 있습니다.

시스템에 충분한 메모리가 있고 메모리에 임시 테이블을 완전히 수용할 수 있는 경우 데이터베이스 수준에서 temp_buffers 설정을 늘리면 성능이 향상될 수 있습니다. 이렇게 하면 테이블이 메모리에 완전히 상주할 수 있으므로 전체 성능이 향상될 수 있습니다.

다음 쿼리는 임시 테이블의 크기를 추정합니다.

```
SELECT
    pg_size_pretty(SUM(pg_column_size(oid))) estimated_lo_temp_table_size
FROM
    pg_largeobject_metadata;
```

대형 객체에 대한 고려 사항

다음은 대형 객체로 작업할 때 유의해야 할 몇 가지 중요한 고려 사항입니다.

- 현재 고립된 LO를 제거하는 다른 방법이 없기 때문에 `Vacuumlo`가 유일한 솔루션입니다.
- pglogical, 네이티브 논리적 복제, 복제 기술을 사용하는 AWS DMS와 같은 도구는 대형 객체 복제를 지원하지 않습니다.
- 데이터베이스 스키마를 설계할 때는 가능하면 대형 객체를 사용하지 말고 `bytea`와 같은 대체 데이터 유형을 사용하는 것이 좋습니다.
- 고립된 LO 관련 문제를 방지하려면 최소 매주 정기적으로 `vacuumlo`를 실행합니다.
- 대형 객체를 저장하는 테이블에서 `lo_manage` 함수와 함께 트리거를 사용하면 고립된 LO가 만들어지는 것을 방지할 수 있습니다.

PostGIS 확장을 사용하여 공간 데이터 관리

PostGIS는 공간 정보를 저장하고 관리하기 위해 PostgreSQL을 확장한 것입니다. PostGIS에 대한 자세한 내용은 PostGIS.net을 참조하세요.

버전 10.5부터 PostgreSQL은 맵 상자 벡터 타일 데이터 작업을 위해 PostGIS에서 사용하는 libprotobuf 1.3.0 라이브러리를 지원합니다.

PostGIS 확장을 설정하려면 `rds_superuser` 권한이 필요합니다. PostGIS 확장 프로그램 및 공간 데이터를 관리할 사용자(역할)를 생성하는 것이 좋습니다. PostGIS 확장 프로그램 및 관련 구성 요소는 PostgreSQL에 수천 개의 함수를 추가합니다. 사용 사례에 적합한 경우 자체 스키마에서 PostGIS 확장 프로그램을 만드는 것이 좋습니다. 다음 예제에서는 자체 데이터베이스에 확장 프로그램을 설치하는 방법을 보여주지만, 필수 사항은 아닙니다.

주제

- 6단계: PostGIS 확장 업그레이드
- PostGIS 확장 버전
- PostGIS 2를 PostGIS 3으로 업그레이드

1단계: PostGIS 확장을 관리할 사용자(역할) 생성

먼저 rds_superuser 권한이 있는 사용자로 RDS for PostgreSQL DB 인스턴스에 연결합니다. 인스턴스를 설정할 때 기본 이름을 유지했다면 postgres로 연결합니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

PostGIS 확장을 관리하기 위해 별도의 역할(사용자)을 생성합니다.

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

역할이 확장을 설치하도록 허용하려면 이 역할 rds_superuser 권한을 부여합니다.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

PostGIS 아티팩트에 사용할 데이터베이스를 만듭니다. 이 단계는 선택 사항입니다. 또는 PostGIS 확장을 위해 사용자 데이터베이스에 스키마를 생성할 수 있지만 이 역시 필수 사항은 아닙니다.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

gis_admin에게 lab_gis 데이터베이스에 대한 모든 권한을 부여합니다.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

세션을 종료하고 다음과 같이 RDS for PostgreSQL DB 인스턴스에 gis_admin으로 다시 연결합니다.

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
```

```
Password for user gis_admin:...
lab_gis=>
```

다음 단계에 설명된 대로 확장 프로그램을 계속 설정합니다.

2단계: PostGIS 확장 모듈을 로드합니다.

PostGIS 확장에는 지리 공간 기능을 제공하기 위해 함께 작동하는 여러 관련 확장이 포함되어 있습니다. 사용 사례에 따라 이 단계에서 만든 확장이 모두 필요하지 않을 수 있습니다.

`CREATE EXTENSION` 문을 사용하여 PostGIS 확장 모듈을 로드합니다.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

다음 예에 표시된 확장 및 해당 소유자가 나열된 SQL 쿼리를 실행하여 결과를 확인할 수 있습니다.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

3단계: 확장 스키마 소유권 이전

ALTER SCHEMA 문을 사용하여 스키마 소유권을 gis_admin 역할로 이전합니다.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

다음 SQL 쿼리를 실행하여 소유권 변경을 확인할 수 있습니다. 또는 psql 명령줄에서 \dn 메타 명령을 사용할 수 있습니다.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

4단계: PostGIS 테이블 소유권 이전

> **Note**
>
> PostGIS 함수의 소유권을 변경하지 마세요. PostGIS를 올바르게 운영하고 향후 업그레이드하려면 이러한 함수가 원래 소유권을 유지해야 합니다. PostGIS 권한에 대한 자세한 내용은 PostgreSQL 보안을 참조하세요.

다음 함수를 사용하여 PostGIS 테이블 소유권을 gis_admin 역할로 이전합니다. psql 프롬프트에서 다음 문을 실행하여 함수를 생성합니다.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

그런 다음 쿼리로 exec 함수를 실행하면 함수가 해당하는 문을 실행하여 권한을 변경합니다.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

5단계: 확장 모듈을 테스트합니다.

스키마 이름을 지정할 필요가 없도록 하려면 다음 명령을 사용하여 검색 경로에 tiger 스키마를 추가하세요.

```
SET search_path=public,tiger;
SET
```

다음 SELECT 문을 사용하여 tiger 스키마를 테스트합니다.

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

이 확장에 대한 자세한 내용은 PostGIS 문서에서 Tiger Geocoder를 참조하세요.

다음 SELECT 문을 사용하여 topology 스키마를 테스트합니다. 이렇게 하면 지정된 공간 참조 식별자(26986) 및 기본 허용 오차(0.5)를 사용하여 새 토폴로지 개체(my_new_topo)를 등록하는 createtopology 함수를 호출합니다. 자세한 내용은 PostgreSQL의 CreateTopology 문서를 참조하세요.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
```

```
----------------
             1
(1 row)
```

6단계: PostGIS 확장 업그레이드

PostgreSQL의 각 새 릴리스는 해당 릴리스와 호환되는 하나 이상의 PostGIS 확장 버전을 지원합니다. PostgreSQL 엔진을 새 버전으로 업그레이드해도 PostGIS 확장은 자동으로 업그레이드되지 않습니다. PostgreSQL 엔진을 업그레이드하려면 일반적으로 PostGIS를 현재 PostgreSQL 버전에서 사용 가능한 최신 버전으로 업그레이드를 먼저 해야 합니다. 자세한 내용은 PostGIS 확장 버전 단원을 참조하세요.

PostgreSQL 엔진 업그레이드 후 PostGIS 확장을 새로 업그레이드된 PostgreSQL 엔진 버전에 대해 지원되는 버전으로 다시 업그레이드합니다. PostgreSQL 엔진 업그레이드에 대한 자세한 내용은 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트 단원을 따라가세요.

Aurora PostgreSQL DB 클러스터 에서 사용 가능한 PostGIS 확장 버전 업데이트를 언제든지 확인할 수 있습니다. 이렇게 하려면 다음 명령을 실행합니다. 이 기능은 PostGIS 2.5.0 이상 버전에서 사용할 수 있습니다.

```
SELECT postGIS_extensions_upgrade();
```

애플리케이션이 최신 PostGIS 버전을 지원하지 않는 경우에도 다음과 같이 사용 중인 메이저 버전에서 사용할 수 있는 이전 버전의 PostGIS를 설치할 수 있습니다.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

이전 버전에서 특정 PostGIS 버전으로 업그레이드하려는 경우 다음 명령을 사용할 수도 있습니다.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

업그레이드하는 버전에 따라 이 기능을 다시 사용해야 할 수도 있습니다. 기능의 첫 번째 실행 결과에 따라 추가 업그레이드 기능이 필요한지 여부가 결정됩니다. PostGIS 2에서 PostGIS 3으로 업그레이드하는 경우를 예로 들 수 있습니다. 자세한 내용은 PostGIS 2를 PostGIS 3으로 업그레이드을 참조하세요.

PostgreSQL 엔진의 메이저 버전 업그레이드를 준비하기 위해 이 확장을 업그레이드했다면 다른 예비 작업을 계속할 수 있습니다. 자세한 내용은 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트 단원을 참조하세요.

PostGIS 확장 버전

Aurora PostgreSQL 릴리스 정보의 Aurora PostgreSQL 호환 버전용 확장 버전에 나열된 PostGIS와 같은 모든 확장 버전을 설치하는 것이 좋습니다. Amazon RDS for PostgreSQL 릴리스 정보의 릴리스에서 사용 가능한 버전 목록을 얻으려면 다음 명령을 사용하세요.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

Aurora PostgreSQL 릴리스 정보의 다음 단원에서 버전 정보를 찾을 수 있습니다.

- Aurora PostgreSQL 확장 버전 14
- Aurora PostgreSQL 호환 버전의 확장 버전 13
- Aurora PostgreSQL 호환 버전의 확장 버전 12
- Aurora PostgreSQL 호환 버전의 확장 버전 11
- Aurora PostgreSQL 호환 버전의 확장 버전 10
- Aurora PostgreSQL 호환 버전의 확장 버전 9.6

PostGIS 2를 PostGIS 3으로 업그레이드

버전 3.0부터 PostGIS 래스터 기능이 이제 별도의 확장인 `postgis_raster`입니다. 이 확장에는 자체 설치 및 업그레이드 경로가 있습니다. 이러한 경로를 통해 코어에서 핵심 `postgis` 확장에서 래스터 이미지 처리에 필요한 수십 가지 기능, 데이터 유형 및 기타 아티팩트를 제거합니다. 즉, 사용 사례에 래스터 처리가 필요하지 않은 경우 `postgis_raster` 확장을 설치할 필요가 없습니다.

다음 업그레이드 예에서 첫 번째 업그레이드 명령은 래스터 기능을 `postgis_raster` 확장으로 추출합니다. 그런 다음 `postgis_raster`를 새 버전으로 업그레이드하려면 두 번째 업그레이드 명령이 필요합니다.

PostGIS 2에서 PostGIS 3으로 업그레이드하려면

1. Aurora PostgreSQL DB 클러스터의 PostgreSQL 버전에 사용할 수 있는 PostGIS의 기본 버전을 식별합니다. 이렇게 하려면 다음 쿼리를 실행합니다.

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name    | default_version | installed_version |                          comment
---------+-----------------+-------------------
+---------------------------------------------------------------
```

```
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에서 각 데이터베이스에 설치된 PostGIS 버전을 식별합니다. 즉, 다음과 같이 각 사용자 데이터베이스를 쿼리합니다.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name    | Version | Schema |                             Description
---------+---------+--------
+-------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

기본 버전(PostGIS 3.1.4)과 설치된 버전(PostGIS 2.3.7) 간의 이러한 불일치는 PostGIS 확장을 업그레이드해야 함을 의미합니다.

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. 다음 쿼리를 실행하여 래스터 기능이 이제 자체 패키지에 포함되어 있는지 확인합니다.

```
SELECT
    probin,
    count(*)
FROM
```

```
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

출력 결과를 통해 여전히 버전 간에 여전히 차이가 있음을 알 수 있습니다. PostGIS 함수는 버전 3(postgis-3)이고 래스터 함수(rtpostgis)는 버전 2(rtpostgis-2.3)입니다. 업그레이드를 완료하려면 다음과 같이 upgrade 명령을 다시 실행합니다.

```
postgres=> SELECT postgis_extensions_upgrade();
```

경고 메시지는 무시해도 됩니다. 다음 쿼리를 다시 실행하여 업그레이드가 완료되었는지 확인합니다. PostGIS 및 모든 관련 확장 프로그램이 업그레이드 필요 항목으로 표시되지 않으면 업그레이드가 완료된 것입니다.

```
SELECT postgis_full_version();
```

4. 다음 쿼리를 사용하여 완료된 업그레이드 프로세스와 별도로 패키징된 확장을 확인하고 해당 버전이 일치하는지 확인합니다.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                              Description
```

```
----------------+---------+--------
+-------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

출력은 PostGIS 2 확장이 PostGIS 3으로 업그레이드되었으며 `postgis`와 현재 별도의 `postgis_raster` 확장이 모두 버전 3.1.5임을 보여줍니다.

이 업그레이드가 완료된 후 래스터 기능을 사용하지 않으려는 경우 다음과 같이 확장을 삭제할 수 있습니다.

```
DROP EXTENSION postgis_raster;
```

pg_partman 확장자를 사용하여 PostgreSQL 파티션 관리하기

PostgreSQL 테이블 파티셔닝은 데이터 입력 및 보고의 고성능 처리를 위한 프레임워크를 제공합니다. 대량의 데이터를 매우 빠르게 입력해야 하는 데이터베이스의 경우 파티셔닝을 사용합니다. 파티셔닝은 큰 테이블의 빠른 쿼리를 제공합니다. 파티셔닝은 I/O 리소스가 적기 때문에 데이터베이스 인스턴스에 영향을 주지 않고 데이터를 유지 관리하는 데 도움이 됩니다.

파티셔닝을 사용하면 데이터를 사용자 지정 크기의 청크로 분할하여 처리할 수 있습니다. 예를 들어, 시간별, 일별, 주별, 월별, 분기별, 연도별, 사용자 지정 또는 이러한 조합과 같은 범위의 시계열 데이터를 분할할 수 있습니다. 시계열 데이터 예제의 경우 테이블을 시간별로 분할한 경우 각 파티션에는 1시간의 데이터가 포함됩니다. 시계열 테이블을 일별로 분할한 경우 파티션에는 하루 분량의 데이터가 저장되는 식입니다. 파티션 키는 파티션의 크기를 제어합니다.

분할된 테이블에서 `INSERT` 또는 `UPDATE` SQL 명령을 사용하는 경우 데이터베이스 엔진은 데이터를 적절한 파티션으로 라우팅합니다. 데이터를 저장하는 PostgreSQL 테이블 파티션은 기본 테이블의 하위 테이블입니다.

데이터베이스 쿼리를 읽는 동안 PostgreSQL 최적화 프로그램은 쿼리 `WHERE` 절을 검사하고 가능한 경우 데이터베이스 스캔을 관련 파티션에만 보냅니다.

버전 10부터, PostgreSQL은 선언적 분할을 사용하여 테이블 분할을 구현합니다. 이를 네이티브 PostgreSQL 파티셔닝이라고도 합니다. PostgreSQL 버전 10 이전에는 트리거를 사용하여 파티션을 구현했습니다.

PostgreSQL 테이블 분할은 다음과 같은 기능을 제공합니다:

- 언제든지 새 파티션을 만들 수 있습니다.
- 가변 분할 영역 범위입니다.
- 데이터 정의 언어(DDL) 문을 사용하여 분리 및 재연결 가능한 파티션

  예를 들어 분리 가능한 파티션은 주 파티션에서 기록 데이터를 제거하지만 분석을 위해 기록 데이터를 유지하는 데 유용합니다.
- 새 파티션은 다음을 포함하여 상위 데이터베이스 테이블 속성을 상속합니다.
  - 인덱스
  - 파티션 키 열을 포함해야 하는 기본 키
  - 외래 키
  - 제약 점검
  - 참조
- 전체 테이블 또는 각 특정 파티션에 대한 인덱스 생성

개별 파티션에 대한 스키마를 변경할 수 없습니다. 그러나 분할 영역에 전파되는 상위 테이블 (예: 새 열 추가) 을 변경할 수 있습니다.

주제

- PostgreSQL pg_partman 확장 개요
- pg_partman 확장 활성화
- create_parent 함수를 사용하여 파티션 구성
- run_maintenance_proc 함수를 사용하여 파티션 유지 관리 구성

PostgreSQL pg_partman 확장 개요

PostgreSQL `pg_partman` 확장을 사용하여 테이블 파티션의 생성 및 유지 관리를 자동화할 수 있습니다. 자세한 내용은 `pg_partman` 문서의 PG 파티션 관리자를 참조하세요.

> **Note**
>
> `pg_partman` 확장은 Aurora PostgreSQL 버전 12.6 이상에서 지원됩니다.

각 파티션을 수동으로 생성하지 않고 다음 설정으로 `pg_partman`을 구성합니다.

- 분할할 테이블
- 파티션 유형
- 파티션 키
- 분할 영역 세분성
- 파티션 사전 생성 및 관리 옵션

PostgreSQL 분할 테이블을 만든 후, `create_parent` 함수를 호출하여 `pg_partman`에 등록합니다. 이렇게 하면 함수에 전달하는 파라미터를 기반으로 필요한 파티션이 생성됩니다.

`pg_partman` 확장은 예약된 시간에 호출하여 파티션을 자동으로 관리할 수 있는 `run_maintenance_proc` 함수도 제공합니다. 필요에 따라 적절한 파티션이 생성되도록 하려면 이 기능을 주기적으로 실행하도록 예약합니다(예: 매 시간). 파티션이 자동으로 삭제되게 할 수도 있습니다.

pg_partman 확장 활성화

파티션을 관리하려는 동일한 PostgreSQL DB 인스턴스 내에 여러 개의 데이터베이스가 있는 경우 각 데이터베이스별로 별도로 `pg_partman` 확장을 활성화해야 합니다. 특정 데이터베이스에 대해 `pg_partman` 확장을 활성화하려면 파티션 유지 관리 스키마를 생성한 후 다음과 같이 `pg_partman` 확장을 생성합니다.

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> `pg_partman` 확장을 만들려면 `rds_superuser` 권한이 있는지 확인하세요.

다음과 같은 오류가 발생하면 계정에 `rds_superuser` 권한을 부여하거나 수퍼유저 계정을 사용하십시오.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

`rds_superuser` 권한을 부여하려면 수퍼유저 계정으로 연결하고 다음 명령을 실행합니다.

```
GRANT rds_superuser TO user-or-role;
```

pg_partman 확장 사용 방법을 보여 주는 예제에서는 다음 예제 데이터베이스 테이블 및 파티션을 사용합니다. 이 데이터베이스는 타임스탬프를 기반으로 분할된 테이블을 사용합니다. data_mart 스키마에는 라는 열이 events 있는 테이블이 포함되어 created_at있습니다. 이 events 테이블에는 다음과 같은 설정이 포함되어 있습니다.

- 기본 키 event_id 및 created_at파티션을 안내하는 데 사용되는 열이 있어야 합니다.
- ck_valid_operation 테이블 열에 대한 값을 operation 적용하는 CHECK 제약 조건.
- 두 개의 외래 키. 하나는 외부 테이블을fk_orga_membership) organization 가리키고 다른 하나는 자체 참조 외래 키입니다.fk_parent_event_id
- 두 개의 인덱스. 여기서 하나 (idx_org_id) 는 외래 키용이고 다른 인덱스 (idx_event_type) 는 이벤트 유형용입니다.

다음 DDL 명령문은 각 파티션에 자동으로 포함되는 이러한 객체를 만듭니다.

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);
```

```
CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

create_parent 함수를 사용하여 파티션 구성

pg_partman 확장을 활성화한 후에는 create_parent 함수를 사용하여 파티션 유지 관리 스키마 내에 파티션을 구성합니다. 다음 예에서는 events에서 만든 pg_partman 확장 활성화 테이블 예제를 사용합니다. create_parent 함수를 다음과 같이 호출합니다.

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

파라미터는 다음과 같습니다.

- p_parent_table – 상위 분할 테이블입니다. 이 테이블은 이미 존재해야 하며 스키마를 포함하여 정규화되어야 합니다.
- p_control – 분할이 기반으로되는 열입니다. 데이터 유형은 정수 또는 시간 기반이어야 합니다.
- p_type - 유형은 'range' 또는 'list'입니다.
- p_interval – 각 파티션에 대한 시간 간격 또는 정수 범위입니다. 값의 예로는 1 day, 1 hour 등이 포함됩니다.
- p_premake – 새 삽입을 지원하기 위해 미리 생성할 분할 영역 수입니다.

create_parent 함수에 대한 자세한 설명은 pg_partman 설명서에서 생성 함수를 참조하세요.

run_maintenance_proc 함수를 사용하여 파티션 유지 관리 구성

파티션 유지 관리 작업을 실행하여 자동으로 새 파티션을 생성하거나, 파티션을 분리하거나, 이전 파티션을 제거할 수 있습니다. 파티션 유지 관리에는 내부 스케줄러를 시작하는 pg_partman 확장 및 pg_cron 확장의 run_maintenance_proc 함수가 사용됩니다. pg_cron 스케줄러는 데이터베이스에 정의된 SQL 문, 함수 및 프로시저를 자동으로 실행합니다.

다음 예제에서는 에서 만든 events 테이블 예제를 pg_partman 확장 활성화 사용하여 파티션 유지 관리 작업이 자동으로 실행되도록 설정합니다. 전제 조건으로 pg_cron을 DB 인스턴스의 파라미터 그룹의 shared_preload_libraries 파라미터에 추가합니다.

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

다음으로, 앞의 예제에 대한 단계별 설명을 찾을 수 있습니다.

1. DB 인스턴스와 연결된 파라미터 그룹을 수정하고 pg_cron 파라미터 값에 shared_preload_libraries 추가합니다. 이 변경 사항을 적용하려면 DB 인스턴스를 다시 시작해야 합니다. 자세한 내용은 [Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정](#) 섹션을 참조하세요.
2. CREATE EXTENSION pg_cron; 권한이 있는 계정을 사용하여 rds_superuser 명령을 실행합니다. 이렇게 하면 pg_cron 확장이 활성화됩니다. 자세한 내용은 [PostgreSQL pg_cron 확장을 사용하여 유지 관리 예약](#) 섹션을 참조하세요.
3. UPDATE partman.part_config 명령을 실행하여 data_mart.events 테이블에 대한 pg_partman 설정을 조정합니다.
4. SET 명령을 실행합니다. data_mart.events 테이블을 구성하려면 다음 절을 함께 사용합니다.
   a. infinite_time_partitions = true, – 제한 없이 자동으로 새 파티션을 만들 수 있도록 테이블을 구성합니다.
   b. retention = '3 months', – 최대 보존 기간이 3개월이 되도록 테이블을 구성합니다.
   c. retention_keep_table=true – 보존 기간이 만료될 때 테이블이 자동으로 삭제되지 않도록 테이블을 구성합니다. 대신 보존 기간보다 오래된 파티션은 상위 테이블에서만 분리됩니다.
5. SELECT cron.schedule 명령을 실행합니다. pg_cron 함수를 호출하려면 이 호출은 스케줄러가 pg_partman 유지 관리 프로시저 partman.run_maintenance_proc를 실행하는 빈도를 정의합니다. 이 예제에서는 프로시저가 매 시간 실행됩니다.

run_maintenance_proc 함수에 대한 자세한 설명은 pg_partman 설명서에서 [유지 관리 함수](#)를 참조하세요.

PostgreSQL pg_cron 확장을 사용하여 유지 관리 예약

PostgreSQL pg_cron 확장을 사용하여 PostgreSQL 데이터베이스 내에서 유지 관리 명령을 예약할 수 있습니다. 확장에 대한 자세한 내용은 pg_cron 설명서의 [pg_cron이란 무엇입니까?](#) 단원을 참조하십시오.

pg_cron 확장은 Aurora PostgreSQL 엔진 버전 12.6 이상에서 지원됩니다.

pg_cron 사용에 대한 자세한 내용은 [RDS for PostgreSQL 또는 Aurora PostgreSQL 호환 에디션 데이터베이스에서 pg_cron을 사용하여 작업 예약](#)을 참조하세요.

주제

pg_cron 확장 설정

다음과 같이 pg_cron 확장을 설정합니다.

1. shared_preload_libraries 파라미터 값에 pg_cron 을 추가하여 PostgreSQL DB 인스턴스와 연결된 사용자 지정 파라미터 그룹을 수정합니다.

   파라미터 그룹에 대한 변경 사항을 적용하려면 PostgreSQL DB 인스턴스를 다시 시작합니다. 파라미터 그룹 사용에 대한 자세한 내용은 Amazon Aurora PostgreSQL parameters. 단원을 참조하십시오.

2. PostgreSQL DB 인스턴스가 다시 시작된 후 rds_superuser 권한이 있는 계정을 사용하여 다음 명령을 수행합니다. 예를 들어 Aurora PostgreSQL DB 클러스터를 생성할 때 기본 설정을 사용한 경우 사용자 postgres로 연결하고 확장을 생성합니다.

   ```
   CREATE EXTENSION pg_cron;
   ```

   pg_cron 스케줄러는 postgres라는 기본 PostgreSQL 데이터베이스에 설정됩니다. pg_cron 객체가 이 postgres 데이터베이스에 생성되고 모든 예약 작업이 이 데이터베이스에서 실행됩니다.

3. 기본 설정을 사용하거나 PostgreSQL DB 인스턴스 내의 다른 데이터베이스에서 실행되도록 작업을 예약할 수 있습니다. PostgreSQL DB 인스턴스 내의 다른 데이터베이스에 대한 작업을 예약하려면 기본 데이터베이스 이외의 데이터베이스에 대한 cron 작업 예약의 예제를 참조하세요.

데이터베이스 사용자에게 pg_cron 사용 권한 부여

pg_cron 확장을 설치하려면 rds_superuser 권한이 필요합니다. 그러나 pg_cron 사용 권한은 (rds_superuser 그룹/역할의 구성원에 의해) 다른 데이터베이스 사용자에게 부여되어 자신의 작업을 예약할 수 있습니다. 프로덕션 환경에서 작업을 개선하는 경우 필요한 경우에만 cron 스키마에 권한을 부여하는 것이 좋습니다.

cron 스키마에서 데이터베이스 사용자 권한을 부여하려면 다음 명령을 실행합니다.

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

이렇게 하면 액세스 권한이 있는 개체에 대해 cron 작업을 예약하기 위해 cron 스키마에 액세스할 수 있는 권한이 *db-user*에게 부여됩니다. 데이터베이스 사용자에게 권한이 없으면 다음과 같이 postgresql.log 파일에 오류 메시지를 게시한 후 작업이 실패합니다.

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

즉, cron 스키마에 대한 사용 권한이 부여된 데이터베이스 사용자는 예약하려는 개체(테이블, 스키마 등)에 대한 사용 권한도 있어야 합니다.

cron 작업 및 성공 또는 실패에 대한 세부 정보도 cron.job_run_details 테이블에 캡처됩니다. 자세한 내용은 작업 예약 및 상태 캡처를 위한 테이블 섹션을 참조하세요.

pg_cron 작업 예약

다음 단원에서는 pg_cron 작업을 사용하여 다양한 관리 태스크를 예약하는 방법을 보여 줍니다.

> **Note**
>
> pg_cron 작업을 생성할 때 max_worker_processes 설정이 cron.max_running_jobs 개수보다 큰지 확인하십시오. 백그라운드 작업자 프로세스가 부족하면 pg_cron 작업은 실패합니다. 기본 pg_cron 작업 수는 5입니다. 자세한 내용은 pg_cron 확장을 관리하기 위한 파라미터 단원을 참조하십시오.

주제

테이블 베큠

Autovacuum은 대부분의 경우 정리 유지 관리를 처리합니다. 하지만 선택한 시간에 특정 테이블을 베큠하도록 예약해야 하는 경우도 있습니다.

다음은 매일 22:00(GMT)에 특정 테이블에 `cron.schedule`를 사용하는 작업을 설정하는 VACUUM FREEZE 함수의 사용 예입니다.

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

앞의 예제를 실행한 후, 다음과 같이 `cron.job_run_details` 테이블에서 기록을 확인할 수 있습니다.

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                           |
 status     | return_message | start_time                         | end_time
-------+-------+---------+----------+----------+------------------------------------
+-----------+----------------+-------------------------------
+-------------------------------
 1      | 1      | 3395     | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

다음은 실패한 작업을 확인하기 위한 `cron.job_run_details` 테이블의 쿼리입니다.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                       | status
 | return_message                                    | start_time                  |
 end_time
```

```
------+-------+---------+----------+----------+------------------------------+--------
+-----------------------------------------------+-----------------------------
+------------------------------
 5    | 4      | 30339    | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

자세한 내용은 작업 예약 및 상태 캡처를 위한 테이블 섹션을 참조하세요.

pg_cron 기록 테이블 지우기

cron.job_run_details 테이블에는 시간이 지남에 따라 매우 커질 수 있는 cron 작업 기록이 포함되어 있습니다. 이 테이블을 지우는 작업을 예약하는 것이 좋습니다. 예를 들어, 1 주일 분량의 항목을 보관하면 문제 해결을 위해 충분할 수 있습니다.

다음 예에서는 cron.schedule 함수를 사용하여, 매일 자정에 실행되어 cron.job_run_details 테이블을 지우는 작업을 예약합니다. 이 작업은 지난 7 일 동안의 항목만 유지합니다. rds_superuser 계정을 사용하여 다음과 같은 작업을 예약합니다.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

자세한 내용은 작업 예약 및 상태 캡처를 위한 테이블 섹션을 참조하세요.

postgresql.log 파일에만 오류 로깅

cron.job_run_details 테이블에 대한 쓰기 작업을 완전히 차단하려면 PostgreSQL DB 인스턴스와 연결된 파라미터 그룹을 수정하고 cron.log_run 파라미터를 '꺼짐'으로 설정합니다. pg_cron 확장이 더 이상 테이블에 기록하지 않고 postgresql.log 파일에만 오류를 캡처합니다. 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

다음 명령을 사용하여 cron.log_run 파라미터 값을 확인합니다.

```
postgres=> SHOW cron.log_run;
```

자세한 내용은 pg_cron 확장을 관리하기 위한 파라미터 섹션을 참조하세요.

기본 데이터베이스 이외의 데이터베이스에 대한 cron 작업 예약

pg_cron의 메타데이터는 모두 postgres라는 PostgreSQL 기본 데이터베이스에 보관됩니다. 백그라운드 작업자는 유지 관리 cron 작업을 실행하는 데 사용되므로 PostgreSQL DB 인스턴스 내의 모든 데이터베이스에서 작업을 예약할 수 있습니다.

> **Note**
>
> rds_superuser 역할 또는 rds_superuser 권한이 있는 사용자만 데이터베이스의 모든 cron 작업을 나열할 수 있습니다. 다른 사용자는 cron.job 테이블에서 자신의 작업만 볼 수 있습니다.

1. cron 데이터베이스에서 정상적으로 cron.schedule을(를) 사용하는 것처럼 작업을 예약합니다.

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
 freeze test_table');
```

2. rds_superuser 역할을 가진 사용자는 방금 생성한 작업에 대한 데이터베이스 열을 업데이트하여 PostgreSQL DB 인스턴스 내의 다른 데이터베이스에서 실행되도록 합니다.

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. cron.job 테이블을 쿼리하여 확인합니다.

```
postgres=> SELECT * FROM cron.job;
jobid | schedule     | command                          | nodename   | nodeport |
 database | username   | active | jobname
------+-------------+---------------------------------+-----------+----------
+----------+-----------+--------+------------------------
106   | 29 03 * * * | vacuum freeze test_table        | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> 경우에 따라, 다른 데이터베이스에서 실행하고자 하는 cron 작업을 추가할 수 있습니다. 이 경우, 올바른 데이터베이스 열을 업데이트하기 전에 작업이 기본 데이터베이스(postgres)에서

실행을 시도할 수 있습니다. 사용자 이름에 권한이 있으면 작업이 기본 데이터베이스에서 성공적으로 실행됩니다.

pg_cron 확장에 대한 참조

pg_cron 확장과 함께 다음 파라미터, 함수 및 테이블을 사용할 수 있습니다. 자세한 내용은 pg_cron 설명서의 pg_cron이란 무엇입니까? 단원을 참조하세요.

주제

- pg_cron 확장을 관리하기 위한 파라미터
- 함수 참조: cron.schedule
- 함수 참조: cron.unschedule
- 작업 예약 및 상태 캡처를 위한 테이블

pg_cron 확장을 관리하기 위한 파라미터

다음은 pg_cron 확장 동작을 제어하는 파라미터의 목록입니다.

파라미터	설명
cron.database_name	pg_cron 메타데이터가 보관되는 데이터베이스입니다.
cron.host	PostgreSQL에 연결할 호스트 이름입니다. 이 값은 수정할 수 없습니다.
cron.log_run	job_run_details 테이블에서 실행되는 모든 작업을 로깅합니다. 유효한 값은 on 또는 off입니다. 자세한 내용은 작업 예약 및 상태 캡처를 위한 테이블 섹션을 참조하세요.
cron.log_statement	모든 cron 문을 실행하기 전에 기록합니다. 유효한 값은 on 또는 off입니다.
cron.max_running_jobs	동시에 실행할 수 있는 최대 작업 수입니다.

파라미터	설명
cron.use_background_workers	클라이언트 세션 대신 백그라운드 작업자를 사용합니다. 이 값은 수정할 수 없습니다.

다음 SQL 명령을 사용하여 이러한 파라미터와 해당 값을 표시합니다.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

함수 참조: cron.schedule

이 함수는 cron 작업을 예약합니다. 작업은 처음에 기본 postgres 데이터베이스에서 예약됩니다. 이 함수는 작업 식별자를 나타내는 bigint 값을 반환합니다. PostgreSQL DB 인스턴스 내의 다른 데이터베이스에서 작업이 실행되도록 예약하려면 기본 데이터베이스 이외의 데이터베이스에 대한 cron 작업 예약의 예제를 참조하세요.

이 함수에는 두 가지 구문 형식이 있습니다.

구문

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

파라미터

파라미터	설명
job_name	cron 작업의 이름입니다.
schedule	cron 작업의 일정을 나타내는 텍스트입니다. 형식은 표준 cron 형식입니다.

파라미터	설명
command	실행할 명령의 텍스트입니다.

예시

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

함수 참조: cron.unschedule

이 함수는 cron 작업을 삭제합니다. job_name 또는 job_id를 지정할 수 있습니다. 정책은 사용자가 작업 일정을 제거할 수 있는 소유자인지를 확인합니다. 이 함수는 성공 또는 실패를 나타내는 부울 값을 반환합니다.

함수의 구문 형식은 다음과 같습니다.

구문

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

파라미터

파라미터	설명
job_id	cron 작업이 예약된 경우 cron.schedule 함수에서 반환된 작업 식별자입니다.

파라미터	설명
job_name	cron.schedule 함수로 예약된 cron 작업의 이름입니다.

예시

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

작업 예약 및 상태 캡처를 위한 테이블

다음 표는 cron 작업을 예약하고 작업 완료 방법을 기록하는 데 사용됩니다.

표	설명
cron.job	예약된 각 작업에 대한 메타데이터를 포함합니다. 이 테이블과의 대부분의 상호 작용은 cron.schedule 및 cron.unschedule 함수를 사용하여 수행해야 합니다. **Important** 이 테이블에 직접 업데이트 또는 삽입 권한을 부여하지 않는 것이 좋습니다. 이렇게 하면 사용자가 username(으)로 실행되도록 rds-superuser 열을 업데이트할 수 있습니다.

표	설명
`cron.job_run_details`	이전에 예약된 작업 실행에 대한 기록 정보를 포함합니다. 이는 실행한 작업에서 상태, 반환 메시지, 시작 및 종료 시간을 조사하는 데 유용합니다. Note 이 테이블이 무한정 증가하지 않게 하려면 정기적으로 삭제하세요. 관련 예제는 pg_cron 기록 테이블 지우기 섹션을 참조하세요

pgAudit를 사용하여 데이터베이스 활동 로깅

금융 기관, 정부 기관 및 많은 업계에서는 규제 요구 사항을 충족하기 위해 감사 로그를 보관해야 합니다. Aurora PostgreSQL DB 클러스터와 함께 PostgreSQL Audit 확장(pgAudit)을 사용하면 감사자가 일반적으로 필요로 하거나 규제 요구 사항을 충족하는 데 필요한 세부 레코드를 캡처할 수 있습니다. 예를 들어 특정 데이터베이스 및 테이블의 변경 내용을 추적하고 변경한 사용자 및 기타 여러 세부 정보를 기록하도록 pgAudit 확장을 설정할 수 있습니다.

pgAudit 확장은 네이티브 PostgreSQL 로깅 인프라의 기능을 기반으로 로그 메시지를 더 자세히 확장한 것입니다. 즉, 다른 로그 메시지를 보는 것과 동일한 접근 방식을 사용하여 감사 로그를 볼 수 있습니다. PostgreSQL 로깅에 대한 자세한 내용은 Aurora PostgreSQL 데이터베이스 로그 파일 섹션을 참조하세요.

pgAudit 확장은 일반 텍스트 암호와 같은 민감한 데이터를 로그에서 삭제합니다. Aurora PostgreSQL DB 클러스터가 Aurora PostgreSQL DB 클러스터에 쿼리 로깅을 활성화합니다.에 설명된 대로 데이터 조작 언어(DML) 문을 로깅하도록 구성된 경우 PostgreSQL Audit 확장을 사용하여 일반 텍스트 암호 문제를 방지할 수 있습니다.

매우 구체적으로 데이터베이스 인스턴스에 대한 감사를 구성할 수 있습니다. 모든 데이터베이스와 모든 사용자를 감사할 수 있습니다. 또는 특정 데이터베이스, 사용자 및 기타 객체만 감사하도록 선택할 수 있습니다. 특정 사용자 및 데이터베이스를 감사에서 명시적으로 제외할 수도 있습니다. 자세한 내용은 감사 로깅에서 사용자 또는 데이터베이스 제외 단원을 참조하십시오.

캡처할 수 있는 세부 정보의 양을 고려하여 pgAudit를 사용하는 경우 스토리지 사용량을 모니터링하는 것이 좋습니다.

pgAudit 확장은 사용 가능한 모든 Aurora PostgreSQL 버전에서 지원됩니다. Aurora PostgreSQL 버전에서 지원되는 pgAudit 버전 목록을 보려면 Aurora PostgreSQL 릴리스 정보의 Extension versions for Amazon Aurora PostgreSQL(Amazon Aurora PostgreSQL 확장 버전)을 참조하세요.

주제

pgAudit 확장 설정

Aurora PostgreSQL DB 클러스터에 pgAudit 확장을 설정하려면 먼저 Aurora PostgreSQL DB 클러스터용 사용자 지정 DB 클러스터 파라미터 그룹의 공유 라이브러리에 pgAudit을 추가해야 합니다. 사용자 지정 DB 클러스터 파라미터 그룹을 만드는 방법에 관한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요. 다음으로 pgAudit 확장을 설치합니다. 마지막으로, 감사하려는 데이터베이스 및 객체를 지정합니다. 이 섹션에 절차가 설명되어 있습니다. AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

이 모든 작업을 수행하려면 `rds_superuser` 역할의 권한이 있어야 합니다.

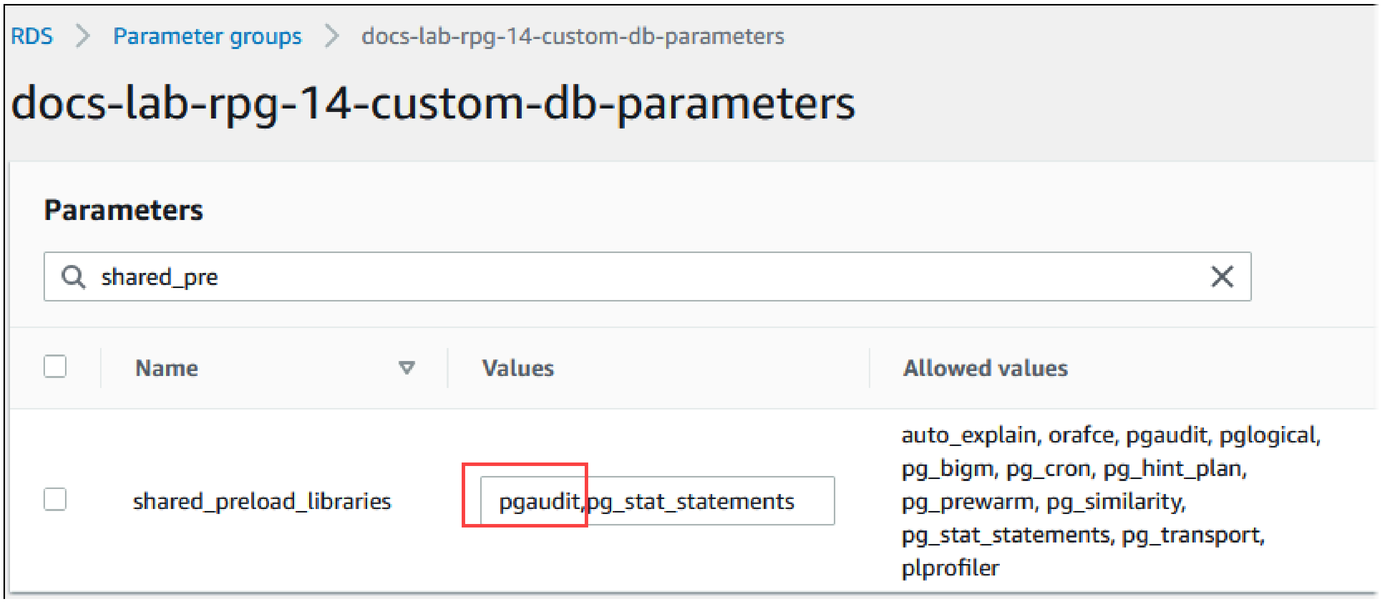
다음 단계에서는 사용자의 Aurora PostgreSQL DB 클러스터가 사용자 지정 DB 클러스터에 연결되어 있다고 가정합니다.

콘솔

pgAudit 확장 설정 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택합니다.
3. Aurora PostgreSQL DB 클러스터 라이터 인스턴스의 구성 탭을 엽니다. 인스턴스 세부 정보 중에서 파라미터 그룹 링크를 찾습니다.
4. 링크를 선택하여 Aurora PostgreSQL DB 클러스터와 연결된 사용자 지정 파라미터를 엽니다.
5. 파라미터 검색 필드에 `shared_pre`를 입력하여 `shared_preload_libraries` 파라미터를 찾습니다.

6. 파라미터 편집을 선택하여 속성 값에 액세스합니다.
7. 값 필드의 목록에 `pgaudit`를 추가합니다. 쉼표를 사용하여 값 목록에서 항목을 구분합니다.



8. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 `shared_preload_libraries` 파라미터 변경 사항이 적용되도록 합니다.
9. 인스턴스를 사용할 수 있게 되면 pgAudit가 초기화되었는지 확인합니다. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. pgAudit가 초기화되었으므로 이제 확장을 생성할 수 있습니다. 확장은 라이브러리를 초기화한 후에 생성해야 합니다. `pgaudit` 확장이 데이터 정의 언어(DDL) 문 감사를 위한 이벤트 트리거를 설치하기 때문입니다.

```
CREATE EXTENSION pgaudit;
```

11. `psql` 세션을 닫습니다.

```
labdb=> \q
```

12. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.

13. 목록에서 `pgaudit.log` 파라미터를 찾아 사용 사례에 적합한 값으로 설정합니다. 예를 들어, 다음 이미지처럼 `pgaudit.log` 파라미터를 `write`로 설정하면 로그에 대한 삽입, 업데이트, 삭제 및 기타 유형의 변경 사항이 캡처됩니다.

`pgaudit.log` 파라미터에 다음 값 중 하나를 선택할 수도 있습니다.

- 없음 - 기본값입니다. 데이터베이스 변경 사항이 로깅되지 않습니다.
- 모두 - 모든 항목(읽기, 쓰기, 함수, 역할, ddl, 기타)을 로깅합니다.
- ddl - `ROLE` 클래스에 포함되지 않은 모든 데이터 정의 언어(DDL) 문을 로깅합니다.
- 함수 - 함수 호출 및 `DO` 블록을 로깅합니다.
- 기타 – 기타 명령(예: `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, `SET`)을 로깅합니다.
- 읽기 – 원본이 관계(예: 테이블) 또는 쿼리인 경우 `SELECT` 및 `COPY`를 로깅합니다.
- 역할 - 역할 및 권한과 관련된 문을 로깅합니다(예: `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, `DROP ROLE`).
- 쓰기 - 대상이 관계(테이블)인 경우 `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `COPY`를 로깅합니다.

14. Save changes(변경 사항 저장)를 선택합니다.
15. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
16. 데이터베이스 목록에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스 를 선택합니다.

AWS CLI

pgAudit 설정 방법

AWS CLI를 사용하여 pgAudit를 설정하려면 다음 절차와 같이 [modify-db-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-parameter-group.html) 작업을 호출하여 사용자 지정 파라미터 그룹의 감사 로그 파라미터를 수정합니다.

1. 다음 AWS CLI 명령을 사용하여 `shared_preload_libraries` 파라미터에 `pgaudit`를 추가합니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 pgaudit 라이브러리가 초기화되도록 합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 인스턴스를 사용할 수 있게 되면 `pgaudit`가 초기화되었는지 확인할 수 있습니다. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

pgAudit가 초기화되었으므로 이제 확장을 생성할 수 있습니다.

```
CREATE EXTENSION pgaudit;
```

4. AWS CLI를 사용할 수 있도록 `psql` 세션을 닫습니다.

```
labdb=> \q
```

5. 다음 AWS CLI 명령을 사용하여 세션 감사 로깅을 통해 로깅할 문의 클래스를 지정합니다. 이 예에서는 `pgaudit.log` 파라미터를 `write`로 설정하여 로그에 대한 삽입, 업데이트 및 삭제를 캡처합니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

`pgaudit.log` 파라미터에 다음 값 중 하나를 선택할 수도 있습니다.

- 없음 - 기본값입니다. 데이터베이스 변경 사항이 로깅되지 않습니다.
- 모두 - 모든 항목(읽기, 쓰기, 함수, 역할, ddl, 기타)을 로깅합니다.
- ddl - `ROLE` 클래스에 포함되지 않은 모든 데이터 정의 언어(DDL) 문을 로깅합니다.
- 함수 - 함수 호출 및 `DO` 블록을 로깅합니다.
- 기타 – 기타 명령(예: `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, `SET`)을 로깅합니다.
- 읽기 – 원본이 관계(예: 테이블) 또는 쿼리인 경우 `SELECT` 및 `COPY`를 로깅합니다.
- 역할 - 역할 및 권한과 관련된 문을 로깅합니다(예: `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, `DROP ROLE`).
- 쓰기 - 대상이 관계(테이블)인 경우 `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `COPY`를 로깅합니다.

다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

데이터베이스 객체 감사

Aurora PostgreSQL DB 클러스터에 pgAudit를 설정하고 요구 사항에 맞게 구성하면 PostgreSQL 로그에 더 자세한 정보가 캡처됩니다. 예를 들어 기본 PostgreSQL 로깅 구성은 데이터베이스 테이블에서 변경 사항이 적용된 날짜 및 시간을 식별하지만 pgAudit 확장을 사용하면 확장 파라미터의 구성에 따라 스키마, 변경한 사용자 및 기타 세부 정보가 로그 항목에 포함될 수 있습니다. 감사를 설정하여 다음 방법으로 변경 사항을 추적할 수 있습니다.

- 세션마다 사용자별로 추적. 세션 수준에서 정규화된 명령 텍스트를 캡처할 수 있습니다.
- 객체마다 사용자별, 데이터베이스별로 추적

객체 감사 기능은 시스템에서 `rds_pgaudit` 역할을 만든 다음, 사용자 지정 파라미터 그룹의 `pgaudit.role` 파라미터에 이 역할을 추가하면 활성화됩니다. 기본적으로 `pgaudit.role` 파라미터는 설정되어 있지 않으며 유일하게 허용되는 값은 `rds_pgaudit`입니다. 다음 단계에서는 `pgaudit`가 초기화되었고 pgAudit 확장 설정의 절차에 따라 `pgaudit` 확장을 만든 것으로 가정합니다.

```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

이 예에서 볼 수 있듯이 LOG: AUDIT: SESSION 행은 테이블 및 해당 스키마를 비롯한 세부 정보를 제공합니다.

객체 감사를 설정하는 방법

1. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

   ```
   psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
   username=postgrespostgres --password --dbname=labdb
   ```

2. 다음 명령을 사용하여 `rds_pgaudit`라는 데이터베이스 역할을 생성합니다.

   ```
   labdb=> CREATE ROLE rds_pgaudit;
   CREATE ROLE
   labdb=>
   ```

3. `psql` 세션을 닫습니다.

   ```
   labdb=> \q
   ```

   이어질 몇 단계에서는 AWS CLI를 사용하여 사용자 지정 파라미터 그룹에서 감사 로그 파라미터를 수정합니다.

4. 다음 AWS CLI 명령을 사용하여 `rds_pgaudit`에 `pgaudit.role` 파라미터를 추가합니다. 기본적으로 이 파라미터는 설정되어 있지 않으며 유일하게 허용되는 값은 `rds_pgaudit`입니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters 
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot" 
 \
   --region aws-region
```

5. 다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 파라미터 변경 사항이 적용되도록 합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. 다음 명령을 실행하여 `pgaudit.role`이 `rds_pgaudit`로 설정되었는지 확인합니다.

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

pgAudit 로깅을 테스트하기 위해 감사하려는 몇 가지 예제 명령을 실행할 수 있습니다. 예를 들어 다음과 같은 명령을 실행할 수 있습니다.

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

데이터베이스 로그에는 다음과 유사한 항목이 포함됩니다.

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT: 
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

로그 확인에 대한 자세한 내용은 Amazon Aurora 로그 파일 모니터링 단원을 참조하십시오.

pgAudit 확장 프로그램에 대한 자세한 내용은 GitHub에서 pgAudit을 참조하세요.

감사 로깅에서 사용자 또는 데이터베이스 제외

Aurora PostgreSQL 데이터베이스 로그 파일에서 설명한 대로 PostgreSQL 로그는 스토리지 공간을 사용합니다. pgAudit 확장을 사용하면 추적하는 변경 사항에 따라 로그에 수집된 데이터 양이 다양한 수준으로 늘어납니다. Aurora PostgreSQL DB 클러스터의 모든 사용자 또는 데이터베이스를 감사할 필요는 없을 수도 있습니다.

스토리지에 미치는 영향을 최소화하고 불필요하게 감사 레코드를 캡처하지 않도록 사용자 및 데이터베이스를 감사에서 제외할 수 있습니다. 지정된 세션 내에서 로깅을 변경할 수도 있습니다. 다음 예에서는 그 방법을 보여줍니다.

> **Note**
>
> 세션 수준의 파라미터 설정은 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 사용자 지정 DB 파라미터 그룹의 설정보다 우선합니다. 데이터베이스 사용자가 감사 로깅 구성 설정을 우회하지 못하게 하려면 사용자의 권한을 변경해야 합니다.

Aurora PostgreSQL DB 클러스터 가 모든 사용자 및 데이터베이스에 대해 동일한 수준의 활동을 감사하도록 구성되어 있다고 가정하겠습니다. 그런데 사용자 `myuser`를 감사하지 않기로 결정한다면 다음 SQL 명령으로 `myuser`에 대한 감사를 해제할 수 있습니다.

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

그런 다음 다음 쿼리를 사용하여 `pgaudit.log`의 `user_specific_settings` 열에서 파라미터가 `NONE`으로 설정되었는지 확인할 수 있습니다.

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

출력은 다음과 같습니다.

```
 user_name | user_specific_settings
```

```
-----------+------------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

다음 명령을 사용하여 데이터베이스 세션 중에 특정 사용자에 대한 로깅을 해제할 수 있습니다.

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

다음 쿼리를 사용하여 pgaudit.log의 설정 열에서 특정 사용자 및 데이터베이스 조합을 확인할 수 있습니다.

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

출력은 다음과 비슷합니다.

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

`myuser`에 대한 감사를 해제한 후 `mydatabase`에 대한 변경 내용을 추적하지 않기로 결정한다면 다음 명령을 사용하여 특정 데이터베이스에 대한 감사를 해제합니다.

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

그런 다음, 다음 쿼리를 사용하여 database_specific_settings 열에서 pgaudit.log가 NONE으로 설정되어 있는지 확인합니다.

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

출력은 다음과 같습니다.

```
 database_name | database_specific_settings
---------------+---------------------------
 mydatabase    | {pgaudit.log=NONE}
(1 row)
```

myuser의 설정을 기본 설정으로 되돌리려면 다음 명령을 사용합니다.

```
ALTER USER myuser RESET pgaudit.log;
```

데이터베이스의 설정을 기본 설정으로 되돌리려면 다음 명령을 사용합니다.

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

사용자와 데이터베이스를 기본 설정으로 초기화하려면 다음 명령을 사용합니다.

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

`pgaudit.log`를 `pgaudit.log` 파라미터에 허용되는 다른 값 중 하나로 설정하여 특정 이벤트를 로그에 캡처할 수도 있습니다. 자세한 내용은 [`pgaudit.log` 파라미터에 허용되는 설정 목록](#) 단원을 참조하십시오.

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

pgAudit 확장 프로그램에 대한 참조

이 섹션에 나열된 파라미터를 하나 이상 변경하여 감사 로그의 세부 정보 수준을 지정할 수 있습니다.

pgAudit 동작 제어

다음 표에 나열된 파라미터 중 하나 이상을 변경하여 감사 로깅을 제어할 수 있습니다.

파라미터	설명
`pgaudit.log`	세션 감사 로깅으로 로깅될 문의 클래스를 지정합니다. 허용되는 값에는 ddl, 함수, 기타, 읽기, 역할, 쓰기, 없음, 모두 등이 포함됩니다. 자세한 내용은 `pgaudit.log` 파라미터에 허용되는 설정 목록 단원을 참조하십시오.
`pgaudit.log_catalog`	활성화(1로 설정)하면 문의 모든 관계가 pg_catalog에 있는 경우 감사 추적에 문을 추가합니다.
`pgaudit.log_level`	로그 항목에 사용할 로그 수준을 지정합니다. 허용되는 값은 ebug5, debug4, debug3, debug2, debug1, 정보, 알림, 경고, 로그입니다.
`pgaudit.log_parameter`	활성화(1로 설정)하면 문과 함께 전달된 파라미터가 감사 로그에 캡처됩니다.
`pgaudit.log_relation`	활성화(1로 설정)하면 세션의 감사 로그에서 SELECT 또는 DML 문에서 참조되는 각 관계(TABLE, VIEW 등) 에 대해 별도의 로그 항목을 생성합니다.
`pgaudit.log_statement_once`	로깅에 문/하위 문 조합에 대한 첫 번째 로그 항목이 있는 문 텍스트 및 파라미터를 포함할지 아니면 모든 항목이 있는 문 텍스트 및 파라미터를 포함할지를 지정합니다.
`pgaudit.role`	객체 감사 로깅에 사용할 마스터 역할을 지정합니다. 유일하게 허용되는 항목은 `rds_pgaudit` 입니다.

`pgaudit.log` 파라미터에 허용되는 설정 목록

값	설명
없음	이 값이 기본값입니다. 데이터베이스 변경 사항이 로깅되지 않습니다.

값	설명
모두	모든 항목(읽기, 쓰기, 함수, 역할, ddl, 기타)을 로깅합니다.
ddl	ROLE 클래스에 포함되지 않은 모든 데이터 정의 언어(DDL) 문을 로깅합니다.
함수	함수 호출 및 DO 블록을 로깅합니다.
기타	기타 명령을 로깅합니다(예: DISCARD, FETCH, CHECKPOINT , VACUUM, SET).
읽기	원본이 관계(예: 테이블) 또는 쿼리인 경우 SELECT 및 COPY를 로깅합니다.
역할	역할 및 권한과 관련된 문을 로깅합니다(예: GRANT, REVOKE, CREATE ROLE, ALTER ROLE, DROP ROLE).
write	대상이 관계(테이블)인 경우 INSERT, UPDATE, DELETE, TRUNCATE, COPY를 로깅합니다.

세션 감사를 사용하여 여러 이벤트 유형을 기록하려면 쉼표로 구분된 목록을 사용합니다. 모든 이벤트 유형을 기록하려면 `pgaudit.log`를 `ALL`로 설정합니다. DB 인스턴스를 재부팅하여 변경 사항을 적용합니다.

객체 감사를 사용하면 특정 관계를 사용하도록 감사 로깅을 구체화할 수 있습니다. 예를 들어 하나 이상의 테이블에서 READ 작업에 대한 감사 로깅을 지정할 수 있습니다.

pglogical을 사용하여 인스턴스 간 데이터 동기화

현재 사용 가능한 모든 Aurora PostgreSQL 버전은 `pglogical` 확장을 지원합니다. pglogical 확장은 PostgreSQL 버전 10에서 도입된 기능적으로 유사한 논리적 복제 기능보다 먼저 출시되었습니다. 자세한 내용은 Aurora에서의 PostgreSQL 논리적 복제 개요 섹션을 참조하세요.

`pglogical` 확장은 둘 이상의 Aurora PostgreSQL DB 클러스터 간의 논리적 복제를 지원합니다. . 서로 다른 PostgreSQL 버전 간의 복제와 PostgreSQL DB 인스턴스용 RDS 및 Aurora PostgreSQL DB 클러스터에서 실행되는 데이터베이스 간의 복제도 지원합니다. `pglogical` 확장은 게시-구독 모델을 사용하여 게시자의 테이블 및 기타 객체(예: 시퀀스)의 변경 사항을 구독자에 복제합니다. 이 확장은 복제 슬롯을 사용하여, 다음과 같이 게시자 노드의 변경 사항이 구독자 노드로 동기화되게 합니다.

- 게시자 노드는 다른 노드에 복제할 데이터의 소스인 Aurora PostgreSQL DB 클러스터입니다. 게시자 노드는 게시 세트에서 복제될 테이블을 정의합니다.
- 구독자 노드는 게시자로부터 WAL 업데이트를 받는 Aurora PostgreSQL DB 클러스터입니다. 구독자는 구독을 생성하여 게시자에 연결하고 디코딩된 WAL 데이터를 얻습니다. 구독자가 구독을 생성하면 게시자 노드에서 복제 슬롯이 생성됩니다.

아래에서 `pglogical` 확장 설정 관련 정보를 확인할 수 있습니다.

주제

plogical 확장에 대한 요구 사항 및 제한

현재 사용 가능한 모든 Aurora PostgreSQL 릴리스는 `pglogical` 확장을 지원합니다.

게시자 노드와 구독자 노드 모두가 논리적 복제를 할 수 있도록 설정되어야 합니다.

게시자에서 구독자로 복제할 테이블은 이름과 스키마가 동일해야 합니다. 또한 이러한 테이블은 동일한 열을 포함해야 하며, 각 열은 동일한 데이터 유형을 사용해야 합니다. 게시자와 구독자 테이블 모두 프라이머리 키가 동일해야 합니다. PRIMARY KEY만 고유 제약 조건으로 사용하는 것이 좋습니다.

구독자 노드의 테이블에는 CHECK 제약 조건 및 NOT NULL 제약 조건에 대해 게시자 노드의 테이블에 있는 것보다 더 많은 허용 제약 조건이 존재할 수 있습니다.

`pglogical` 확장은 PostgreSQL(버전 10 이상)에 내장된 논리적 복제 기능에서는 지원하지 않는 양방향 복제 같은 기능을 제공합니다. 자세한 내용은 pglogical을 사용한 PostgreSQL 양방향 복제를 참조하십시오.

pglogical 확장 설정

Aurora PostgreSQL DB 클러스터에 `pglogical` 확장을 설정하려면 Aurora PostgreSQL DB 클러스터용 사용자 지정 DB 클러스터 파라미터 그룹의 공유 라이브러리에 `pglogical`을 추가해야 합니다. 논리적 디코딩을 켜려면 `rds.logical_replication` 파라미터의 값을 `1`로 설정해야 합니다. 마지

막으로, 데이터베이스에서 확장을 만듭니다. 이러한 작업에는 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

이러한 작업을 수행하려면 `rds_superuser` 역할의 권한이 있어야 합니다.

다음 단계에서는 사용자의 Aurora PostgreSQL DB 클러스터가 사용자 지정 DB 클러스터에 연결되어 있다고 가정합니다. 사용자 지정 DB 클러스터 파라미터 그룹을 만드는 방법에 관한 자세한 내용은 Amazon Aurora의 파라미터 그룹 섹션을 참조하세요.

콘솔

pglogical 확장 설정 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택합니다.
3. Aurora PostgreSQL DB 클러스터 라이터 인스턴스의 구성 탭을 엽니다. 인스턴스 세부 정보 중에서 파라미터 그룹 링크를 찾습니다.
4. 링크를 선택하여 Aurora PostgreSQL DB 클러스터와 연결된 사용자 지정 파라미터를 엽니다.
5. 파라미터 검색 필드에 `shared_pre`를 입력하여 `shared_preload_libraries` 파라미터를 찾습니다.
6. 파라미터 편집을 선택하여 속성 값에 액세스합니다.
7. 값 필드의 목록에 `pglogical`를 추가합니다. 쉼표를 사용하여 값 목록에서 항목을 구분합니다.

8. `rds.logical_replication` 파라미터를 찾아 1로 설정하여 논리적 복제를 켭니다.
9. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 파라미터 변경 사항이 적용되게 합니다.
10. 인스턴스를 사용할 수 있다면 `psql`(또는 pgAdmin)을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결할 수 있습니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. plogical이 초기화되었는지 확인하려면 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. 논리적 디코딩을 활성화하는 설정을 다음과 같이 확인합니다.

```
SHOW wal_level;
wal_level
-----------
 logical
(1 row)
```

13. 다음과 같이 확장을 생성합니다.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. Save changes(변경 사항 저장)를 선택합니다.
15. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
16. 데이터베이스 목록에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택한 다음, 작업 메뉴에서 재부팅을 선택합니다.

AWS CLI

plogical 확장 설정 방법

AWS CLI를 사용하여 pglogical을 설정하려면 다음 절차와 같이 [modify-db-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-parameter-group.html) 작업을 호출하여 사용자 지정 파라미터 그룹의 특정 파라미터를 수정합니다.

1. 다음 AWS CLI 명령을 사용하여 `shared_preload_libraries` 파라미터에 `pglogical`를 추가합니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 다음 AWS CLI 명령을 사용하여 `rds.logical_replication`을 `1`로 설정하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스용 논리적 디코딩 기능을 켭니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. 다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 pglogical 라이브러리가 초기화되도록 합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. 인스턴스를 사용할 수 있다면 `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. 다음과 같이 확장을 생성합니다.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. 다음 AWS CLI 명령으로 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

Aurora PostgreSQL DB 클러스터용 논리적 복제 설정

다음 절차는 Aurora PostgreSQL DB 클러스터 간에 논리적 복제를 시작하는 방법을 보여줍니다. 다음 단계에서는 원본(게시자)과 대상(구독자) 모두에 pglogical 확장 설정에서 설명하는 방법에 따라 `pglogical` 확장이 설정되어 있다고 가정합니다.

> **Note**
> 구독자 노드의 node_name은 `rds`로 시작할 수 없습니다.

게시자 노드를 생성하고 복제할 테이블을 정의하는 방법

이 단계에서는 Aurora PostgreSQL DB 클러스터에 라이터 인스턴스와 데이터베이스가 하나씩 있고 이러한 데이터베이스에는 다른 노드에 복제할 하나 이상의 테이블이 있다고 가정합니다. 구독자의 테이블 구조를 게시자에서 다시 만들어야 하므로, 필요한 경우 먼저 테이블 구조를 가져와야 합니다. 이렇게 하려면 `psql` 메타 명령을 사용한 다음 구독자 인스턴스에서 동일한 테이블을 생성해야 합니다. 다음 절차에서는 시연을 위해 게시자(원본)에서 예제 테이블을 만듭니다.

1. `psql`을 사용하여 구독자용 소스로 사용할 테이블이 있는 인스턴스에 연결합니다.

```
psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

복제하려는 기존 테이블이 없는 경우 다음과 같이 샘플 테이블을 생성할 수 있습니다.

a. 다음 SQL 문을 사용하여 예제 테이블을 생성합니다.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

b. 다음 SQL 문을 사용하여 테이블에 생성된 데이터를 입력합니다.

```
INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
INSERT 0 5000
```

c. 다음 SQL 문을 사용하여 테이블에 데이터가 있는지 확인합니다.

```
SELECT count(*) FROM docs_lab_table;
```

2. 다음과 같이 이 Aurora PostgreSQL DB 클러스터를 게시자 노드로 식별합니다.

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_provider',
    dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 dbname=labdb');
 create_node
-------------
   3410995529
(1 row)
```

3. 복제할 테이블을 기본 복제 세트에 추가합니다. 복제 세트에 대한 자세한 내용은 pglogical 설명서의 복제 세트를 참조하십시오.

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  ---------------------------
  t
  (1 row)
```

게시자 노드 설정이 완료되었습니다. 이제 게시자로부터 업데이트를 수신하도록 구독자 노드를 설정할 수 있습니다.

구독자 노드를 설정하고, 업데이트를 수신할 구독을 만드는 방법

이 단계에서는 Aurora PostgreSQL DB 클러스터 가 `pglogical` 확장을 이용해 설정되었다고 가정합니다. 자세한 내용은 pglogical 확장 설정 섹션을 참조하세요.

1. `psql`을 사용하여 게시자로부터 업데이트를 수신할 인스턴스에 연결합니다.

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. 구독자 Aurora PostgreSQL DB 클러스터에서 게시자에 존재하는 것과 동일한 테이블을 만듭니다. 이 예제에서 테이블은 `docs_lab_table`입니다. 다음과 같이 테이블을 만들 수 있습니다.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. 이 테이블이 비어 있는지 확인합니다.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. 다음과 같이 이 Aurora PostgreSQL DB 클러스터를 구독자 노드로 식별합니다.

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. 구독을 생성합니다.

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

이 단계를 완료하면 게시자 테이블의 데이터가 구독자 테이블에서 생성됩니다. 다음 SQL 쿼리를 사용하면 이 문제가 발생했는지 확인할 수 있습니다.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

이 시점 이후로는 게시자의 테이블에 적용한 변경 사항이 구독자의 테이블에 복제됩니다.

메이저 업그레이드 후 논리적 복제 재설정

논리적 복제용 게시자 노드로 설정된 Aurora PostgreSQL DB 클러스터 의 메이저 버전 업그레이드를 수행하려면, 먼저 활성화되지 않는 슬롯을 포함한 모든 복제 슬롯을 삭제해야 합니다. 게시자 노드에서 데이터베이스 트랜잭션을 일시적으로 전환하고, 복제 슬롯을 삭제하고, Aurora PostgreSQL DB 클러스터를 업그레이드한 다음 복제를 다시 설정하고 재시작하는 것이 좋습니다.

복제 슬롯은 게시자 노드에서만 호스팅됩니다. 논리적 복제 시나리오에서 Aurora PostgreSQL 구독자 노드는 삭제할 슬롯이 없습니다. Aurora PostgreSQL 메이저 버전 업그레이드 프로세스에서는 게시자 노드와 별개로 구독자를 PostgreSQL의 새 메이저 버전으로 업그레이드할 수 있습니다. 그러나 업그레이드 프로세스를 진행하면 복제 프로세스가 중단되고 게시자 노드와 구독자 노드 간의 WAL 데이터 동기화가 간섭을 받습니다. 게시자나 구독자 또는 둘 항목 모두를 업그레이드한 후에는 게시자와 구독자 간의 논리적 복제를 다시 설정해야 합니다. 다음 절차에서는 복제가 중단되었는지 판단하는 방법과 문제를 해결하는 방법을 확인할 수 있습니다.

논리적 복제가 중단되었는지 확인

다음과 같이 게시자 노드 또는 구독자 노드를 쿼리하면 복제 프로세스 중단 여부를 확인할 수 있습니다.

게시자 노드를 확인하는 방법

- `psql`을 사용하여 게시자 노드에 연결한 다음 `pg_replication_slots` 함수를 쿼리합니다. 활성 열의 값을 기록해 둡니다. 일반적으로 이 값은 `t`(true)를 반환하며, 복제가 활성 상태라는 뜻입니다. 쿼리가 `f`(false)를 반환한다면 구독자로의 복제가 중단되었다는 뜻입니다.

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
```

```
                    slot_name                 |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

구독자 노드를 확인하는 방법

구독자 노드에서는 세 가지 방법으로 복제 상태를 확인할 수 있습니다.

- 구독자 노드의 PostgreSQL 로그를 확인하여 실패 메시지를 찾습니다. 로그는 다음과 같이 종료 코드 1을 포함하는 메시지를 이용해 실패를 식별합니다.

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- `pg_replication_origin` 함수를 쿼리합니다. 다음과 같이 `psql`을 사용하여 구독자 노드의 데이터베이스에 연결하고 `pg_replication_origin` 함수를 쿼리합니다.

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

결과 집합이 비어 있다면 복제가 중단되었다는 뜻입니다. 일반적인 출력은 다음과 같습니다.

```
  roident |                       roname
 ---------+----------------------------------------------------
        1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
 (1 row)
```

- 다음 예제와 같이 `pglogical.show_subscription_status` 함수를 쿼리합니다.

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
---====----------------+--------+-------------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

이 출력은 복제가 중단되었음을 보여줍니다. 상태는 down입니다. 일반적으로 출력에서는 상태가 replicating으로 표시됩니다.

논리적 복제 프로세스가 중단된 경우 다음 단계에 따라 복제를 재설정할 수 있습니다.

게시자와 구독자 노드 간의 논리적 복제를 재설정하는 방법

복제를 다시 설정하려면 먼저 게시자 노드에서 구독자의 연결을 끊은 다음, 다음 단계의 설명에 따라 구독을 다시 설정합니다.

1. 다음과 같이 psql을 사용하여 구독자 노드에 연결합니다.

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. pglogical.alter_subscription_disable 함수를 사용하여 구독을 비활성화합니다.

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. 다음과 같이 pg_replication_origin을 쿼리하여 게시자 노드의 식별자를 가져옵니다.

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+-------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. 이전 단계의 응답을 pg_replication_origin_create 명령과 함께 사용하여, 구독을 다시 설정할 때 사용할 수 있는 식별자를 할당합니다.

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. 다음 예제와 같이 구독의 이름을 true 상태로 전달하여 구독을 활성화합니다.

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

노드의 상태를 확인합니다. 노드의 상태는 이 예제에서처럼 replicating이어야 합니다.

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
             subscription_name |   status    |              slot_name
-------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

게시자 노드에서 구독자 복제 슬롯의 상태를 확인합니다. 슬롯의 active 열은 t(true)를 반환해야 하며, 복제가 다시 설정되었다는 뜻입니다.

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Aurora PostgreSQL용 논리적 복제 슬롯 관리

논리적 복제 시나리오에서 게시자 노드 역할을 하는 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스 의 메이저 버전 업그레이드를 수행하려면, 먼저 인스턴스에서 복제 슬롯을 삭제해야 합니다. 메이저 버전 업그레이드 사전 점검 프로세스에서는 업그레이드를 진행하려면 슬롯을 삭제해야 한다는 메시지가 표시됩니다.

pglogical 확장을 사용하여 만든 복제 슬롯을 식별하려면 각 데이터베이스에 로그인하여 노드 이름을 확인하십시오. 구독자 노드를 쿼리하면 이 예제에서처럼 출력에 게시자와 구독자 노드가 모두 표시됩니다.

```
SELECT * FROM pglogical.node;
node_id   |     node_name
```

```
------------+-------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

다음 쿼리를 사용하면 구독 세부 정보를 확인할 수 있습니다.

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

이제 다음과 같이 구독을 삭제할 수 있습니다.

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

구독을 삭제한 후에는 노드를 삭제해도 됩니다.

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

다음과 같은 방법을 이용해 노드가 더 이상 존재하지 않는지 확인할 수 있습니다.

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

plogical 확장용 파라미터 참조

표에서는 `pglogical` 확장과 관련된 파라미터를 확인할 수 있습니다. `pglogical.conflict_log_level` 및 `pglogical.conflict_resolution` 같은 파라미터는 업

데이트 충돌을 처리하는 용도로 사용합니다. 게시자의 변경 사항을 구독한 테이블과을 로컬로 변경하면 충돌이 발생할 수 있습니다. 양방향 복제 또는 동일한 게시자로부터 여러 구독자를 복제하는 경우를 비롯한 다양한 시나리오에서도 충돌이 발생할 수 있습니다. 자세한 내용은 pglogical을 사용한 PostgreSQL 양방향 복제를 참조하십시오.

파라미터	설명
pglogical.batch_inserts	가능한 경우 배치를 삽입합니다. 기본적으로는 설정되지 않습니다. 켜려면 '1'로 변경하고 해제하려면 '0'으로 변경합니다.
pglogical.conflict_log_level	해결된 충돌을 로깅하는 데 사용할 로그 수준을 설정합니다. 지원되는 문자열 값은 debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal 및 panic입니다.
pglogical.conflict_resolution	해결할 수 있는 충돌인 경우 충돌을 해결하는 데 사용할 메서드를 설정합니다. 지원되는 문자열 값은 error, apply_remote, keep_local, last_update_wins 및 first_update_wins입니다.
pglogical.extra_connection_options	모든 피어 노드 연결에 추가할 연결 옵션입니다.
pglogical.synchronous_commit	pglogical 특정 동기 커밋 값입니다.
pglogical.use_spi	하위 수준 API 대신 SPI(서버 프로그래밍 인터페이스)를 사용하여 변경 사항을 적용합니다. 켜려면 '1'로 설정하고 해제하려면 '0'으로 설정합니다. SPI에 대한 자세한 내용은 PostgreSQL 설명서의 서버 프로그래밍 인터페이스를 참조하십시오.

Amazon Aurora PostgreSQL용 지원되는 외부 데이터 래퍼 작업

외부 데이터 래퍼(FDW)는 외부 데이터에 대한 액세스를 제공하는 특정 유형의 확장입니다. 예를 들어 `oracle_fdw` 확장을 사용하면 Aurora PostgreSQL DB 인스턴스 가 Oracle 데이터베이스와 함께 작동할 수 있습니다.

아래에서는 여러 지원되는 PostgreSQL 외부 데이터 래퍼에 관한 정보를 확인할 수 있습니다.

주제

- log_fdw 확장으로 SQL을 사용하여 DB 로그에 액세스

- postgres_fdw 확장을 사용하여 외부 데이터 액세스
- mysql_fdw 확장을 사용하여 MySQL 데이터베이스 작업
- oracle_fdw 확장을 사용하여 Oracle 데이터베이스 작업
- tds_fdw 확장을 사용하여 SQL Server 데이터베이스 작업

log_fdw 확장으로 SQL을 사용하여 DB 로그에 액세스

Aurora PostgreSQL DB 클러스터는 SQL 인터페이스를 사용하여 데이터베이스 엔진 로그에 액세스할 수 있는 `log_fdw` 확장을 지원합니다. `log_fdw` 확장은 데이터베이스 로그용 외부 테이블을 간편하게 생성할 수 있게 해주는 2가지 함수를 제공합니다.

- `list_postgres_log_files` – 데이터베이스 로그 디렉터리의 파일과 파일 크기(단위: 바이트)를 나열합니다.
- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` – 현재 데이터베이스에서 지정된 파일에 대해 외부 테이블을 빌드합니다.

`log_fdw`가 생성하는 모든 함수는 `rds_superuser`가 소유합니다. `rds_superuser` 역할의 구성원은 다른 데이터베이스 사용자에게 이러한 함수에 대한 액세스 권한을 부여할 수 있습니다.

기본적으로 로그 파일은 `log_destination` 파라미터에 명시된 대로 Amazon Aurora에 의해 `stderr`(표준 오류) 형식으로 생성됩니다. 이 파라미터에는 두 가지 옵션만 있습니다. `stderr` 및 `csvlog`(쉼표로 구분된 값, CSV)입니다. 파라미터에 `csvlog` 옵션을 추가하는 경우 Amazon Aurora가 `stderr` 및 `csvlog` 로그를 모두 생성합니다. 이는 DB 클러스터의 스토리지 용량에 영향을 줄 수 있으므로 로그 처리에 영향을 주는 다른 파라미터를 알고 있어야 합니다. 자세한 내용은 로그 대상 설정 (`stderr`, `csvlog`) 단원을 참조하십시오.

`csvlog` 로그를 생성할 때의 한 가지 이점은 `log_fdw` 확장을 사용하여 여러 열로 깔끔하게 분할된 데이터로 외부 테이블을 작성할 수 있다는 점입니다. 그러려면 인스턴스를 사용자 지정 DB 파라미터 그룹과 연결해야 `log_destination`에 대한 설정을 변경할 수 있습니다. 이에 관한 정보는 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.

다음 예제에서는 `log_destination` 파라미터에 `cvslog`가 포함된 것으로 간주합니다.

log_fdw 확장을 사용하려면

1. `log_fdw` 확장을 설치합니다.

```
postgres=> CREATE EXTENSION log_fdw;
```

```
CREATE EXTENSION
```

2. 로그 서버를 외부 데이터 래퍼로 생성합니다.

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. 로그 파일 목록에서 모든 파일을 선택합니다.

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

샘플 응답은 다음과 같습니다.

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. 선택한 파일에 대해 단일 'log_entry' 열이 있는 테이블을 생성합니다.

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

응답은 테이블이 현재 존재한다는 것 외에는 세부 정보를 제공하지 않습니다.

```
-----------------------------------
(1 row)
```

5. 로그 파일의 샘플을 선택합니다. 다음 코드는 로그 시간 및 오류 메시지 설명을 검색합니다.

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

샘플 응답은 다음과 같습니다.

```
             log_time             |                                  message
----------------------------------
+----------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
```

```
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

postgres_fdw 확장을 사용하여 외부 데이터 액세스

postgres_fdw 확장으로 원격 데이터베이스에 있는 테이블의 데이터에 액세스할 수 있습니다. PostgreSQL DB 인스턴스에서 원격 연결을 설정하는 경우 읽기 전용 복제본에도 액세스할 수 있습니다.

postgres_fdw로 원격 데이터베이스 서버에 액세스하려면

1. postgres_fdw 확장을 설치합니다.

```
CREATE EXTENSION postgres_fdw;
```

2. CREATE SERVER로 외부 데이터 서버를 생성합니다.

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. 원격 서버에 사용할 역할 식별을 위하여 사용자 매핑을 생성합니다.

> **Important**
>
> 로그에 표시되지 않도록 암호를 수정하려면 세션 수준에서 `log_statement=none`을 설정합니다. 파라미터 수준에서 설정하면 암호가 수정되지 않습니다.

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. 원격 서버에서 테이블을 매핑할 테이블을 생성합니다.

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

mysql_fdw 확장을 사용하여 MySQL 데이터베이스 작업

Aurora PostgreSQL DB 클러스터에서 MySQL 호환 데이터베이스에 액세스하려면 `mysql_fdw` 확장을 설치하고 사용하면 됩니다. 이 외부 데이터 래퍼를 사용하면 RDS for MySQL, Aurora MySQL, MariaDB 및 기타 MySQL 호환 데이터베이스로 작업할 수 있습니다. Aurora PostgreSQL DB 클러스터에서 MySQL 데이터베이스로의 연결은 클라이언트 및 서버 구성에 따라 최선의 방식으로 암호화됩니다. 그러나 원하는 경우 암호화를 적용할 수 있습니다. 자세한 내용은 확장에서 전송 중 암호화 사용 단원을 참조하십시오.

`mysql_fdw` 확장은 Amazon Aurora PostgreSQL 버전 15.4, 14.9, 13.12, 12.16 이상에서 지원됩니다. RDS for PostgreSQL DB에서 MySQL 호환 데이터베이스 인스턴스의 테이블에 대한 선택, 삽입, 업데이트 및 삭제를 지원합니다.

주제

- mysql_fdw 확장을 사용하도록 Aurora PostgreSQL DB 설정
- 예: Aurora PostgreSQL에서 Aurora MySQL 데이터베이스로 작업
- 확장에서 전송 중 암호화 사용

mysql_fdw 확장을 사용하도록 Aurora PostgreSQL DB 설정

Aurora PostgreSQL DB 클러스터에서 `mysql_fdw` 확장을 설정하려면 DB 클러스터에서 확장을 로드한 다음 MySQL DB 인스턴스에 대한 연결 지점을 생성해야 합니다. 이 작업을 수행하려면 MySQL DB 인스턴스에 대한 다음과 같은 세부 정보가 필요합니다.

- 호스트 이름 또는 엔드포인트. Aurora MySQL DB 클러스터의 경우 콘솔을 사용하여 엔드포인트를 찾을 수 있습니다. 연결 및 보안(Connectivity & security) 탭을 선택하고 '엔드포인트 및 포트(Endpoint and port)' 섹션을 살펴봅니다.
- 포트 번호. MySQL의 기본 포트 번호는 3306입니다.
- 데이터베이스 이름 DB 식별자입니다.

또한 MySQL 포트 3306의 액세스 제어 목록(ACL) 또는 보안 그룹에 대한 액세스를 제공해야 합니다. Aurora PostgreSQL DB 클러스터와 Aurora MySQL DB 클러스터에 포트 3306에 대한 액세스 권한이 있어야 합니다. 액세스가 올바르게 구성되지 않은 경우 MySQL 호환 테이블에 연결하려고 하면 다음과 비슷한 오류 메시지가 표시됩니다.

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-region.rds.amazonaws.com:3306' (110)
```

다음 절차에서는 `rds_superuser` 계정으로 외부 서버를 생성합니다. 그런 다음 특정 사용자에게 외부 서버에 대한 액세스 권한을 부여합니다. 그러면 이러한 사용자가 MySQL DB 인스턴스로 작업하기 위해 적절한 MySQL 사용자 계정에 대한 자체 매핑을 생성합니다.

mysql_fdw로 MySQL 데이터베이스 서버에 액세스하려면

1. `rds_superuser` 역할이 있는 계정을 사용하여 PostgreSQL DB 인스턴스에 연결합니다. Aurora PostgreSQL DB 클러스터를 생성할 때 기본값을 수락한 경우 사용자 이름은 `postgres`이고 다음과 같이 `psql` 명령줄 도구를 사용하여 연결할 수 있습니다.

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --username=postgres --password
   ```

2. 다음과 같이 `mysql_fdw` 확장을 설치합니다.

   ```
   postgres=> CREATE EXTENSION mysql_fdw;
   CREATE EXTENSION
   ```

Aurora PostgreSQL DB 클러스터에 확장을 설치한 후 MySQL 데이터베이스에 대한 연결을 제공하는 외부 서버를 설정합니다.

외부 서버를 생성하려면

Aurora PostgreSQL DB 클러스터에서 작업을 수행합니다. 이러한 단계에서는 `postgres`와 같은 `rds_superuser` 권한이 있는 사용자로 연결되어 있다고 가정합니다.

1. Aurora PostgreSQL DB 클러스터에서 외부 서버를 생성합니다.

   ```
   postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-name.111122223333.aws-region.rds.amazonaws.com', port '3306');
   CREATE SERVER
   ```

2. 적절한 사용자에게 외부 서버에 대한 액세스 권한을 부여합니다. 관리자가 아닌 사용자, 즉 `rds_superuser` 역할이 없는 사용자여야 합니다.

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

PostgreSQL 사용자는 외부 서버를 통해 MySQL 데이터베이스에 대한 자체 연결을 생성하고 관리합니다.

예: Aurora PostgreSQL에서 Aurora MySQL 데이터베이스로 작업

Aurora PostgreSQL DB 인스턴스에 간단한 테이블이 있다고 가정합니다. Aurora PostgreSQL 사용자는 해당 테이블에 대해 (`SELECT`), `INSERT`, `UPDATE`, `DELETE` 항목을 쿼리하려고 합니다. 앞의 절차에서 설명한 대로 RDS for PostgreSQL DB 인스턴스에 `mysql_fdw` 확장이 생성되었다고 가정합니다. `rds_superuser` 권한이 있는 사용자로 RDS for PostgreSQL DB 인스턴스에 연결한 후 다음 단계를 진행할 수 있습니다.

1. Aurora PostgreSQL DB 인스턴스에서 외부 서버를 생성합니다.

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. `rds_superuser` 권한이 없는 사용자(예: `user1`)에게 사용 권한을 부여합니다.

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. *user1*으로 연결한 다음 MySQL 사용자에 대한 매핑을 생성합니다.

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser', password 'mypassword');
CREATE USER MAPPING
```

4. MySQL 테이블에 연결된 외부 테이블을 만듭니다.

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname 'test', table_name '');
CREATE FOREIGN TABLE
```

5. 외부 테이블에 대한 간단한 쿼리를 실행합니다.

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. MySQL 테이블에서 데이터를 추가, 변경 및 제거할 수 있습니다. 예를 들면 다음과 같습니다.

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

SELECT 쿼리를 다시 실행하여 결과를 확인합니다.

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

확장에서 전송 중 암호화 사용

Aurora PostgreSQL에서 MySQL에 대한 연결은 기본적으로 전송 중 암호화(TLS/SSL)를 사용합니다. 그러나 클라이언트와 서버 구성이 다를 경우 연결이 암호화되지 않은 상태로 폴백됩니다. RDS for MySQL 사용자 계정에서 REQUIRE SSL 옵션을 지정하여 나가는 모든 연결에 대해 암호화를 적용할 수 있습니다. 이와 동일한 접근 방식이 MariaDB 및 Aurora MySQL 사용자 계정에서도 작동합니다.

REQUIRE SSL로 구성된 MySQL 사용자 계정의 경우 보안 연결을 설정할 수 없으면 연결 시도가 실패합니다.

기존 MySQL 데이터베이스 사용자 계정에 암호화를 적용하려면 ALTER USER 명령을 사용할 수 있습니다. 구문은 다음 표에 나온 대로 MySQL 버전에 따라 다릅니다. 자세한 내용은 MySQL 참조 매뉴얼의 [ALTER USER](https://dev.mysql.com/doc/refman/8.0/en/alter-user.html)를 참조하세요.

MySQL 5.7, MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

mysql_fdw 확장에 대한 자세한 내용은 mysql_fdw 설명서를 참조하세요.

oracle_fdw 확장을 사용하여 Oracle 데이터베이스 작업

Aurora PostgreSQL DB 클러스터 에서 Oracle 데이터베이스에 액세스하려면 oracle_fdw 확장을 설치하고 사용할 수 있습니다. 이 확장은 Oracle 데이터베이스의 외부 데이터 래퍼입니다. 이 확장 프로그램에 대해 자세히 알아보려면 oracle_fdw 문서를 참조하세요.

oracle_fdw 확장은 Aurora PostgreSQL 12.7(Amazon Aurora 릴리스 4.2) 이상에서 지원됩니다.

주제

oracle_fdw 확장 켜기

oracle_fdw 확장을 사용하려면 다음 절차를 수행하십시오.

oracle_fdw 확장을 켜려면

- rds_superuser 권한이 있는 계정을 사용하여 다음 명령을 실행합니다.

```
CREATE EXTENSION oracle_fdw;
```

예제: Amazon RDS for Oracle Database에 연결된 외부 서버 사용

다음 예에서는 Amazon RDS for Oracle Database에 연결된 외부 서버를 사용하는 방법을 보여줍니다.

RDS for Oracle Database에 연결된 외부 서버를 만들려면

1. RDS for Oracle DB 인스턴스에서는 다음 사항에 유의하세요.

   - Endpoint
   - 포트
   - 데이터베이스 이름

2. 외부 서버를 만듭니다.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. `rds_superuser` 권한이 없는 사용자(예: `user1`)에게 사용 권한을 부여합니다.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. `user1`로 연결하고 Oracle 사용자에 대한 매핑을 생성합니다.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Oracle 테이블에 연결된 외부 테이블을 만듭니다.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. 외부 테이블을 쿼리합니다.

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

쿼리에서 다음 오류를 보고하면 보안 그룹 및 액세스 제어 목록(ACL)을 확인하여 두 인스턴스가 모두 통신할 수 있는지 확인합니다.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

전송 중 암호화 작업

PostgreSQL에서 Oracle 간의 전송 중인 암호화는 클라이언트와 서버 구성 파라미터의 조합을 기반으로 합니다. Oracle 21c를 사용하는 예제는 Oracle 문서의 About the Values for Negotiating Encryption and Integrity를 참조하세요. Amazon RDS에서 oracle_fdw에 사용되는 클라이언트는 ACCEPTED로 구성됩니다. 즉, Oracle 데이터베이스 서버 구성에 따라 암호화가 달라지며, 암호화에 Oracle Security Library(libnnz)를 사용한다는 의미입니다.

데이터베이스가 RDS for Oracle에 있는 경우 암호화를 구성하려면 Oracle 기본 네트워크 암호화를 참조하세요.

pg_user_mappings 보기 및 권한 이해

PostgreSQL 카탈로그 pg_user_mapping에서 Aurora PostgreSQL 사용자의 매핑을 외부 데이터(원격) 서버의 사용자에게 저장합니다. 카탈로그에 대한 액세스는 제한되어 있지만 pg_user_mappings 보기를 사용하여 매핑을 확인합니다. 다음에서는 예제 Oracle 데이터베이스를 통해 권한이 적용되는 방법을 보여 주는 예제를 확인할 수 있지만, 이 정보는 다른 외부 데이터 래퍼에도 일반적으로 적용됩니다.

다음 결과에서는 세 가지 예제 사용자에 매핑된 역할 및 권한을 찾을 수 있습니다. 사용자 rdssu1과 rdssu2는 rds_superuser 역할이며 user1은 아닙니다. 다음 예제에서는 psql 메타 명령 \du를 사용하여 기존 역할을 나열합니다.

```
test=>  \du
                                                               List of roles
    Role name     |                         Attributes                              |
                   Member of
-----------------+-------------------------------------------------------------
+-------------------------------------------------------------
 rdssu1          |                                                              |
 {rds_superuser}
 rdssu2          |                                                              |
 {rds_superuser}
 user1           |                                                              | {}
```

rds_superuser 권한이 있는 사용자를 포함한 모든 사용자는 pg_user_mappings 테이블에서 자신의 사용자 매핑(umoptions)을 볼 수 있습니다. 다음 예제와 같이 rdssu1이 모든 사용자 매핑을 얻으려고 하면 rdssu1rds_superuser 권한에도 불구하고 오류가 발생합니다.

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

다음은 일부 예입니다.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+----------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+----------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+--------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

information_schema._pg_user_mappings 및 pg_catalog.pg_user_mappings의 구현 차이점 때문에 수동으로 생성된 rds_superuser는 pg_catalog.pg_user_mappings에서 암호를 보려면 추가 권한이 필요합니다.

information_schema._pg_user_mappings에서 암호를 보려면 rds_superuser에 대한 추가 권한은 필요하지 않습니다.

`rds_superuser` 역할이 없는 사용자는 다음 조건에서만 `pg_user_mappings`에서 암호를 볼 수 있습니다.

- 현재 사용자는 매핑되는 사용자이며 서버를 소유하거나 서버에 대한 `USAGE` 권한을 보유하고 있습니다.
- 현재 사용자는 서버 소유자이고 매핑은 `PUBLIC`에 대한 것입니다.

tds_fdw 확장을 사용하여 SQL Server 데이터베이스 작업

PostgreSQL `tds_fdw` 확장을 사용하여 Sybase 및 Microsoft SQL Server 데이터베이스와 같은 테이블 형식 데이터 스트림(TDS) 프로토콜을 지원하는 데이터베이스에 액세스할 수 있습니다. 이 외부 데이터 래퍼를 사용하면 Aurora PostgreSQL DB 클러스터에서 Amazon RDS for Microsoft SQL Server를 포함하여 TDS 프로토콜을 사용하는 데이터베이스에 연결할 수 있습니다. 자세한 내용은 GitHub에서 tds-fdw/tds_fdw 설명서를 참조하세요.

`tds_fdw` 확장은 Amazon Aurora PostgreSQL 버전 13.6 이상에서 지원됩니다.

tds_fdw 확장을 사용하도록 Aurora PostgreSQL DB 설정

다음 절차에서는 Aurora PostgreSQL DB 클러스터와 함께 `tds_fdw`를 설정하고 사용하는 예를 확인할 수 있습니다. `tds_fdw`를 사용하여 SQL Server 데이터베이스에 연결하기 전에 인스턴스에 대한 다음 세부 정보를 확인해야 합니다.

- 호스트 이름 또는 엔드포인트. RDS for SQL Server DB 인스턴스의 경우 콘솔을 사용하여 엔드포인트를 찾을 수 있습니다. 연결 및 보안(Connectivity & security) 탭을 선택하고 '엔드포인트 및 포트(Endpoint and port)' 섹션을 살펴봅니다.
- 포트 번호. Microsoft SQL Server의 기본 포트 번호는 포트 1433입니다.
- 데이터베이스 이름 DB 식별자입니다.

또한 SQL Server 포트 1433의 액세스 제어 목록(ACL) 또는 보안 그룹에 대한 액세스를 제공해야 합니다. Aurora PostgreSQL DB 클러스터와 RDS for MySQL DB 인스턴스가 모두 포트 1433에 액세스할 수 있어야 합니다. 액세스가 올바르게 구성되지 않은 경우 Microsoft SQL Server를 쿼리하려고 할 때 다음 오류 메시지가 나타납니다.

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

tds_fdw를 사용하여 SQL Server 데이터베이스에 연결하려면

1. `rds_superuser` 역할이 있는 계정을 사용하여 Aurora PostgreSQL DB 클러스터의 기본 인스턴스에 연결합니다.

```
psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
 --username=test --password
```

2. `tds_fdw` 확장을 설치합니다.

```
test=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

Aurora PostgreSQL DB 클러스터 에 확장을 설치한 후 외부 서버를 설정합니다.

외부 서버를 생성하려면

`rds_superuser` 권한이 있는 계정을 사용하여 Aurora PostgreSQL DB 클러스터에서 다음과 같은 작업을 수행합니다.

1. Aurora PostgreSQL DB 클러스터에서 외부 서버를 생성합니다.

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
 (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
 'tds_fdw_testing');
CREATE SERVER
```

SQLServer 측에서 ASCII가 아닌 데이터에 액세스하려면 Aurora PostgreSQL DB 클러스터에서 character_set 옵션을 사용하여 서버 링크를 생성합니다.

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. `rds_superuser` 역할 권한이 없는 사용자(예: `user1`)에게 권한을 부여합니다.

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. user1으로 연결한 다음 SQL Server 사용자에 대한 매핑을 생성합니다.

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. SQL Server 테이블에 연결된 외부 테이블을 만듭니다.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. 외부 테이블을 쿼리합니다.

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

연결에 전송 중 암호화 사용

Aurora PostgreSQL에서 SQL Server로의 연결은 SQL Server 데이터베이스 구성에 따라 전송 중 암호화(TLS/SSL)를 사용합니다. SQL Server에 암호화가 구성되지 않은 경우 SQL Server 데이터베이스에 대한 요청을 수행하는 RDS for PostgreSQL 클라이언트가 암호화되지 않은 상태로 폴백됩니다.

`rds.force_ssl` 파라미터를 설정하여 RDS for SQL Server DB 인스턴스에 대한 연결에 암호화를 적용할 수 있습니다. 자세한 방법은 DB 인스턴스 연결이 SSL을 사용하도록 지정을 참조하세요. RDS for SQL Server에 대한 SSL/TLS 구성과 관련된 자세한 내용은 Microsoft SQL Server DB 인스턴스와 함께 SSL 사용을 참조하세요.

PostgreSQL용 신뢰할 수 있는 언어 확장 작업

PostgreSQL용 신뢰할 수 있는 언어 확장은 PostgreSQL 확장을 구축하기 위한 오픈 소스 개발 키트입니다. 이를 통해 고성능 PostgreSQL 확장을 구축하고 Aurora PostgreSQL DB 클러스터에서 안전하게 실행할 수 있습니다. PostgreSQL용 신뢰할 수 있는 언어 확장(TLE)을 사용하면 PostgreSQL 기능 확장을 위해 문서화된 접근 방식을 따르는 PostgreSQL 확장을 생성할 수 있습니다. 자세한 내용은 PostgreSQL 설명서에서 Packaging Related Objects into an Extension을 참조하세요.

TLE의 주요 이점 중 하나는 PostgreSQL 인스턴스의 기반이 되는 파일 시스템에 대한 액세스를 제공하지 않는 환경에서 사용할 수 있다는 것입니다. 이전에는 새 확장을 설치하려면 파일 시스템에 액세스해야 했습니다. TLE는 이러한 제약을 제거합니다. TLE는 Aurora PostgreSQL DB 클러스터에서 실행되는 데이터베이스를 포함하여 모든 PostgreSQL 데이터베이스를 위한 새 확장을 생성할 수 있는 개발 환경을 제공합니다.

TLE는 TLE를 사용하여 만든 확장의 안전하지 않은 리소스에 대한 액세스를 방지하도록 설계되었습니다. 런타임 환경은 확장 결함이 미치는 영향을 단일 데이터베이스 연결로 제한합니다. 또한 TLE는 데이터베이스 관리자에게 확장을 설치할 수 있는 사용자를 세밀하게 제어할 수 있도록 하며, 확장을 실행하기 위한 권한 모델을 제공합니다.

TLE는 Aurora PostgreSQL 버전 14.5 이상 버전에서 지원됩니다.

신뢰할 수 있는 언어 확장 개발 환경 및 런타임은 `pg_tle` PostgreSQL 확장 버전 1.0.1로 패키징됩니다. JavaScript, Perl, Tcl, PL/pgSQL 및 SQL에서 확장 생성을 지원합니다. 다른 PostgreSQL 확장을 설치하는 것과 동일한 방식으로 Aurora PostgreSQL DB 클러스터에 `pg_tle` 확장을 설치합니다. `pg_tle`가 설정되면 개발자는 이를 사용하여 TLE 확장이라는 새 PostgreSQL 확장을 생성할 수 있습니다.

다음 주제에는 신뢰할 수 있는 언어 확장 설정 방법 및 자체 TLE 확장 생성 시작 방법에 대한 정보가 나와 있습니다.

주제

- PostgreSQL용 신뢰할 수 있는 언어 확장 제거
- TLE 확장과 함께 PostgreSQL 후크 사용
- PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 함수 참조
- PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 후크 참조

용어

신뢰할 수 있는 언어 확장을 더 잘 이해하려면 이 항목에서 사용되는 용어에 대한 다음 용어집을 참조하세요.

PostgreSQL용 신뢰할 수 있는 언어 확장

PostgreSQL용 신뢰할 수 있는 언어 확장은 `pg_tle` 확장으로 패키징된 오픈 소스 개발 키트의 공식 이름입니다. 모든 PostgreSQL 시스템에서 사용할 수 있습니다. 자세한 내용은 GitHub의 aws/pg_tle를 참조하세요.

신뢰할 수 있는 언어 확장

신뢰할 수 있는 언어 확장은 PostgreSQL용 신뢰할 수 있는 언어 확장의 약칭입니다. 이 설명서에서는 이 약칭과 약어(TLE)도 사용됩니다.

신뢰할 수 있는 언어

신뢰할 수 있는 언어는 특정 보안 속성을 가진 프로그래밍 또는 스크립팅 언어입니다. 예를 들어 신뢰할 수 있는 언어는 일반적으로 파일 시스템에 대한 액세스를 제한하며, 지정된 네트워킹 속성의 사용을 제한합니다. TLE 개발 키트는 신뢰할 수 있는 언어를 지원하도록 설계되었습니다. PostgreSQL은 신뢰할 수 있거나 신뢰할 수 없는 확장 생성에 사용되는 여러 언어를 지원합니다. 예제는 PostgreSQL 설명서의 Trusted and Untrusted PL/Perl을 참조하세요. 신뢰할 수 있는 언어 확장을 사용하여 생성한 확장은 기본적으로 신뢰할 수 있는 언어 메커니즘을 사용합니다.

TLE 확장

TLE 확장은 신뢰할 수 있는 언어 확장(TLE) 개발 키트를 사용하여 생성된 PostgreSQL 확장입니다.

PostgreSQL용 신뢰할 수 있는 언어 확장을 사용하기 위한 요구 사항

TLE 개발 키트를 설정하고 사용하기 위한 요구 사항은 다음과 같습니다.

- Aurora PostgreSQL 버전 – 신뢰할 수 있는 언어 확장은 Aurora PostgreSQL 버전 14.5 이상 버전에서만 지원됩니다.

  - Aurora PostgreSQL DB 클러스터,를 업그레이드해야 하는 경우, Amazon Aurora PostgreSQL DB 클러스터 업그레이드 를 참조하세요.
  - PostgreSQL을 실행하는 Aurora DB 클러스터 가 아직 없는 경우 새로 생성할 수 있습니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터 생성 및 연결 단원을 참조하세요.
- **`rds_superuser`** 권한 필요 - `pg_tle` 확장을 설정하고 구성하려면 데이터베이스 사용자 역할에 `rds_superuser` 역할의 권한이 있어야 합니다. 기본적으로 이 역할은 Aurora PostgreSQL DB 클러스터를 생성한 `postgres` 사용자에게 부여됩니다.
- 사용자 지정 DB 파라미터 그룹 필요 - Aurora PostgreSQL DB 클러스터는 사용자 지정 DB 파라미터 그룹을 사용하여 구성해야 합니다. 사용자 지정 DB 파라미터 그룹은 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 사용합니다.
  - Aurora PostgreSQL DB 클러스터 가 사용자 지정 DB 파라미터 그룹을 사용하여 구성되지 않은 경우, 파라미터 그룹을 하나 생성하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결해야 합니다. 단계에 대한 간략한 요약은 사용자 지정 DB 파라미터 그룹 생성 및 적용 섹션을 참조하세요.
  - Aurora PostgreSQL DB 클러스터 가 이미 사용자 지정 DB 파라미터 그룹을 사용하여 구성되어 있는 경우 신뢰할 수 있는 언어 확장을 설정할 수 있습니다. 세부 정보는 Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정을 참조하세요.

사용자 지정 DB 파라미터 그룹 생성 및 적용

다음 단계를 사용하여 사용자 지정 DB 파라미터 그룹을 생성하고 이를 사용하도록 Aurora PostgreSQL DB 클러스터를 구성합니다.

콘솔

사용자 지정 DB 파라미터 그룹을 생성하고 Aurora PostgreSQL DB 클러스터에 사용하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 메뉴에서 Parameter groups(파라미터 그룹)를 선택합니다.
3. Create parameter group(파라미터 그룹 생성)을 선택합니다.
4. Parameter group details(파라미터 그룹 세부 정보) 페이지에서 다음 정보를 입력합니다.

    - Parameter group family(파라미터 그룹 패밀리)에서 aurora-postgresql14 를 선택합니다.
    - Type(유형)에서 DB Parameter Group을 선택합니다.
    - Group name(그룹 이름)에서 작업 컨텍스트에서 의미 있는 파라미터 그룹 이름을 지정합니다.

   - Description(설명)에 다른 팀원이 쉽게 찾을 수 있도록 유용한 설명을 입력합니다.

5. Create(생성)를 선택합니다. 사용자 지정 DB 파라미터 그룹이 AWS 리전에 생성됩니다. 이제 다음 단계에 따라 파라미터 그룹을 사용하도록 Aurora PostgreSQL DB 클러스터를 수정할 수 있습니다.
6. Amazon RDS 메뉴에서 Databases(데이터베이스)를 선택합니다.
7. 나열된 항목 중에서 TLE와 함께 사용할 Aurora PostgreSQL DB 클러스터를 선택한 다음 Modify(수정)를 선택합니다.
8. Modify DB Cluster settings(DB 클러스터 설정 수정) 페이지에서 데이터베이스 옵션을 찾은 다음 선택기를 사용하여 사용자 지정 DB 파라미터 그룹을 선택합니다.
9. Continue(계속)를 선택하여 변경 내용을 저장합니다.
10. Apply immediately(즉시 적용)를 선택하면 TLE를 사용하도록 Aurora PostgreSQL DB 클러스터를 계속 설정할 수 있습니다.

신뢰할 수 있는 언어 확장을 사용하도록 시스템을 계속 설정하려면 Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정 섹션을 참조하세요.

DB 클러스터 및 DB 파라미터 그룹 작업에 대한 자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 섹션을 참조하세요.

AWS CLI

기본 AWS 리전으로 AWS CLI를 구성하면 CLI 명령을 사용할 때 `--region` 인수를 지정하지 않아도 됩니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 구성 기초 섹션을 참조하세요.

사용자 지정 DB 파라미터 그룹을 생성하고 Aurora PostgreSQL DB 클러스터에 사용하는 방법

1. create-db-parameter-group AWS CLI 명령을 사용하여 사용자 AWS 리전의 aurora-postgresql14를 기반으로 사용자 지정 DB 파라미터 그룹을 만들 수 있습니다. 이 단계에서는 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 적용할 DB 파라미터 그룹을 생성합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

Windows의 경우:

```
aws rds create-db-parameter-group ^
  --region aws-region ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --db-parameter-group-family aurora-postgresql14 ^
  --description "My custom DB parameter group for Trusted Language Extensions"
```

사용자 지정 DB 파라미터 그룹은 AWS 리전에서 사용할 수 있으므로 파라미터 그룹을 사용하도록 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 수정할 수 있습니다.

2. modify-db-instance AWS CLI 명령을 사용하여 사용자 지정 DB 파라미터 그룹을 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 적용합니다. 이 명령은 활성 인스턴스를 즉시 재부팅합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
  --region aws-region \
  --db-instance-identifier your-writer-instance-name \
  --db-parameter-group-name custom-params-for-pg-tle \
  --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

신뢰할 수 있는 언어 확장을 사용하도록 시스템을 계속 설정하려면 Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정 섹션을 참조하세요.

자세한 내용은 Amazon Aurora DB 인스턴스용 DB 파라미터 그룹 단원을 참조하세요.

Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정

다음 단계에서는 사용자의 Aurora PostgreSQL DB 클러스터 가 사용자 지정 DB 클러스터 파라미터 그룹에 연결되어 있다고 가정합니다. 이러한 단계에 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

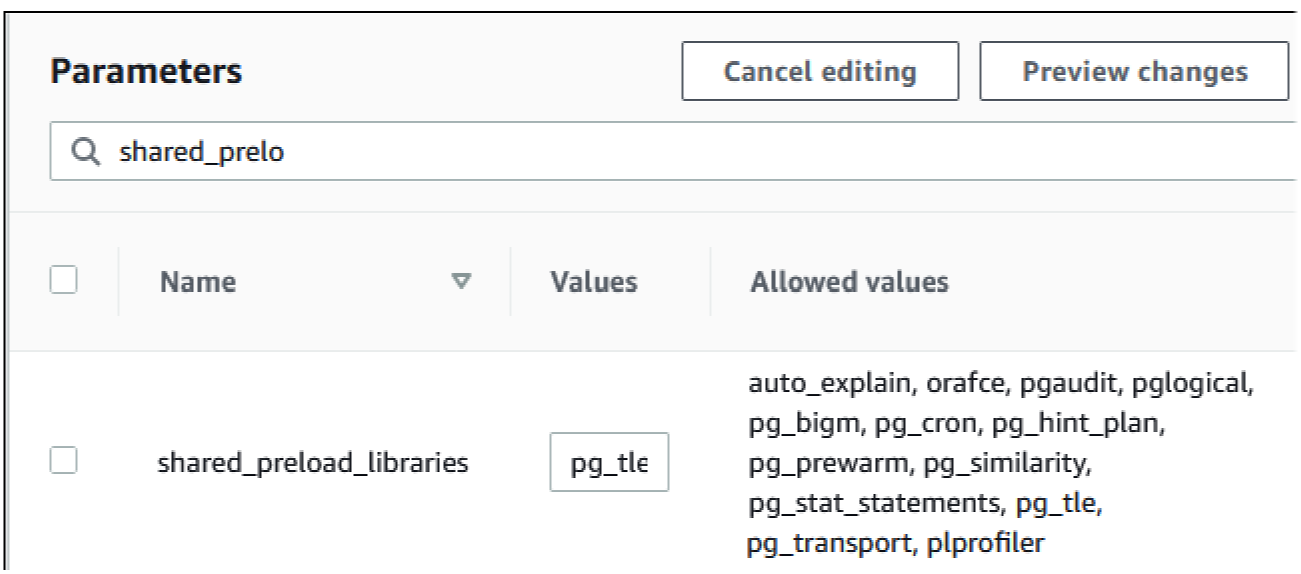
Aurora PostgreSQL DB 클러스터 에 신뢰할 수 있는 언어 확장을 설정하는 경우 특정 데이터베이스 권한이 있는 데이터베이스 사용자가 사용할 수 있도록 특정 데이터베이스에 설치합니다.

콘솔

신뢰할 수 있는 언어 확장 설정 방법

`rds_superuser` 그룹(역할)의 구성원인 계정을 사용하여 다음 단계를 수행합니다.

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 선택합니다.
3. Aurora PostgreSQL DB 클러스터 라이터 인스턴스의 구성 탭을 엽니다. 인스턴스 세부 정보 중에서 파라미터 그룹 링크를 찾습니다.
4. 링크를 선택하여 Aurora PostgreSQL DB 클러스터와 연결된 사용자 지정 파라미터를 엽니다.
5. 파라미터 검색 필드에 `shared_pre`를 입력하여 `shared_preload_libraries` 파라미터를 찾습니다.
6. 파라미터 편집을 선택하여 속성 값에 액세스합니다.
7. 값 필드의 목록에 `pg_tle`를 추가합니다. 쉼표를 사용하여 값 목록에서 항목을 구분합니다.



8. Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하여 shared_preload_libraries 파라미터 변경 사항이 적용되도록 합니다.
9. 인스턴스를 사용할 수 있게 되면 pg_tle가 초기화되었는지 확인합니다. psql을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. pg_tle 확장이 초기화되었으므로 이제 확장을 생성할 수 있습니다.

```
CREATE EXTENSION pg_tle;
```

다음 psql 메타 명령을 사용하여 확장이 설치되었는지 확인할 수 있습니다.

```
labdb=> \dx
                         List of installed extensions
  Name    | Version |   Schema   |                Description
---------+---------+------------+--------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. PostgreSQL DB 인스턴스를 설정할 때 Aurora PostgreSQL DB 클러스터를 위해 만든 기본 사용자 이름에 pgtle_admin 역할을 부여합니다. 기본값을 수락했다면 postgres입니다.

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

다음 예제와 같이 psql 메타 명령을 사용하여 역할이 부여되었는지 확인할 수 있습니다. pgtle_admin 및postgres 역할만 출력에 표시됩니다. 자세한 내용은 PostgreSQL 역할 및 권한 이해. 섹션을 참조하세요.

```
labdb=> \du
                          List of roles
    Role name     |             Attributes             |               Member of
-----------------+----------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                      | {}
```

```
postgres         | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                 | Password valid until infinity    |...
```

12. \q 메타 명령을 사용하여psql 세션을 닫습니다.

```
\q
```

TLE 확장 생성을 시작하려면 예: SQL을 사용하여 신뢰할 수 있는 언어 확장 생성 섹션을 참조하세요.

AWS CLI

기본 AWS 리전으로 AWS CLI를 구성하면 CLI 명령을 사용할 때 `--region` 인수를 지정하지 않아도 됩니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 구성 기초 섹션을 참조하세요.

신뢰할 수 있는 언어 확장 설정 방법

1. modify-db-parameter-group AWS CLI 명령을 사용하여 `pg_tle`를 `shared_preload_libraries` 파라미터에 추가합니다.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. reboot-db-instance AWS CLI 명령을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 재부팅하고 `pg_tle` 라이브러리를 초기화합니다.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 인스턴스를 사용할 수 있게 되면 `pg_tle`가 초기화되었는지 확인할 수 있습니다. `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결하고 다음 명령을 실행합니다.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

pg_tle이 초기화되었으므로 이제 확장을 생성할 수 있습니다.

```
CREATE EXTENSION pg_tle;
```

4. PostgreSQL DB 인스턴스를 설정할 때 Aurora PostgreSQL DB 클러스터를 위해 만든 기본 사용자 이름에 pgtle_admin 역할을 부여합니다. 기본값을 수락했다면 postgres입니다.

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

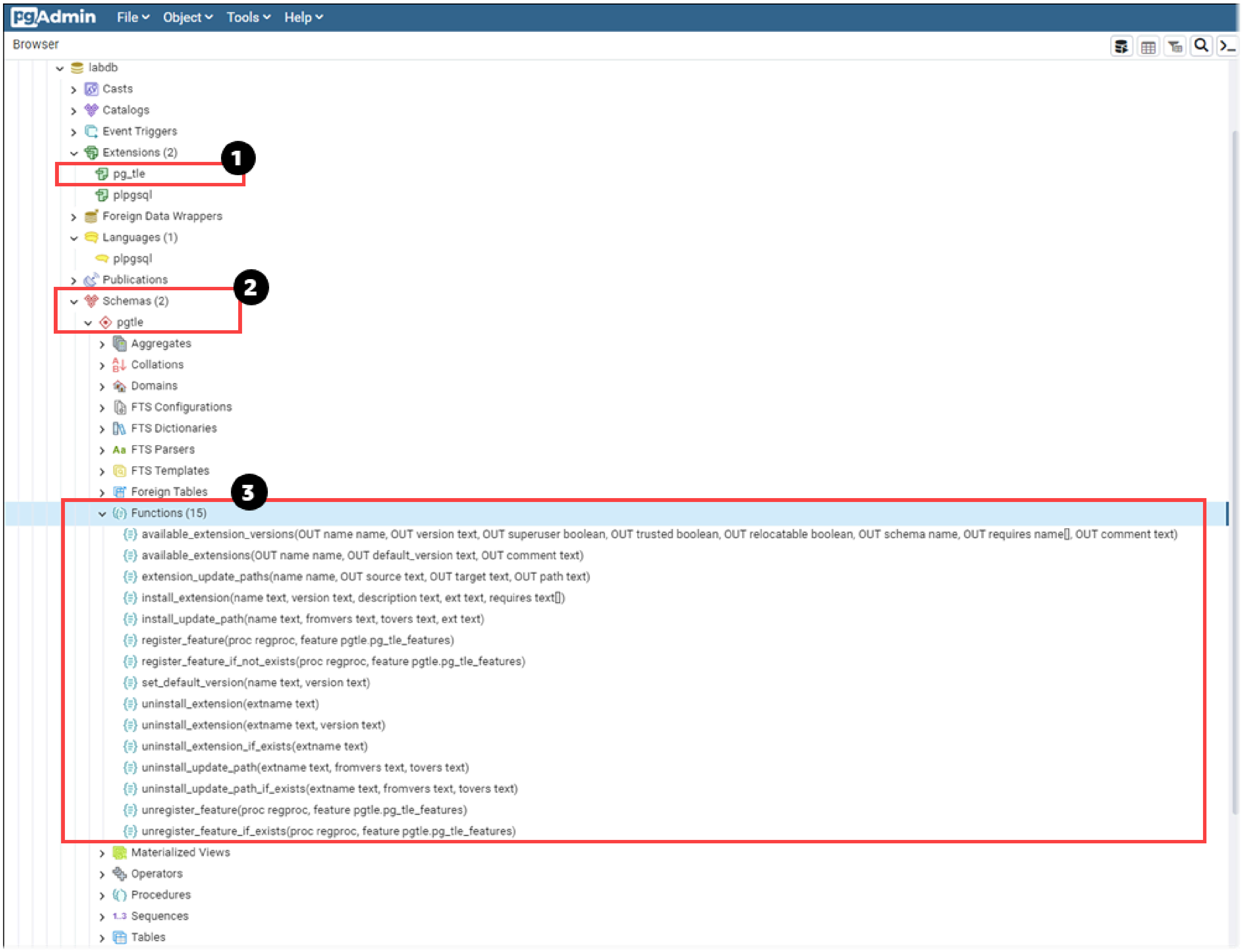
5. 다음과 같이 psql 세션을 닫습니다.

```
labdb=> \q
```

TLE 확장 생성을 시작하려면 예: SQL을 사용하여 신뢰할 수 있는 언어 확장 생성 섹션을 참조하세요.

PostgreSQL용 신뢰할 수 있는 언어 확장 개요

PostgreSQL용 신뢰할 수 있는 언어 확장은 다른 PostgreSQL 확장을 설정하는 것과 동일한 방식으로 Aurora PostgreSQL DB 클러스터에 설치하는 PostgreSQL 확장입니다. PgAdmin 클라이언트 도구에 있는 예제 데이터베이스의 다음 이미지에서 pg_tle 확장을 구성하는 일부 구성 요소를 볼 수 있습니다.



다음 세부 정보를 볼 수 있습니다.

1. PostgreSQL용 신뢰할 수 있는 언어 확장(TLE)은 `pg_tle` 확장으로 패키징된 개발 키트입니다. 따라서 `pg_tle`는 설치되는 데이터베이스의 사용 가능한 확장에 추가됩니다.
2. TLE에는 자체 스키마 `pgtle`이 있습니다. 이 스키마에는 생성한 확장을 설치하고 관리하는 도우미 함수(3)가 포함되어 있습니다.
3. TLE는 확장 설치, 등록, 관리를 위한 12개 이상의 도우미 함수를 제공합니다. 이러한 함수에 대한 자세한 내용은 [PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 함수 참조](#) 섹션을 참조하세요.

`pg_tle` 확장의 기타 구성 요소에는 다음이 포함됩니다.

- **pgtle_admin** 역할 - pg_tle 확장이 설치될 때 pgtle_admin 역할이 생성됩니다. 이 역할은 권한이 부여되므로 그에 따라 취급되어야 합니다. 데이터베이스 사용자에게 pgtle_admin 역할을 부여할 때는 최소 권한 원칙을 따르는 것이 좋습니다. 즉, 새 TLE 확장을 생성, 설치, 관리할 수 있는 데이터베이스 사용자에게만 pgtle_admin 역할을 부여합니다(예: postgres).
- **pgtle.feature_info** 테이블 - pgtle.feature_info 테이블은 TLE, 후크, TLE와 후크가 사용하는 사용자 지정 저장 프로시저 및 함수에 대한 정보가 포함된 보호된 테이블입니다. pgtle_admin 권한이 있는 경우 다음과 같은 신뢰할 수 있는 언어 확장 함수를 사용하여 테이블에서 해당 정보를 추가하고 업데이트할 수 있습니다.
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Aurora PostgreSQL용 TLE 확장 생성

pg_tle 확장이 설치된 Aurora PostgreSQL DB 클러스터 에서 TLE를 사용하여 생성한 확장을 설치할 수 있습니다. pg_tle 확장은 설치된 PostgreSQL 데이터베이스로 범위가 지정됩니다. TLE를 사용하여 생성한 확장은 동일한 데이터베이스로 범위가 지정됩니다.

다양한 pgtle 함수를 사용하여 TLE 확장을 구성하는 코드를 설치하세요. 다음과 같은 신뢰할 수 있는 언어 확장 함수는 모두 pgtle_admin 역할이 필요합니다.

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature

- pgtle.unregister_feature_if_exists

예: SQL을 사용하여 신뢰할 수 있는 언어 확장 생성

다음 예제는 다양한 공식을 사용하여 거리를 계산하는 몇 가지 SQL 함수가 포함된 pg_distance라는 TLE 확장을 생성하는 방법을 보여 줍니다. 목록에서 맨해튼 거리를 계산하는 함수와 유클리드 거리를 계산하는 함수를 찾을 수 있습니다. 이러한 공식의 차이에 대한 자세한 내용은 Wikipedia의 Taxicab geometry와 Euclidean geometry를 참조하세요.

Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정에 설명된 대로 pg_tle 확장을 설정한 경우 자체 Aurora PostgreSQL DB 클러스터 에서 이 예제를 사용할 수 있습니다.

> **Note**
>
> 이 절차를 수행하려면 pgtle_admin 역할의 권한이 있어야 합니다.

예제 TLE 확장을 생성하는 방법

다음 단계에서는 labdb라는 예제 데이터베이스를 사용합니다. 이 데이터베이스는 postgres 기본 사용자가 소유합니다. postgres 역할에는 pgtle_admin 역할의 권한도 있습니다.

1. psql을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 다음 코드를 복사하고 psql 세션 콘솔에 붙여넣어 이름이 pg_distance인 TLE 확장을 생성합니다.

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;
```

```
    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

출력은 다음과 같습니다.

```
install_extension
---------------
 t
(1 row)
```

pg_distance 확장을 구성하는 아티팩트가 이제 데이터베이스에 설치되었습니다. 이러한 아티팩트에는 제어 파일과 확장 코드가 포함되며, 이러한 항목은 CREATE EXTENSION 명령을 사용하여 확장을 생성하려면 있어야 하는 항목입니다. 즉, 데이터베이스 사용자가 확장의 함수를 사용할 수 있도록 하려면 확장을 생성해야 합니다.

3. 확장을 생성하려면 다른 확장과 마찬가지로 CREATE EXTENSION 명령을 사용하세요. 다른 확장과 마찬가지로 데이터베이스 사용자는 데이터베이스에서 CREATE 권한이 있어야 합니다.

```
CREATE EXTENSION pg_distance;
```

4. pg_distance TLE 확장을 테스트하려면 TLE 확장을 사용하여 네 점 사이의 [맨해튼 거리]를 계산하면 됩니다.

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

동일한 점 집합 간의 [유클리드 거리]를 계산하려면 다음을 사용하면 됩니다.

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
```

```
5.656854249492381
```

pg_distance 확장은 데이터베이스에 함수를 로드하여 데이터베이스에서 권한이 있는 모든 사용자가 사용할 수 있도록 합니다.

TLE 확장 수정

이 TLE 확장에 패키징된 함수의 쿼리 성능을 향상시키려면 다음 두 PostgreSQL 속성을 해당 사양에 추가합니다.

- IMMUTABLE - IMMUTABLE 속성은 쿼리 최적화 프로그램이 최적화를 통해 쿼리 응답 시간을 개선할 수 있도록 합니다. 자세한 내용은 PostgreSQL 설명서에서 Function Volatility Categories를 참조하세요.
- PARALLEL SAFE - PARALLEL SAFE 속성은 PostgreSQL이 병렬 모드에서 함수를 실행할 수 있도록 하는 또 다른 속성입니다. 자세한 내용은 PostgreSQL 설명서에서 CREATE FUNCTION을 참조하십시오.

다음 예제에서는 pgtle.install_update_path 함수를 사용하여 각 함수에 이러한 속성을 추가하여 pg_distance TLE 확장의 0.2 버전을 생성하는 방법을 확인할 수 있습니다. 이 함수에 대한 자세한 내용은 pgtle.install_update_path 단원을 참조하세요. 이 작업을 수행하려면 pgtle_admin 역할이 있어야 합니다.

기존 TLE 확장을 업데이트하고 기본 버전을 지정하는 방법

1. psql 또는 pgAdmin 같은 다른 클라이언트 도구를 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스 에 연결합니다.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 다음 코드를 복사하고 psql 세션 콘솔에 붙여넣어 기존 TLE 확장을 수정합니다.

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
```

```
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

다음과 비슷한 응답이 나타납니다.

```
install_update_path
---------------------
 t
(1 row)
```

이 확장 버전을 기본 버전으로 지정하면 데이터베이스 사용자가 데이터베이스에서 확장을 생성 또는 업데이트할 때 버전을 지정하지 않아도 됩니다.

3. TLE 확장의 수정된 버전(버전 0.2)을 기본 버전으로 지정하려면 다음 예제와 같이 `pgtle.set_default_version` 함수를 사용하세요.

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

이 함수에 대한 자세한 내용은 [pgtle.set_default_version](#) 단원을 참조하세요.

4. 코드가 준비되면 다음과 같이 `ALTER EXTENSION ... UPDATE` 명령을 사용하여 일반적인 방법으로 설치된 TLE 확장을 업데이트할 수 있습니다.

```
ALTER EXTENSION pg_distance UPDATE;
```

데이터베이스에서 TLE 확장 삭제

다른 PostgreSQL 확장과 동일한 방식으로 DROP EXTENSION 명령을 사용하여 TLE 확장을 삭제할 수 있습니다. 확장을 삭제해도 확장을 구성하는 설치 파일은 제거되지 않으므로 사용자가 확장을 다시 생성할 수 있습니다. 확장 및 해당 설치 파일을 제거하려면 다음 2단계 프로세스를 수행하세요.

TLE 확장을 삭제하고 해당 설치 파일을 제거하는 방법

1. psql 또는 다른 클라이언트 도구를 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. 다른 PostgreSQL 확장과 같은 방법으로 확장을 삭제합니다.

```
DROP EXTENSION your-TLE-extension
```

예를 들어 예: SQL을 사용하여 신뢰할 수 있는 언어 확장 생성에 설명된 대로 pg_distance 확장을 생성한 경우 다음과 같이 확장을 삭제할 수 있습니다.

```
DROP EXTENSION pg_distance;
```

다음과 같이 확장이 삭제되었음을 확인하는 출력이 표시됩니다.

```
DROP EXTENSION
```

이 시점에서 확장은 더 이상 데이터베이스에서 활성 상태가 아닙니다. 그러나 설치 파일과 제어 파일은 여전히 데이터베이스에서 사용할 수 있으므로 데이터베이스 사용자는 원하는 경우 확장을 다시 생성할 수 있습니다.

- 데이터베이스 사용자가 TLE 확장자를 생성할 수 있도록 확장 파일을 그대로 두려면 여기서 멈추면 됩니다.
- 확장을 구성하는 모든 파일을 제거하려면 다음 단계를 계속합니다.

3. 확장의 모든 설치 파일을 제거하려면 pgtle.uninstall_extension 함수를 사용하세요. 이 함수는 확장의 모든 코드와 제어 파일을 제거합니다.

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

예를 들어 모든 pg_distance 설치 파일을 제거하려면 다음 명령을 사용합니다.

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

PostgreSQL용 신뢰할 수 있는 언어 확장 제거

더 이상 TLE를 사용하여 TLE 확장을 직접 생성하지 않으려면 pg_tle 확장을 삭제하고 모든 아티팩트를 제거하면 됩니다. 이 작업에는 데이터베이스의 모든 TLE 확장 삭제 및 pgtle 스키마 삭제가 포함됩니다.

데이터베이스에서 **pg_tle** 확장 및 해당 스키마를 삭제하는 방법

1. psql 또는 다른 클라이언트 도구를 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. 데이터베이스에서 pg_tle 확장을 삭제합니다. 데이터베이스에서 자체 TLE 확장이 아직 실행 중인 경우 해당 확장도 삭제해야 합니다. 이를 위해 다음과 같이 CASCADE 키워드를 사용할 수 있습니다.

```
DROP EXTENSION pg_tle CASCADE;
```

pg_tle 확장이 데이터베이스에서 아직 활성 상태가 아닌 경우 CASCADE 키워드를 사용할 필요가 없습니다.

3. pgtle 스키마를 삭제합니다. 이 작업을 수행하면 데이터베이스에서 모든 관리 함수가 제거됩니다.

```
DROP SCHEMA pgtle CASCADE;
```

이 명령은 프로세스가 완료되면 다음을 반환합니다.

```
DROP SCHEMA
```

`pg_tle` 확장, 해당 스키마와 함수, 모든 아티팩트가 제거됩니다. TLE를 사용하여 새 확장을 생성하려면 설정 프로세스를 다시 진행하세요. 자세한 내용은 [Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정](#) 단원을 참조하십시오.

TLE 확장과 함께 PostgreSQL 후크 사용

후크는 PostgreSQL에서 사용할 수 있는 콜백 메커니즘으로, 개발자가 일반 데이터베이스 작업 중에 사용자 지정 함수 또는 기타 루틴을 호출할 수 있습니다. TLE 개발 키트는 PostgreSQL 후크를 지원하므로 런타임에 사용자 지정 함수를 PostgreSQL 동작과 통합할 수 있습니다. 예를 들어 후크를 사용하여 인증 프로세스를 사용자 지정 코드와 연결하거나 특정 요구 사항에 맞게 쿼리 계획 및 실행 프로세스를 수정할 수 있습니다.

TLE 확장은 후크를 사용할 수 있습니다. 후크의 범위가 전역적이면 모든 데이터베이스에 적용됩니다. 따라서 TLE 확장이 전역 후크를 사용하는 경우 사용자가 액세스할 수 있는 모든 데이터베이스에 TLE 확장을 생성해야 합니다.

`pg_tle` 확장을 사용하여 신뢰할 수 있는 언어 확장을 직접 구축하는 경우 SQL API에서 사용 가능한 후크를 사용하여 확장의 함수를 구축할 수 있습니다. 모든 후크는 `pg_tle`에 등록해야 합니다. 일부 후크의 경우 다양한 구성 파라미터를 설정해야 할 수도 있습니다. 예를 들어 `passcode` 확인 후크는 켜기, 해제 또는 필수로 설정할 수 있습니다. 사용 가능한 `pg_tle` 후크의 특정 요구 사항에 대한 자세한 내용은 [PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 후크 참조](#) 섹션을 참조하세요.

예: PostgreSQL 후크를 사용하는 확장 생성

이 섹션에서 설명하는 예제에서는 PostgreSQL 후크를 사용하여 특정 SQL 작업 중에 제공된 암호를 확인하고 데이터베이스 사용자가 `password_check.bad_passwords` 테이블에 포함된 암호로 암호를 설정하지 못하도록 합니다. 이 테이블에는 가장 일반적으로 사용되지만 쉽게 깰 수 있는 상위 10개 암호가 나와 있습니다.

이 예제를 Aurora PostgreSQL DB 클러스터 에서 설정하려면 신뢰할 수 있는 언어 확장이 이미 설치되어 있어야 합니다. 자세한 내용은 [Aurora PostgreSQL DB 클러스터에서 신뢰할 수 있는 언어 확장 설정]을 참조하세요.

암호 확인 후크 예제를 설정하는 방법

1. psql을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. [암호 확인 후크 코드 목록]에서 코드를 복사하여 데이터베이스에 붙여넣습니다.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
```

```
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

확장이 데이터베이스에 로드되면 다음과 같은 출력이 표시됩니다.

```
 install_extension
-------------------
 t
(1 row)
```

3. 데이터베이스에 연결되어 있는 동안 확장을 생성할 수 있습니다.

```
CREATE EXTENSION my_password_check_rules;
```

4. 다음 psql 메타 명령을 사용하여 데이터베이스에 확장이 생성되었는지 확인할 수 있습니다.

```
\dx
```

```
                            List of installed extensions
          Name           | Version |    Schema    |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. AWS CLI로 작업할 다른 터미널 세션을 엽니다. 암호 확인 후크를 켜려면 사용자 지정 DB 파라미터 그룹을 수정해야 합니다. 이렇게 하려면 다음 예제에서와 같이 [modify-db-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-parameter-group.html) CLI 명령을 사용합니다.

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

   파라미터 그룹 설정 변경 사항이 적용되기까지 몇 분 정도 걸릴 수 있습니다. 하지만 이 파라미터는 동적이므로 설정을 적용하기 위해 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스 를 다시 시작할 필요는 없습니다.

6. psql 세션을 열고 데이터베이스를 쿼리하여 password_check 후크가 켜졌는지 확인합니다.

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
-----------------------------
on
(1 row)
```

이제 password-check 후크가 활성 상태입니다. 다음 예제와 같이 새 역할을 생성하고 잘못된 암호 중 하나를 사용하여 이를 테스트할 수 있습니다.

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
```

```
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

출력은 가독성을 위해 형식이 지정되었습니다.

다음 예제는 pgsql 대화형 메타 명령 `\password` 동작도 password_check 후크의 영향을 받는다는 것을 보여 줍니다.

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

원하는 경우 이 TLE 확장을 삭제하고 소스 파일을 제거할 수 있습니다. 자세한 내용은 데이터베이스에서 TLE 확장 삭제 섹션을 참조하세요.

암호 확인 후크 코드 목록

여기에 표시된 예제 코드는 `my_password_check_rules` TLE 확장의 사양을 정의합니다. 이 코드를 복사하여 데이터베이스에 붙여넣으면 `my_password_check_rules` 확장 코드가 데이터베이스에 로드되고 확장에서 사용할 수 있도록 `password_check` 후크가 등록됩니다.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
```

```
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;
```

```
  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 함수 참조

PostgreSQL용 신뢰할 수 있는 언어 확장에서 사용할 수 있는 함수에 대한 다음 참조 설명서를 참조하세요. 이러한 함수를 사용하면 TLE 확장, 즉 신뢰할 수 있는 언어 확장 개발 키트를 사용하여 개발한 PostgreSQL 확장을 설치, 등록, 업데이트 및 관리할 수 있습니다.

함수

- pgtle.available_extensions
- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

`pgtle.available_extensions` 함수는 집합을 반환하는 함수입니다. 이 함수는 데이터베이스에서 사용 가능한 모든 TLE 확장을 반환합니다. 반환된 각 행에는 단일 TLE 확장에 대한 정보가 포함되어 있습니다.

함수 프로토타입

```
pgtle.available_extensions()
```

역할

없음.

인수

없음.

출력

- name - TLE 확장의 이름입니다.
- default_version - 버전을 지정하지 않고 CREATE EXTENSION을 호출할 때 사용할 TLE 확장의 버전입니다.
- description - TLE 확장에 대한 자세한 설명입니다.

사용 예

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

available_extension_versions 함수는 집합을 반환하는 함수입니다. 이 함수는 사용 가능한 모든 TLE 확장 및 버전의 목록을 반환합니다. 각 행에는 특정 역할이 필요한지 여부를 포함하여 지정된 TLE 확장의 특정 버전에 대한 정보가 포함되어 있습니다.

함수 프로토타입

```
pgtle.available_extension_versions()
```

역할

없음.

인수

없음.

출력

- name - TLE 확장의 이름입니다.
- version - TLE 확장의 버전입니다.
- superuser - TLE 확장의 경우 이 값은 항상 false입니다. TLE 확장을 생성하거나 업데이트하는 데 필요한 권한은 지정된 데이터베이스에서 다른 객체를 만드는 데 필요한 권한과 동일합니다.
- trusted - TLE 확장의 경우 이 값은 항상 false입니다.
- relocatable - TLE 확장의 경우 이 값은 항상 false입니다.
- schema - TLE 확장이 설치된 스키마의 이름을 지정합니다.
- requires - 이 TLE 확장에 필요한 다른 확장의 이름을 포함하는 배열입니다.
- description - TLE 확장에 대한 자세한 설명입니다.

출력 값에 대한 자세한 내용은 PostgreSQL 설명서에서 Packaging Related Objects into an Extension > Extension Files를 참조하세요.

사용 예

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

extension_update_paths 함수는 집합을 반환하는 함수입니다. 이 함수는 TLE 확장에 가능한 모든 업데이트 경로 목록을 반환합니다. 각 행에는 해당 TLE 확장에 사용할 수 있는 업그레이드 또는 다운그레이드가 포함됩니다.

함수 프로토타입

```
pgtle.extension_update_paths(name)
```

역할

없음.

인수

name - 업그레이드 경로를 가져올 TLE 확장의 이름입니다.

출력

- source - 업데이트의 소스 버전입니다.
- target - 업데이트의 대상 버전입니다.
- path - TLE 확장을 source 버전에서 target 버전으로 업데이트하는 데 사용되는 업그레이드 경로입니다(예: 0.1--0.2).

사용 예

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

install_extension 함수를 사용하면 데이터베이스에 TLE 확장을 구성하는 아티팩트를 설치한 다음 CREATE EXTENSION 명령을 사용하여 TLE 확장을 생성할 수 있습니다.

함수 프로토타입

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

역할

없음.

인수

- name - TLE 확장의 이름입니다. 이 값은 CREATE EXTENSION 호출에 사용됩니다.
- version - TLE 확장의 버전입니다.
- description - TLE 확장에 대한 자세한 설명입니다. 이 설명은 pgtle.available_extensions()의 comment 필드에 표시됩니다.
- ext - TLE 확장의 콘텐츠입니다. 이 값에는 함수와 같은 객체가 포함됩니다.
- requires - 이 TLE 확장의 종속성을 지정하는 선택적 파라미터입니다. pg_tle 확장은 종속성으로 자동 추가됩니다.

이러한 인수 중 다수는 PostgreSQL 인스턴스의 파일 시스템에 PostgreSQL 확장을 설치하기 위한 확장 제어 파일에 포함된 인수와 동일합니다. PostgreSQL 확장에 대한 자세한 내용은 PostgreSQL 설명서에서 Packaging Related Objects into an Extension의 Extension Files를 참조하세요.

출력

이 함수는 성공 시 OK를, 오류 시 NULL을 반환합니다.

- OK - TLE 확장이 데이터베이스에 성공적으로 설치되었습니다.
- NULL - TLE 확장이 데이터베이스에 성공적으로 설치되지 않았습니다.

사용 예

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

`install_update_path` 함수는 서로 다른 두 버전의 TLE 확장 간의 업데이트 경로를 제공합니다. 이 함수를 사용하면 TLE 확장의 사용자가 `ALTER EXTENSION ... UPDATE` 구문을 사용하여 버전을 업데이트할 수 있습니다.

함수 프로토타입

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

역할

`pgtle_admin`

인수

- `name` - TLE 확장의 이름입니다. 이 값은 `CREATE EXTENSION` 호출에 사용됩니다.
- `fromvers` - 업그레이드를 위한 TLE 확장의 소스 버전입니다.

- tovers - 업그레이드를 위한 TLE 확장의 대상 버전입니다.
- ext - 업데이트의 콘텐츠입니다. 이 값에는 함수와 같은 객체가 포함됩니다.

출력

없음.

사용 예

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

register_feature 함수는 지정된 내부 PostgreSQL 기능을 pgtle.feature_info 테이블에 추가합니다. PostgreSQL 후크는 내부 PostgreSQL 기능의 예입니다. 신뢰할 수 있는 언어 확장 개발 키트는 PostgreSQL 후크 사용을 지원합니다. 현재 이 함수는 다음 기능을 지원합니다.

- passcheck - PostgreSQL의 암호 확인 동작을 사용자 지정하는 프로시저 또는 함수에 암호 확인 후크를 등록합니다.

함수 프로토타입

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

역할

pgtle_admin

인수

- proc - 기능에 사용할 저장 프로시저 또는 함수의 이름입니다.
- feature - 함수에 등록할 pg_tle 기능(예: passcheck)의 이름입니다.

출력

없음.

사용 예

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

`pgtle.register_feature_if_not_exists` 함수는 지정된 PostgreSQL 기능을 `pgtle.feature_info` 테이블에 추가하고 해당 기능을 사용하는 TLE 확장 또는 기타 프로시저나 함수를 식별합니다. 후크 및 신뢰할 수 있는 언어 확장에 대한 자세한 내용은 TLE 확장과 함께 PostgreSQL 후크 사용 섹션을 참조하세요.

함수 프로토타입

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

역할

`pgtle_admin`

인수

- `proc` - TLE 확장을 위한 기능으로 사용할 로직(코드)이 포함된 저장 프로시저 또는 함수의 이름입니다. pw_hook 코드가 그 예입니다.
- `feature` - TLE 함수에 등록할 PostgreSQL 기능의 이름입니다. 현재 사용 가능한 기능은 `passcheck` 후크뿐입니다. 자세한 내용은 암호-확인 후크(passcheck) 단원을 참조하십시오.

출력

지정된 확장에 대한 기능을 등록한 후 `true`를 반환합니다. 기능이 이미 등록된 경우 `false`를 반환합니다.

사용 예

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

set_default_version 함수를 사용하면 TLE 확장의 default_version을 지정할 수 있습니다. 이 함수를 사용하여 업그레이드 경로를 정의하고 해당 버전을 TLE 확장의 기본값으로 지정할 수 있습니다. 데이터베이스 사용자가 CREATE EXTENSION 및 ALTER EXTENSION ... UPDATE 명령에서 TLE 확장을 지정하면 해당 버전의 TLE 확장이 해당 사용자의 데이터베이스에 생성됩니다.

이 함수는 성공 시 true를 반환합니다. name 인수에 지정된 TLE 확장자가 없는 경우에는 함수가 오류를 반환합니다. 마찬가지로 TLE 확장의 version이 존재하지 않으면 오류가 반환됩니다.

함수 프로토타입

```
pgtle.set_default_version(name text, version text)
```

역할

pgtle_admin

인수

- name - TLE 확장의 이름입니다. 이 값은 CREATE EXTENSION 호출에 사용됩니다.
- version - 기본값을 설정할 TLE 확장의 버전입니다.

출력

- true - 기본 버전 설정에 성공하면 함수가 true를 반환합니다.
- ERROR - 지정된 이름 또는 버전의 TLE 확장이 존재하지 않는 경우 오류 메시지를 반환합니다.

사용 예

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

uninstall_extension 함수는 데이터베이스에서 TLE 확장의 모든 버전을 제거합니다. 이 함수는 CREATE EXTENSION의 이후 호출이 TLE 확장을 설치하는것을 방지합니다. 데이터베이스에 TLE 확장자가 없으면 오류가 발생합니다.

uninstall_extension 함수는 현재 데이터베이스에서 활성 상태인 TLE 확장은 제거하지 않습니다. 현재 활성 상태인 TLE 확장을 제거하려면 명시적으로 DROP EXTENSION을 호출하여 제거해야 합니다.

함수 프로토타입

```
pgtle.uninstall_extension(extname text)
```

역할

pgtle_admin

인수

- extname - 제거할 TLE 확장의 이름입니다. 이 이름은 지정된 데이터베이스에서 사용할 TLE 확장을 로드하기 위해 CREATE EXTENSION과 함께 사용되는 이름과 같습니다.

출력

없음.

사용 예

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

uninstall_extension(name, version) 함수는 지정된 버전의 TLE 확장을 데이터베이스에서 제거합니다. 이 기능은 CREATE EXTENSION 및 ALTER EXTENSION이 TLE 확장을 설치하거나 지정된 버전으로 업데이트하는 것을 방지합니다. 이 함수는 TLE 확장의 모든 업데이트 경로도 제공합니다. 이 함수는 현재 데이터베이스에서 활성 상태인 TLE 확장은 제거하지 않습니다. TLE 확장을 제거하려면 명시적으로 DROP EXTENSION을 호출해야 합니다. TLE 확장의 모든 버전을 제거하려면 pgtle.uninstall_extension(name)을 참조하세요.

함수 프로토타입

```
pgtle.uninstall_extension(extname text, version text)
```

역할

pgtle_admin

인수

- extname - TLE 확장의 이름입니다. 이 값은 CREATE EXTENSION 호출에 사용됩니다.
- version - 데이터베이스에서 제거할 TLE 확장의 버전입니다.

출력

없음.

사용 예

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

uninstall_extension_if_exists 함수는 지정된 데이터베이스에서 TLE 확장의 모든 버전을 제거합니다. TLE 확장자가 존재하지 않는 경우 함수는 아무것도 반환하지 않습니다(오류 메시지가 표시되지 않음). 지정된 확장이 현재 데이터베이스 내에서 활성 상태인 경우 이 함수는 해당 확장을 삭제하지 않습니다. 이 함수를 사용하여 아티팩트를 제거하려면 먼저 명시적으로 DROP EXTENSION을 호출하여 TLE 확장을 제거해야 합니다.

함수 프로토타입

```
pgtle.uninstall_extension_if_exists(extname text)
```

역할

pgtle_admin

인수

- extname - TLE 확장의 이름입니다. 이 값은 CREATE EXTENSION 호출에 사용됩니다.

출력

uninstall_extension_if_exists 함수는 지정된 확장을 제거한 true를 반환합니다. 지정된 확장이 없는 경우 함수는 false를 반환합니다.

- true - TLE 확장을 제거한 후 true를 반환합니다.
- false - 데이터베이스에 TLE 확장자가 없으면 false를 반환합니다.

사용 예

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

`uninstall_update_path` 함수는 TLE 확장에서 특정 업데이트 경로를 제거합니다. 이렇게 하면 `ALTER EXTENSION ... UPDATE TO`가 이 경로를 업데이트 경로로 사용할 수 없습니다.

이 업데이트 경로의 버전 중 하나가 현재 사용 중인 TLE 확장은 데이터베이스에 남아 있습니다.

지정한 업데이트 경로가 존재하지 않는 경우 이 함수는 오류를 반환합니다.

함수 프로토타입

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

역할

`pgtle_admin`

인수

- `extname` - TLE 확장의 이름입니다. 이 값은 `CREATE EXTENSION` 호출에 사용됩니다.
- `fromvers` - 업데이트 경로에서 사용할 TLE 확장의 소스 버전입니다.
- `tovers` - 업데이트 경로에서 사용할 TLE 확장의 대상 버전입니다.

출력

없음.

사용 예

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

`uninstall_update_path_if_exists` 함수는 TLE 확장에서 지정된 업데이트 경로를 제거한다는 점에서 `uninstall_update_path`와 비슷합니다. 하지만 업데이트 경로가 존재하지 않는 경우 이 함수는 오류 메시지를 표시하지 않습니다. 대신 함수는 `false`를 반환합니다.

함수 프로토타입

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

역할

pgtle_admin

인수

- extname - TLE 확장의 이름입니다. 이 값은 CREATE EXTENSION 호출에 사용됩니다.
- fromvers - 업데이트 경로에서 사용할 TLE 확장의 소스 버전입니다.
- tovers - 업데이트 경로에서 사용할 TLE 확장의 대상 버전입니다.

출력

- true - 함수가 TLE 확장 경로를 성공적으로 업데이트했습니다.
- false - 함수가 TLE 확장의 경로를 업데이트하지 못했습니다.

사용 예

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

unregister_feature 함수는 후크 등 pg_tle 기능을 사용하도록 등록된 함수를 제거하는 방법을 제공합니다. 기능 등록에 대한 자세한 내용은 pgtle.register_feature 섹션을 참조하세요.

함수 프로토타입

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

역할

pgtle_admin

인수

- proc - pg_tle 기능에 등록할 저장된 함수의 이름입니다.

- feature - 함수에 등록할 pg_tle 기능의 이름입니다. 예를 들어 passcheck는 개발하는 신뢰할 수 있는 언어 확장에서 사용하도록 등록할 수 있는 기능입니다. 자세한 내용은 암호-확인 후크 (passcheck) 단원을 참조하십시오.

출력

없음.

사용 예

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

unregister_feature 함수는 후크 등 pg_tle 기능을 사용하도록 등록된 함수를 제거하는 방법을 제공합니다. 자세한 내용은 TLE 확장과 함께 PostgreSQL 후크 사용 단원을 참조하십시오. 기능을 성공적으로 등록 취소한 후 true를 반환합니다. 기능이 등록되지 않은 경우 false를 반환합니다.

TLE 확장의 pg_tle 기능 등록에 대한 자세한 내용은 pgtle.register_feature 섹션을 참조하세요.

함수 프로토타입

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

역할

pgtle_admin

인수

- proc - pg_tle 기능을 포함하도록 등록된 저장된 함수의 이름입니다.
- feature - 신뢰할 수 있는 언어 확장에 등록된 pg_tle 기능의 이름입니다.

출력

다음과 같이 true 또는false를 반환합니다.

- true - 함수가 확장에서 성공적으로 기능 등록을 취소했습니다.
- false - 함수가 TLE 확장에서 기능을 등록 취소하지 못했습니다.

사용 예

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

PostgreSQL용 신뢰할 수 있는 언어 확장에 대한 후크 참조

PostgreSQL용 신뢰할 수 있는 언어 확장은 PostgreSQL 후크를 지원합니다. 후크는 PostgreSQL의 핵심 기능을 확장하기 위해 개발자가 사용할 수 있는 내부 콜백 메커니즘입니다. 개발자는 후크를 사용하여 다양한 데이터베이스 작업 중에 사용할 자체 함수 또는 프로시저를 구현하여 PostgreSQL의 동작을 일정 방식으로 수정할 수 있습니다. 예를 들어 `passcheck` 후크를 사용하면 사용자(역할)의 암호를 생성하거나 변경할 때 제공된 암호를 PostgreSQL이 처리하는 방법을 사용자 지정할 수 있습니다.

TLE 확장에 사용할 수 있는 암호 확인 후크에 대해 알아보려면 다음 설명서를 참조하세요. 클라이언트 인증 후크를 포함하여 사용 가능한 후크에 대한 자세한 내용은 Trusted Language Extensions hooks를 참조하세요.

암호-확인 후크(passcheck)

`passcheck` 후크는 다음 SQL 명령 및 `psql` 메타 명령에 대한 암호 확인 프로세스 도중 PostgreSQL 동작을 사용자 지정하는 데 사용됩니다.

- `CREATE ROLE` *`username`* `...PASSWORD` - 자세한 내용은 PostgreSQL 설명서의 CREATE ROLE을 참조하세요.
- `ALTER ROLE` *`username`*`...PASSWORD` - 자세한 내용은 PostgreSQL 설명서의 ALTER ROLE을 참조하세요.
- `\password` *`username`* - 이 대화형 `psql` 메타 명령은 `ALTER ROLE ... PASSWORD` 구문을 투명하게 사용하기 전에 암호를 해시하여 지정된 사용자의 암호를 안전하게 변경합니다. 메타 명령은 `ALTER ROLE ... PASSWORD` 명령의 보안 래퍼이므로 후크는 `psql` 메타 명령의 동작에 적용됩니다.

문제 해결 예는 암호 확인 후크 코드 목록을(를) 참조하세요.

목차

함수 프로토타입

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

인수

passcheck 후크 함수는 다음 인수를 사용합니다.

- username - 암호를 설정하는 역할(사용자 이름)의 이름(텍스트)입니다.
- password - 일반 텍스트 또는 해시된 암호입니다. 입력한 암호는 password_type에서 지정한 유형과 일치해야 합니다.
- password_type - 암호의 pgtle.password_type 형식을 지정합니다. 이 형식은 다음 옵션 중 하나일 수 있습니다.
  - PASSWORD_TYPE_PLAINTEXT - 일반 텍스트 암호입니다.
  - PASSWORD_TYPE_MD5 - MD5(message digest 5) 알고리즘을 사용하여 해시된 암호입니다.
  - PASSWORD_TYPE_SCRAM_SHA_256 - SCRAM-SHA-256 알고리즘을 사용하여 해시된 암호입니다.
- valid_until - 암호가 무효가 되는 시간을 지정합니다. 이 인수는 선택 사항입니다. 이 인수를 사용하는 경우 시간을 timestamptz 값으로 지정하세요.
- valid_null - 이 부울이 true로 설정된 경우 valid_until 옵션은 NULL로 설정됩니다.

구성

함수 pgtle.enable_password_check는 암호 확인 후크가 활성 상태인지 여부를 제어합니다. 암호 확인 후크에서 세 가지 설정이 가능합니다.

- off - passcheck 암호 확인 후크를 해제합니다. 이것이 기본값입니다.
- on - 테이블과 비교해 암호를 검사할 수 있도록 passcode 암호 확인 후크를 켭니다.
- require - 암호 확인 후크를 정의해야 합니다.

사용 노트

passcheck 후크를 켜거나 해제하려면 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 대한 사용자 지정 DB 파라미터 그룹을 수정해야 합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Amazon Aurora PostgreSQL 참조

다음 주제에는 Amazon Aurora PostgreSQL의 데이터 정렬, 함수, 파라미터, 대기 이벤트에 관한 정보가 나와 있습니다.

주제

EBCDIC 및 기타 메인프레임 마이그레이션을 위한 Aurora PostgreSQL 데이터 정렬

애플리케이션 동작을 AWS와 같이 이상적으로 보존하는 것과 같이 메인프레임 애플리케이션을 새로운 플랫폼으로 마이그레이션합니다. 새 플랫폼에서의 애플리케이션 동작을 메인프레임과 동일하게 유지하려면 마이그레이션된 데이터가 동일한 데이터 정렬 및 정렬 규칙을 사용하여 데이터 정렬되어야 합니다. 예를 들어 많은 Db2 마이그레이션 솔루션은 null 값을 u0180(Unicode 위치 0180) 으로 이동하므로 이러한 데이터 정렬은 u0180을 먼저 정렬합니다. 데이터 정렬이 메인프레임 소스와 차이점 및 원본 EBCDIC 데이터 정렬에 더 잘 매핑되는 데이터 정렬을 선택해야 하는 이유를 보여주는 한 가지 예입니다.

Aurora PostgreSQL 14.3 이상 버전은 많은 ICU 및 EBCDIC 데이터 정렬을 제공하여 AWS Mainframe Modernization 서비스를 사용하는 AWS로 마이그레이션과 같은 서비스를 지원합니다. 이 서비스에 대한 자세한 내용을 알아보려면 AWS Mainframe Modernization이란? 단원을 참조하세요.

다음 테이블에서 Aurora PostgreSQL 제공 데이터 정렬을 찾을 수 있습니다. 이러한 데이터 정렬은 EBCDIC 규칙을 따르고 메인프레임 애플리케이션이 메인프레임 환경에서와 마찬가지로 AWS에서도 동일하게 작동하도록 합니다. 데이터 정렬 이름에는 관련 코드 페이지(cpnnnn)가 포함되어 있으므로 메인프레임 소스에 적합한 데이터 정렬을 선택할 수 있습니다. 예를 들어 `en-US-cp037-x-icu`를 사용하여 코드 페이지 03을 사용한 메인프레임 애플리케이션에서 생성된 EBCDIC 데이터에 대한 데이터 정렬 동작을 달성합니다.

EBDICD 데이터 정렬	AWS Blu Age 데이터 정렬	AWS Micro Focus 데이터 정렬
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142b-x-icu
da-DK-cp277-x-icu	da-DK-cp1142b-x-icu	–
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

AWS Blu Age에 대한 자세한 내용은 AWS Mainframe Modernization 사용 설명서의 자습서: AWS Blu Age용 관리형 런타임을 참조하세요.

AWS Micro Focus 작업에 대한 자세한 내용은 AWS Mainframe Modernization 사용 설명서의 [자습서: Micro Focus]를 위한 관리형 런타임을 참조하십시오.

PostgreSQL의 데이터 정렬에 대한 자세한 내용은 PostgreSQL 설명서의 [데이터 정렬 지원]을 참조하세요.

Aurora PostgreSQL에서 지원되는 데이터 정렬

데이터 정렬은 데이터베이스에 저장된 문자열을 정렬하고 비교하는 방법을 결정하는 일련의 규칙입니다. 데이터 정렬은 컴퓨터 시스템에서 기본적인 역할을 하며 운영 체제의 일부로 포함됩니다. 새 문자가 언어에 추가되거나 순서 지정 규칙이 변경되면 시간이 지남에 따라 데이터 정렬이 변경됩니다.

데이터 정렬 라이브러리는 데이터 정렬에 대한 특정 규칙 및 알고리즘을 정의합니다. PostgreSQL에서 가장 많이 사용되는 데이터 정렬 라이브러리는 GNU C(glibc) 및 유니코드용 국제화 구성 요소(ICU)입니다. 기본적으로 Aurora PostgreSQL은 멀티바이트 문자 시퀀스에 대한 유니코드 문자 정렬 순서를 포함하는 glibc 데이터 정렬을 사용합니다.

새로운 Aurora PostgreSQL DB 클러스터를 만들면 운영 체제에서 사용 가능한 데이터 정렬을 확인합니다. `CREATE DATABASE` 명령의 PostgreSQL 파라미터인 `LC_COLLATE` 및 `LC_CTYPE`은 해당 데이터베이스의 기본 데이터 정렬로 사용되는 데이터 정렬을 지정하는 데 사용됩니다. 또는 `CREATE DATABASE`의 `LOCALE` 파라미터를 사용하여 이러한 파라미터를 설정할 수도 있습니다. 이는 데이터베이스의 문자열에 대한 기본 데이터 정렬과 문자를 문자, 숫자 또는 기호로 분류하는 규칙을 결정합니다. 열, 색인 또는 쿼리에 사용할 데이터 정렬을 선택할 수도 있습니다.

Aurora PostgreSQL은 데이터 정렬 지원을 위해 운영 체제의 glibc 라이브러리를 사용합니다. Aurora PostgreSQL 인스턴스는 최신 버전의 운영 체제로 정기적으로 업데이트됩니다. 이러한 업데이트에는 glibc 라이브러리의 최신 버전도 포함되는 경우가 있습니다. 드물게 최신 버전의 glibc에서는 일부 문자의 정렬 순서 또는 데이터 정렬이 변경되어 데이터가 다르게 정렬되거나 잘못된 색인 항목이 생성될 수 있습니다. 업데이트 중에 데이터 정렬의 정렬 순서 문제가 발견되면 색인을 다시 작성해야 할 수 있습니다.

glibc 업데이트의 영향을 줄이기 위해 Aurora PostgreSQL에는 이제 독립적인 기본 데이터 정렬 라이브러리가 포함되어 있습니다. 이 데이터 정렬 라이브러리는 Aurora PostgreSQL 14.6, 13.9, 12.13, 11.18 및 최신 마이너 버전 릴리스에서 사용할 수 있습니다. glibc 2.26-59.amzn2와 호환되며 정렬 순서 안정성을 제공하여 잘못된 쿼리 결과가 나오지 않도록 방지합니다.

Aurora PostgreSQL 함수 참조

다음에서 Aurora PostgreSQL 호환 버전 DB 엔진을 실행하는 Aurora DB 클러스터에 사용할 수 있는 Aurora PostgreSQL 함수 목록을 찾을 수 있습니다. 이러한 Aurora PostgreSQL 함수는 표준

PostgreSQL 함수에 추가됩니다. 표준 PostgreSQL 함수에 대한 자세한 내용은 PostgreSQL—함수와 연산자를 참조하세요.

개요

Aurora PostgreSQL을 실행하는 Amazon RDS DB 인스턴스에 다음 함수를 사용할 수 있습니다.

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status
- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

연결된 DB 인스턴스의 이름을 보고합니다.

구문

```
aurora_db_instance_identifier()
```

인수

없음

반환 타입

VARCHAR 문자열

사용 노트

이 함수는 데이터베이스 클라이언트 또는 애플리케이션 연결을 위한 Aurora PostgreSQL-Compatible Edition 클러스터의 DB 인스턴스 이름을 표시합니다.

이 함수는 Aurora PostgreSQL 버전 13.7, 12.11, 11.16, 10.21 및 기타 모든 이후 버전의 릴리스부터 사용할 수 있습니다.

예시

다음 예제에서는 `aurora_db_instance_identifier` 함수 호출의 결과를 보여줍니다.

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
--------------------------------
 test-my-instance-name
```

이 함수의 결과를 `aurora_replica_status` 함수와 결합하여 연결을 위한 DB 인스턴스에 대한 세부 정보를 얻을 수 있습니다. aurora_replica_status 단독으로는 사용 중인 DB 인스턴스가 표시되지 않습니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다.

```
=> SELECT *
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]---------------------+-----------------------
server_id                         | test-my-instance-name
```

```
session_id                          | MASTER_SESSION_ID
durable_lsn                         | 88492069
highest_lsn_rcvd                    |
current_read_lsn                    |
cur_replay_latency_in_usec          |
active_txns                         |
is_current                          | t
last_transport_error                | 0
last_error_timestamp                |
last_update_timestamp               | 2022-06-03 11:18:25+00
feedback_xmin                       |
feedback_epoch                      |
replica_lag_in_msec                 |
log_stream_speed_in_kib_per_second  | 0
log_buffer_sequence_number          | 0
oldest_read_view_trx_id             |
oldest_read_view_lsn                |
pending_read_ios                    | 819
```

aurora_ccm_status

클러스터 캐시 관리자의 상태를 표시합니다.

구문

```
aurora_ccm_status()
```

인수

없음.

반환 타입

다음 열이 있는 SETOF 레코드:

- buffers_sent_last_minute - 지난 1분 동안 지정된 리더로 전송된 버퍼 수입니다.
- buffers_found_last_minute - 지난 1분 동안 식별된 자주 액세스된 버퍼 수입니다.
- buffers_sent_last_scan – 버퍼 캐시의 마지막 전체 검색 중에 지정된 리더로 전송된 버퍼 수입니다.
- buffers_found_last_scan – 버퍼 캐시의 마지막 전체 검색 중에 자주 액세스된 버퍼 수입니다. 지정된 리더에 이미 캐시된 버퍼는 전송되지 않습니다.

- `buffers_sent_current_scan` - 현재 스캔 중에 전송된 버퍼 수입니다.
- `buffers_found_current_scan` - 현재 스캔에서 식별된 자주 액세스된 버퍼 수입니다.
- `current_scan_progress` - 현재 스캔 중에 지금까지 방문한 버퍼 수입니다.

사용 노트

이 함수를 사용하여 클러스터 캐시 관리(CCM) 기능을 확인하고 모니터링할 수 있습니다. 이 함수는 Aurora PostgreSQL DB 클러스터에서 CCM이 활성화된 경우에만 작동합니다. 이 함수를 사용하려면 Aurora PostgreSQL DB 클러스터의 Write DB 인스턴스에 연결합니다.

클러스터의 Custom DB 클러스터 파라미터 그룹에서 `apg_ccm_enabled`를 1로 설정하여 Aurora PostgreSQL DB 클러스터에 대한 CCM을 켭니다. 자세한 방법은 클러스터 캐시 관리 구성을 참조하세요.

클러스터에 Aurora Reader 인스턴스가 다음과 같이 구성된 경우 Aurora PostgreSQL DB 클러스터에서 클러스터 캐시 관리가 활성화됩니다.

- Aurora Reader 인스턴스는 클러스터의 Writer 인스턴스와 동일한 DB 인스턴스 클래스 유형 및 크기를 사용합니다.
- Aurora Reader 인스턴스는 클러스터의 Tier-0으로 구성됩니다. 클러스터에 둘 이상의 Reader가 있는 경우 이게 유일한 Tier-0 Reader입니다.

둘 이상의 Reader를 Tier-0으로 설정하면 CCM이 사용 중지됩니다. CCM이 사용 중지된 경우 이 함수를 호출하면 다음과 같은 오류 메시지가 반환됩니다.

```
ERROR: Cluster Cache Manager is disabled
```

또한 PostgreSQL pg_buffercache 확장 프로그램을 사용하여 버퍼 캐시를 분석할 수 있습니다. 자세한 내용은 PostgreSQL 설명서의 pg_buffercache를 참조하세요.

자세한 내용은 Introduction to Aurora PostgreSQL cluster cache management(Aurora PostgreSQL 클러스터 캐시 관리 소개)를 참조하세요.

예시

다음 예제에서는 `aurora_ccm_status` 함수 호출의 결과를 보여줍니다. 이 첫 번째 예제에서는 CCM 통계를 보여줍니다.

```
=> SELECT * FROM aurora_ccm_status();
```

```
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+-----------------------
+-------------------------+---------------------------+--------------------------
+-----------------------
                  2242000 |                   2242003 |               17920442 |
         17923410 |                14098000 |                  14100964 |
   15877443
```

보다 자세한 내용을 보려면 다음과 같이 확장된 디스플레이를 사용하면 됩니다.

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]-----------------------+---------
buffers_sent_last_minute           | 2242000
buffers_found_last_minute          | 2242003
buffers_sent_last_scan             | 17920442
buffers_found_last_scan            | 17923410
buffers_sent_current_scan          | 14098000
buffers_found_current_scan         | 14100964
current_scan_progress              | 15877443
```

이 예제에서는 웜 비율과 웜 백분율을 확인하는 방법을 보여줍니다.

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

Aurora 전역 DB 클러스터의 복제본을 포함하여 모든 Aurora 인스턴스의 상태를 표시합니다.

구문

```
aurora_global_db_instance_status()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- server_id - DB 인스턴스의 식별자입니다.
- session_id - 현재 세션에 대한 고유한 식별자입니다. MASTER_SESSION_ID의 값은 Writer(프라이머리) DB 인스턴스를 식별합니다.
- aws_region - 이 전역 DB 인스턴스가 실행되는 AWS 리전입니다. 리전 목록은 리전 가용성 섹션을 참조하세요.
- durable_lsn - 스토리지에서 내구성이 뛰어난 로그 시퀀스 번호(LSN)입니다. LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
- highest_lsn_rcvd - 라이터 DB 인스턴스에서 DB 인스턴스가 수신한 가장 높은 LSN입니다.
- feedback_epoch – DB 인스턴스가 상시 대기 방식 정보를 생성할 때 사용하는 epoch입니다. 상시 대기 방식은 프라이머리 DB가 복구 또는 대기 모드에 있는 동안 연결 및 쿼리를 지원하는 DB 인스턴스입니다. 상시 대기 방식 정보에는 epoch(시점)와 상시 대기 방식으로 사용되는 DB 인스턴스에 대한 기타 세부 정보가 포함됩니다. 자세한 내용은 PostgreSQL 설명서의 핫 스탠바이를 참조하세요.
- feedback_xmin - DB 인스턴스가 사용하는 최소(가장 오래된) 활성 트랜잭션 ID입니다.
- oldest_read_view_lsn – DB 인스턴스가 스토리지에서 읽는 데 사용하는 가장 오래된 LSN입니다.
- visibility_lag_in_msec - 이 DB 인스턴스가 라이터 DB 인스턴스보다 지연된 시간(밀리초)입니다.

사용 노트

이 함수는 Aurora DB 클러스터에 대한 복제 통계를 보여줍니다. 클러스터의 각 Aurora PostgreSQL DB 인스턴스에 대해 이 함수는 글로벌 데이터베이스 구성에서 크로스 리전 복제본을 포함하는 데이터 행을 표시합니다.

Aurora PostgreSQL DB 클러스터의 인스턴스나 Aurora PostgreSQL 전역 데이터베이스의 인스턴스에서 이 함수를 실행할 수 있습니다. 이 함수는 모든 복제본 인스턴스의 지연에 대한 세부 정보를 반환합니다.

이 함수(`aurora_global_db_instance_status`)를 사용하거나 `aurora_global_db_status`를 사용하여 지연 모니터링에 대해 자세히 알아보려면 [Aurora PostgreSQL 기반 글로벌 데이터베이스 모니터링]() 섹션을 참조하세요.

Aurora 전역 데이터베이스에 대한 자세한 내용은 [Amazon Aurora Global Database 개요]() 섹션을 참조하세요.

Aurora 전역 데이터베이스를 시작하려면 [Amazon Aurora Global Database 시작하기]() 섹션 또는 [Amazon Aurora FAQ]()를 참조하세요.

예시

이 예시에서는 크로스 리전 인스턴스 통계를 보여줍니다.

```
=> SELECT *
  FROM aurora_global_db_instance_status();
                server_id                 |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
------------------------------------------+--------------------------------------
+--------------+-------------+------------------+----------------+---------------
+----------------------+------------------------
 db-119-001-instance-01                   | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                 [NULL]
 db-119-001-instance-02                   | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-03                   | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-04                   | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-instance-05                   | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-global-instance-1             | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    996
 db-119-001-global-instance-1             | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     14
```

```
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     0
```

이 예제에서는 밀리초 단위로 전역 복제본 지연을 확인하는 방법을 보여줍니다.

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |                server_id                |
 visibility_lag_in_msec
-------------+--------------+-----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                  |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1            |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                  |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                  |
      9
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

이 예제에서는 전역 데이터베이스 구성에서 AWS 리전당 최소, 최대 및 평균 지연을 확인하는 방법을 보여줍니다.

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
```

```
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
   FROM aurora_global_db_instance_status()
  WHERE aws_region NOT IN (SELECT   aws_region
                             FROM aurora_global_db_instance_status()
                            WHERE session_id='MASTER_SESSION_ID')
                         GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

Aurora Global Database 지연의 다양한 측면, 특히 기본 Aurora 스토리지의 지연(내구성 지연) 및 Recovery Point Objective(RPO) 간의 지연에 대한 정보를 표시합니다.

구문

```
aurora_global_db_status()
```

인수

없음.

반환 타입

다음 열이 있는 SETOF 레코드:

- `aws_region` - 이 DB 클러스터가 있는 AWS 리전입니다. 엔진별 AWS 리전의 전체 목록을 보려면 리전 및 가용 영역 섹션을 참조하세요.

- `highest_lsn_written` - 이 DB 클러스터에 현재 존재하는 가장 높은 로그 시퀀스 번호(LSN)입니다. LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
- `durability_lag_in_msec` - 보조 DB 클러스터의 `highest_lsn_written`과 기본 DB 클러스터의 `highest_lsn_written` 간 타임스탬프 값 차이입니다. -1 값은 Aurora Global Database의 기본 DB 클러스터를 식별합니다.
- `rpo_lag_in_msec` - Recovery Point Objective(RPO) 지연입니다. RPO 지연은 Aurora Global Databse의 기본 DB 클러스터에 저장된 후 가장 최근의 사용자 트랜잭션 COMMIT 보조 DB 클러스터에 저장하는 데 걸리는 시간입니다. -1 값은 기본 DB 클러스터를 나타냅니다. 따라서 지연은 관련이 없습니다.

  간단히 말해서 이 지표는 Aurora Global Databse의 각 Aurora PostgreSQL DB 클러스터에 대한 복구 시점 목표, 즉 중단 시 손실될 수 있는 데이터의 양을 계산합니다. 지연과 마찬가지로 RPO는 시간 단위로 측정됩니다.
- `last_lag_calculation_time` - `durability_lag_in_msec` 및 `rpo_lag_in_msec`에 대한 값이 마지막으로 계산된 시점을 지정하는 타임스탬프입니다. 시간 값(예: `1970-01-01 00:00:00+00`)은 이것이 기본 DB 클러스터임을 의미합니다.
- `feedback_epoch` – 보조 DB 클러스터가 상시 대기 방식 정보를 생성할 때 사용하는 epoch입니다. 상시 대기 방식은 프라이머리 DB가 복구 또는 대기 모드에 있는 동안 연결 및 쿼리를 지원하는 DB 인스턴스입니다. 상시 대기 방식 정보에는 epoch(시점)와 상시 대기 방식으로 사용되는 DB 인스턴스에 대한 기타 세부 정보가 포함됩니다. 자세한 내용은 PostgreSQL 설명서의 [핫 스탠바이]를 참조하세요.
- `feedback_xmin` – 보조 DB 클러스터가 사용하는 최소(가장 오래된) 활성 트랜잭션 ID입니다.

사용 노트

현재 사용 가능한 모든 Aurora PostgreSQL 버전에서 이 함수를 지원합니다. 이 함수는 Aurora 전역 데이터베이스에 대한 복제 통계를 보여줍니다. Aurora PostgreSQL 전역 데이터베이스의 각 DB 클러스터에 대해 하나의 행을 표시합니다. Aurora PostgreSQL 전역 데이터베이스의 인스턴스에서 이 함수를 실행할 수 있습니다.

표시되는 데이터 지연인 Aurora 전역 데이터베이스 복제 지연을 평가하려면aurora_global_db_instance_status 섹션을 참조하세요.

`aurora_global_db_status` 및 `aurora_global_db_instance_status`를 사용하여 Aurora 전역 데이터베이스 지연을 모니터링하는 방법에 대해 자세히 알아보려면 Aurora PostgreSQL 기반 글

로벌 데이터베이스 모니터링 섹션을 참조하세요. Aurora 전역 데이터베이스에 대한 자세한 내용은 Amazon Aurora Global Database 개요 섹션을 참조하세요.

예시

이 예제에서는 교차 리전 스토리지 통계를 표시하는 방법을 보여줍니다.

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+----------------------------+----------------+---------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |              0 |             0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
```

aurora_list_builtins

사용 가능한 모든 Aurora PostgreSQL 내장 함수를 간략한 설명 및 함수 세부 정보와 함께 나열합니다.

구문

```
aurora_list_builtins()
```

인수

없음

반환 타입

SETOF 레코드

예시

다음 예제에서는 `aurora_list_builtins` 함수 호출의 결과를 보여줍니다.

```
=> SELECT *
FROM aurora_list_builtins();

              Name                | Result data type |                   Argument data
 types                    | Type | Volatility |  Parallel  | Security |
            Description
-----------------------------------+------------------
+-------------------------------------------------------------+------+------------
+------------+----------
+---------------------------------------------------------------------
 aurora_version                   | text             |
                            | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type            | SETOF record     | OUT type_id smallint, OUT
 type_name text                | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event           | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                  |                  |.me text
                           |      |            |            |          |
 aurora_list_builtins             | SETOF record     | OUT "Name" text, OUT "Result
 data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
 Aurora built-in functions
                                  |                  |.ent data types" text, OUT
 "Type" text, OUT "Volatility" .|      |            |            |          |
                                  |                  |.text, OUT "Parallel" text, OUT
 "Security" text, OUT "Des.|      |            |            |          |
                                  |                  |.cription" text
                           |      |            |            |          |
 .
 .
 .
 aurora_stat_file                 | SETOF record     | OUT filename text, OUT
 allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
 all files present in Aurora storage
                                  |                  |.bytes bigint
                           |      |            |            |          |
 aurora_stat_get_db_commit_latency | bigint           | oid
                            | func | stable     | restricted | invoker  | Per DB commit
 latency in microsecs
```

aurora_replica_status

모든 Aurora PostgreSQL 리더 노드의 상태를 표시합니다.

구문

```
aurora_replica_status()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- `server_id` - DB 인스턴스 ID(식별자)
- `session_id` - 현재 세션의 고유 식별자로, 다음과 같이 기본 인스턴스 및 리더 인스턴스에 대해 반환됩니다.
  - 기본 인스턴스의 경우`session_id`는 항상 ``MASTER_SESSION_ID'`입니다.
  - 리더 인스턴스의 경우`session_id`는 항상 리더 인스턴스의 UUID(Universally Unique Identifier)입니다.
- `durable_lsn` - 스토리지에 저장된 LSN(로그 시퀀스 번호)입니다.
  - 기본 볼륨의 경우 현재 적용 중인 기본 볼륨 내구성 LSN(VDL)입니다.
  - 보조 볼륨의 경우 보조 볼륨이 성공적으로 적용된 기본 볼륨의 VDL입니다.

> **Note**
>
> LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 시퀀스에서 발생한 트랜잭션을 나타내도록 정렬됩니다.

- `highest_lsn_rcvd` - 라이터 DB 인스턴스에서 DB 인스턴스가 수신한 가장 높은(최신) LSN입니다.
- `current_read_lsn` - 모든 리더에 적용된 최신 스냅샷의 LSN입니다.
- `cur_replay_latency_in_usec` - 보조 볼륨에서 로그를 재생하는 데 걸리는 시간(마이크로초)입니다.

- `active_txns` - 현재 활성 DML 트랜잭션의 수입니다.
- `is_current` - 사용되지 않습니다.
- `last_transport_error` - 마지막 복제 오류 코드입니다.
- `last_error_timestamp` - 마지막 복제 오류의 타임스탬프입니다.
- `last_update_timestamp` - 복제본 상태에 대한 마지막 업데이트의 타임스탬프입니다. Aurora PostgreSQL 13.9부터는 연결된 DB 인스턴스의 `last_update_timestamp` 값이 NULL로 설정됩니다.
- `feedback_xmin` - 복제본의 상시 대기 feedback_xmin입니다. DB 인스턴스가 사용하는 최소(가장 오래된) 활성 트랜잭션 ID입니다.
- `feedback_epoch` – DB 인스턴스가 상시 대기 정보를 생성할 때 사용하는 Epoch입니다.
- `replica_lag_in_msec` - 리더 인스턴스가 라이터 인스턴스보다 지연되는 시간(밀리초)입니다.
- `log_stream_speed_in_kib_per_second` - 로그 스트림 처리량(초당 KB)입니다.
- `log_buffer_sequence_number` - 로그 버퍼 시퀀스 번호입니다.
- `oldest_read_view_trx_id` - 사용되지 않습니다.
- `oldest_read_view_lsn` – DB 인스턴스가 스토리지에서 읽는 데 사용하는 가장 오래된 LSN입니다.
- `pending_read_ios` - 복제본에서 여전히 보류 중인 미해결 페이지 읽기입니다.
- `read_ios` - 복제본의 총 페이지 읽기 횟수입니다.
- `iops` - 사용되지 않습니다.
- `cpu` - 클러스터의 각 노드에 대한 Aurora 스토리지 대몬의 CPU 사용량입니다. 인스턴스별 CPU 사용량에 대한 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표](#) 단원을 참조하세요.

사용 노트

현재 사용 가능한 모든 Aurora PostgreSQL 버전에서 이 함수를 지원합니다. `aurora_replica_status` 함수는 Aurora PostgreSQL DB 클러스터의 복제본 상태 관리자에서 값을 반환합니다. 이 함수를 사용하면 Aurora DB 클러스터의 모든 DB 인스턴스에 대한 지표를 포함하여 Aurora PostgreSQL DB 클러스터의 복제 상태에 대한 정보를 얻을 수 있습니다. 예를 들어, 다음을 수행할 수 있습니다.

- Aurora PostgreSQL DB 클러스터의 인스턴스 유형(라이터, 리더)에 대한 정보 가져오기 - 다음 열의 값을 확인하여 이 정보를 얻을 수 있습니다.

  - `server_id` - 인스턴스를 생성할 때 지정한 인스턴스의 이름을 포함합니다. 기본(라이터) 인스턴스 등 일부 경우에는 일반적으로 Aurora PostgreSQL DB 클러스터에 대해 생성한 이름에 -instance-1을 추가하여 이름이 생성됩니다.
  - `session_id` - `session_id` 필드는 인스턴스가 리더인지 라이터인지를 나타냅니다. 라이터 인스턴스의 경우 `session_id`는 항상 `"MASTER_SESSION_ID"`로 설정됩니다. 리더 인스턴스의 경우,`session_id`는 특정 리더의 `UUID`로 설정됩니다.
- 복제본 지연과 같은 일반적인 복제 문제 진단 - 복제본 지연은 리더 인스턴스의 페이지 캐시가 라이터 인스턴스의 페이지 캐시보다 뒤처지는 시간(밀리초)입니다. 이 지연은 Amazon Aurora를 사용한 복제에 설명된 대로 Aurora 클러스터에서 비동기 복제를 사용하기 때문에 발생합니다. 이 함수가 반환한 결과의 `replica_lag_in_msec` 열에 표시됩니다. 대기 서버의 복구 충돌로 인해 쿼리가 취소되는 경우에도 지연이 발생할 수 있습니다. `pg_stat_database_conflicts()`를 통해 이러한 충돌로 인해 복제본 지연이 발생하는지 확인할 수 있습니다. 자세한 내용은 PostgreSQL 설명서에서 통계 수집기를 참조하세요. 고가용성 및 복제에 대한 자세한 내용은 Amazon Aurora FAQ 섹션을 참조하세요.

  Amazon CloudWatch는 시간 경과에 따른 `replica_lag_in_msec` 결과를 `AuroraReplicaLag` 지표로 저장합니다. Aurora에 CloudWatch 지표를 사용하는 방법에 관한 자세한 내용은 Amazon CloudWatch로 Amazon Aurora 지표 모니터링 단원을 참조하세요.

Aurora 읽기 전용 복제본 및 재시작 문제 해결에 대한 자세한 내용은 AWS Support 센터의 Amazon Aurora 읽기 전용 복제본이 뒤처지고 다시 시작되는 이유는 무엇입니까? 섹션을 참조하세요.

예시

다음 예제에서는 Aurora PostgreSQL DB 클러스터에 있는 모든 인스턴스의 복제 상태를 가져오는 방법을 보여줍니다.

```
=> SELECT *
FROM aurora_replica_status();
```

다음 예제는 `docs-lab-apg-main` Aurora PostgreSQL DB 클러스터의 라이터 인스턴스를 보여줍니다.

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
```

```
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

다음 예제에서는 클러스터의 모든 리더 인스턴스를 나열합니다.

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

다음 예제에서는 모든 인스턴스, 각 인스턴스가 라이터보다 지연되는 시간 및 마지막 업데이트 이후의 기간을 나열합니다.

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

aurora_stat_activity

서버 프로세스당 하나의 행을 반환하며 해당 프로세스의 현재 활동과 관련된 정보를 표시합니다.

구문

```
aurora_stat_activity();
```

인수

없음

반환 타입

서버 프로세스당 하나의 행을 반환합니다. `pg_stat_activity` 열 외에도 다음 필드가 추가됩니다.

- planid – 계획 식별자

사용 노트

현재 쿼리 실행 계획을 보여주는 추가 `plan_id` 열과 함께 동일한 열을 반환하기 위한 `pg_stat_activity`의 보조 뷰입니다.

뷰에서 plan_id를 반환하려면 `aurora_compute_plan_id`가 활성화되어야 합니다.

이 함수는 Aurora PostgreSQL 버전 14.10 및 15.5 이상 모든 버전의 릴리스부터 사용할 수 있습니다.

예시

아래 예제 쿼리는 query_id 및 plan_id를 기준으로 상위 부하를 집계합니다.

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
```

```
(3 rows)
```

query_id에 사용되는 계획이 변경되면 aurora_stat_activity에서 새 plan_id를 보고합니다.

```
count  |  query_id             |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_waits

특정 백엔드 프로세스의 대기 활동에 대한 통계를 표시합니다.

구문

```
aurora_stat_backend_waits(pid)
```

인수

`pid` - 백엔드 프로세스의 ID입니다. `pg_stat_activity` 보기를 사용하여 프로세스 ID를 얻을 수 있습니다.

반환 타입

다음 열이 있는 SETOF 레코드:

- `type_id` - 몇 가지 예제에 이름을 지정하기 위해 경량 잠금(LWLock)용 1, 잠금용 3, 클라이언트 세션용 6과 같이 대기 이벤트 유형을 나타내는 숫자입니다. 이 함수의 결과를 예시에서 볼 수 있듯 `aurora_stat_wait_type`의 열과 결합하면 이러한 값이 의미 있게 됩니다.
- `event_id` - 대기 이벤트에 대한 식별 번호입니다. 이 값을 `aurora_stat_wait_event`의 열과 결합하여 유의미한 이벤트 이름을 획득합니다.
- `waits` - 지정한 프로세스 ID에 대해 누적된 대기 시간입니다.

- `wait_time` - 대기 시간(밀리초)입니다.

사용 노트

이 함수를 사용하여 연결이 개방된 이후 발생한 특정 백엔드(세션) 대기 이벤트를 분석할 수 있습니다. 대기 이벤트 이름 및 유형에 대한 보다 의미 있는 정보를 얻으려면 예제에 나와 있듯 JOIN을 사용하여 함수 `aurora_stat_wait_type` 및 `aurora_stat_wait_event`를 결합하면 됩니다.

예시

이 예제에서는 백엔드 프로세스 ID 3027에 대한 모든 대기, 유형 및 이벤트 이름을 보여줍니다.

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
-----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

이 예제에서는 모든 활성 세션(`pg_stat_activity state <> 'idle'`)에 대한 현재 및 누적 대기 유형과 대기 이벤트를 보여줍니다(단, 함수(`pid <> pg_backend_pid()`)를 호출하는 현재 세션은 제외).

```
=> SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
```

```
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
```

```
30109 | postgres | pgbench  | IO                | XactSync            | active         |
LWLock    | ProcArrayLock          |     3 |        71
30109 | postgres | pgbench  | IO                | XactSync            | active         |
LWLock    | wal_insert             |  1940 |     27741
30109 | postgres | pgbench  | IO                | XactSync            | active         |
LWLock    | buffer_content         | 22962 |    776352
30109 | postgres | pgbench  | IO                | XactSync            | active         |
LWLock    | lock_manager           |  9879 |    218826
30109 | postgres | pgbench  | IO                | XactSync            | active         |
Lock      | tuple                  | 20401 |  63581306
30109 | postgres | pgbench  | IO                | XactSync            | active         |
Lock      | transactionid          | 50769 | 211645008
30109 | postgres | pgbench  | IO                | XactSync            | active         |
Client    | ClientRead             | 89901 |  44192439
```

이 예제에서는 현재 세션(`pid <>pg_backend_pid()`)을 제외한 모든 활성 세션(`pg_stat_activity state <> 'idle'`)에 대한 현재 및 상위 3개의 누적 대기 유형과 대기 이벤트를 보여줍니다.

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
```

```
 WHERE RANK <=3;
  pid   | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+---------------
+-----------+-----------------+---------+------------+------
 20567 | postgres | psql     | CPU               | CPU                | active        |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | BufFileWrite    |   20750 |     184559 |    3
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93641 |  341472318 |    2
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417556 |  204796837 |    3
 .
 .
 .
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | transactionid   |  238068 | 1265816822 |    1
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | tuple           |   93210 |  338312247 |    2
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Client    | ClientRead      |  419157 |  207836533 |    3
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237621 | 1264528811 |    1
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93563 |  339799310 |    2
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

aurora_stat_bgwriter는 최적화된 읽기 캐시 쓰기에 대한 정보를 보여주는 통계 뷰입니다.

구문

```
aurora_stat_bgwriter()
```

인수

없음

반환 타입

모든 pg_stat_bgwriter 열과 다음과 같은 추가 열이 포함된 SETOF 레코드. pg_stat_bgwriter 열에 대한 자세한 내용은 pg_stat_bgwriter 섹션을 참조하세요.

pg_stat_reset_shared("bgwriter")을 사용하여 이 함수의 통계를 재설정할 수 있습니다.

- orcache_blks_written - 기록된 최적화된 읽기 캐시 데이터 블록의 총 수입니다.
- orcache_blk_write_time - track_io_timing이 활성화된 경우 최적화된 읽기 캐시에서 데이터를 쓰는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 자세한 내용은 track_io_timing을 참조하세요.

사용 노트

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.4 이상의 15 버전
- 14.9 이상의 14 버전

예시

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

pg_stat_database의 모든 열을 포함하고 끝에 새 열을 추가합니다.

구문

```
aurora_stat_database()
```

인수

없음

반환 타입

모든 pg_stat_database 열과 다음과 같은 추가 열이 포함된 SETOF 레코드. pg_stat_database 열에 대한 자세한 내용은 pg_stat_database 섹션을 참조하세요.

- storage_blks_read - 이 데이터베이스의 Aurora 스토리지에서 읽은 총 공유 블록 수입니다.
- orcache_blks_hit - 이 데이터베이스의 최적화된 읽기 캐시 적중률의 총 수입니다.
- local_blks_read - 이 데이터베이스에서 읽은 총 로컬 블록 수입니다.
- storage_blk_read_time - track_io_timing이 활성화된 경우 Aurora 스토리지에서 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.
- local_blk_read_time - track_io_timing이 활성화된 경우 로컬 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.
- orcache_blk_read_time - track_io_timing이 활성화된 경우 최적화된 읽기 캐시에서 데이터 파일 블록을 읽는 데 소요된 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.

> **Note**
>
> blks_read의 값은 storage_blks_read, orcache_blks_hit, local_blks_read의 합계입니다.
> blk_read_time의 값은 storage_blk_read_time, orcache_blk_read_time, local_blk_read_time의 합계입니다.

사용 노트

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.4 이상의 15 버전
- 14.9 이상의 14 버전

예시

다음 예는 모든 `pg_stat_database` 열을 포함하고 끝에 새 열 6개를 추가하는 방법을 보여줍니다.

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
numbackends              | 1
xact_commit              | 223
xact_rollback            | 4
blks_read                | 1059
blks_hit                 | 11456
tup_returned             | 27746
tup_fetched              | 5220
tup_inserted             | 165
tup_updated              | 42
tup_deleted              | 91
conflicts                | 0
temp_files               | 0
temp_bytes               | 0
deadlocks                | 0
checksum_failures        |
checksum_last_failure    |
blk_read_time            | 3358.689
blk_write_time           | 0
session_time             | 1076007.997
active_time              | 3684.371
idle_in_transaction_time | 0
sessions                 | 10
sessions_abandoned       | 0
sessions_fatal           | 0
sessions_killed          | 0
stats_reset              | 2023-01-12 20:15:17.370601+00
orcache_blks_hit         | 425
orcache_blk_read_time    | 89.934
storage_blks_read        | 623
storage_blk_read_time    | 3254.914
local_blks_read          | 0
local_blk_read_time      | 0
```

aurora_stat_dml_activity

Aurora PostgreSQL 클러스터의 데이터베이스에서 각 DML(데이터 조작 언어) 작업 유형에 대한 누적 활동을 보고합니다.

구문

```
aurora_stat_dml_activity(database_oid)
```

인수

database_oid

Aurora PostgreSQL 클러스터에 있는 데이터베이스의 객체 ID(OID)입니다.

반환 타입

SETOF 레코드

사용 노트

`aurora_stat_dml_activity` 함수는 PostgreSQL 엔진 11.6 이상과 호환되는 Aurora PostgreSQL 릴리스 3.1에서만 사용할 수 있습니다.

많은 수의 데이터베이스가 있는 Aurora PostgreSQL 클러스터에서 이 함수를 사용하여 DML 작업이 많거나 느린 데이터베이스 또는 둘 다 해당하는 데이터베이스를 식별합니다.

`aurora_stat_dml_activity` 함수가 실행된 횟수와 SELECT, INSERT, UPDATE 및 DELETE 작업의 누적 대기 시간을 마이크로초 단위로 반환합니다. 이 보고서에는 성공적인 DML 작업만 포함됩니다.

PostgreSQL 통계 액세스 함수 `pg_stat_reset`을 사용하여이 통계를 재설정할 수 있습니다. `pg_stat_get_db_stat_reset_time` 함수를 사용하면 이 통계가 마지막으로 재설정된 시간을 확인할 수 있습니다. PostgreSQL 통계 액세스 함수에 대한 자세한 내용은 PostgreSQL 설명서에서 [통계 수집기](#)를 참조하세요.

예시

다음 예는 연결된 데이터베이스에 대한 DML 작업 통계를 보고하는 방법을 보여줍니다.

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
```

```
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |             1 |                    53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+--------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

다음 예는 Aurora PostgreSQL 클러스터의 모든 데이터베이스에 대한 DML 작업 통계를 보여줍니다. 이 클러스터에는 `postgres`와 `mydb`라는 두 개의 데이터베이스가 있습니다. 쉼표로 구분된 목록은 `select_count`, `select_latency_microsecs`, `insert_count`, `insert_latency_microsecs`, `update_count`, `update_latency_microsecs`, `delete_count` 및 `delete_latency_microsecs` 필드와 일치합니다.

Aurora PostgreSQL은 `rdsadmin`이라는 시스템 데이터베이스를 만들고 이 데이터베이스를 사용하여 백업, 복원, 상태 확인, 복제 등의 관리 작업을 지원합니다. 이러한 DML 작업은 Aurora PostgreSQL 클러스터에 영향을 주지 않습니다.

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                    aurora_stat_dml_activity
-------+----------------
+-------------------------------------------------------------------
```

```
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

다음 예는 가독성을 높이기 위해 열로 구성된 모든 데이터베이스의 DML 작업 통계를 보여줍니다.

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
----------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
            |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
            | 0            | 0                          | 0            | 0
 template1      |              |                          |              |
            |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
            | 0            | 0                          | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
            | 0            | 0                          | 0            | 0
 postgres       | 42035        | 95179203                 | 5752         | 2678832898
            | 21157        | 441883182488               | 2            | 1520
 mydb           | 71           | 453514                   | 0            | 0
            | 1            | 190                        | 1            | 152
```

다음 예는 OID가 16401인 데이터베이스의 각 DML 작업에 대한 평균 누적 대기 시간(누적 대기 시간을 카운트로 나눈 값)을 보여줍니다.

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

Aurora PostgreSQL 데이터베이스의 누적 커밋 대기 시간을 마이크로초 단위로 가져옵니다. [커밋 대기 시간(Commit latency)]은 클라이언트가 커밋 요청을 제출하는 시점과 커밋 승인을 수신하는 시점 사이의 시간으로 측정됩니다.

구문

```
aurora_stat_get_db_commit_latency(database_oid)
```

인수

database_oid

Aurora PostgreSQL 데이터베이스의 객체 ID(OID)입니다.

반환 타입

SETOF 레코드

사용 노트

Amazon CloudWatch는 이 함수를 사용하여 평균 커밋 대기 시간을 계산합니다. Amazon CloudWatch 지표와 긴 커밋 대기 시간을 해결하는 방법에 대한 자세한 내용은 Amazon RDS 콘솔에서 지표 보기 및 Amazon CloudWatch 지표를 사용하여 Amazon RDS에 대한 더 나은 의사 결정 내리기를 참조하세요.

PostgreSQL 통계 액세스 함수 pg_stat_reset을 사용하여이 통계를 재설정할 수 있습니다. pg_stat_get_db_stat_reset_time 함수를 사용하면 이 통계가 마지막으로 재설정된 시간을 확인할 수 있습니다. PostgreSQL 통계 액세스 함수에 대한 자세한 내용은 PostgreSQL 설명서에서 통계 수집기를 참조하세요.

예시

다음 예에서는 pg_database 클러스터의 각 데이터베이스에 대한 누적 커밋 대기 시간을 가져옵니다.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |     datname     | aurora_stat_get_db_commit_latency
-------+-----------------+----------------------------------
 14006 | template0       |                                0
 16384 | rdsadmin        |                        654387789
     1 | template1       |                                0
 16401 | mydb            |                           229556
 69768 | postgres        |                            22011
```

다음 예에서는 현재 연결된 데이터베이스에 대한 누적 커밋 대기 시간을 가져옵니다. 이 예에서는 aurora_stat_get_db_commit_latency 함수를 호출하기 전에 먼저 \gset을 사용하여 oid 인수의 변수를 정의하고 연결된 데이터베이스에서 해당 값을 설정합니다.

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

다음 예에서는 pg_database 클러스터의 mydb 데이터베이스에 대한 누적 커밋 대기 시간을 가져옵니다. 그런 다음, `pg_stat_reset` 함수를 사용하여 이 통계를 재설정하고 결과를 표시합니다. 마지막으로 `pg_stat_get_db_stat_reset_time` 함수를 사용하여 이 통계가 마지막으로 재설정된 시간을 확인합니다.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname   | aurora_stat_get_db_commit_latency
-------+------------+-----------------------------------
 16427 | mydb       |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------



=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
  oid  |  datname   | aurora_stat_get_db_commit_latency
-------+------------+-----------------------------------
 16427 | mydb       |                                 6
```

```
=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

슬롯당 미리 쓰기 로그(WAL) 캐시 사용량을 보여 줍니다.

구문

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- `name` - 복제 슬롯의 이름입니다.
- `active_pid` - walsender 프로세스의 ID입니다.
- `cache_hit` - 마지막 재설정 이후 wal 캐시 총 적중 횟수입니다.
- `cache_miss` - 마지막 재설정 이후 wal 캐시 총 미적중 횟수입니다.
- `blks_read`— wal 캐시 읽기 요청 총합입니다.
- `hit_rate` – WAL 캐시 적중률(cache_hit / blks_read)입니다.
- `last_reset_timestamp` - 마지막으로 카운터를 재설정한 시간입니다.

사용 노트

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.2 이상의 모든 버전
- 14.7 이상 버전

- 13.8 이상 버전
- 12.12 이상 버전
- 11.17 이상 버전

예시

다음 예제에서는 활성 `aurora_stat_logical_wal_cache` 함수가 하나뿐인 복제 슬롯 2개를 확인할 수 있습니다.

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

aurora_stat_memctx_usage

각 PostgreSQL 프로세스의 메모리 컨텍스트 사용량을 보고합니다.

구문

```
aurora_stat_memctx_usage()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- `pid` - 프로세스의 ID입니다.
- `name` - 메모리 컨텍스트의 이름입니다.
- `allocated` - 메모리 컨텍스트가 기본 메모리 하위 시스템에서 가져온 바이트 수입니다.

- `used` - 메모리 컨텍스트의 클라이언트에 커밋된 바이트 수입니다.
- `instances` - 이 유형의 현재 존재하는 컨텍스트 수입니다.

사용 노트

이 함수는 PostgreSQL 프로세스의 메모리 컨텍스트 사용량을 표시합니다. 일부 프로세스에는 anonymous라는 레이블이 지정되어 있습니다. 프로세스는 제한된 키워드를 포함하므로 노출되지 않습니다.

이 함수는 다음 Aurora PostgreSQL 버전 이상에서 사용할 수 있습니다.

- 15.3 이상의 15 버전
- 14.8 이상의 14 버전
- 13.11 이상의 13 버전
- 12.15 이상의 12 버전
- 11.20 이상의 11 버전

예시

다음 예제에서는 `aurora_stat_memctx_usage` 함수 호출의 결과를 보여줍니다.

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used  | instances
-------+---------------------------------+-----------+---------+-----------
123864 | Miscellaneous                   |     19520 |   15064 |         3
123864 | Aurora File Context             |      8192 |     616 |         1
123864 | Aurora WAL Context              |      8192 |     296 |         1
123864 | CacheMemoryContext              |    524288 |  422600 |         1
123864 | Catalog tuple context           |     16384 |   13736 |         1
123864 | ExecutorState                   |     32832 |   28304 |         1
123864 | ExprContext                     |      8192 |    1720 |         1
123864 | GWAL record construction        |      1024 |     832 |         1
123864 | MdSmgr                          |      8192 |     296 |         1
123864 | MessageContext                  |    532480 |  353832 |         1
123864 | PortalHeapMemory                |      1024 |     488 |         1
123864 | PortalMemory                    |      8192 |     576 |         1
123864 | printtup                        |      8192 |     296 |         1
123864 | RelCache hash table entries     |      8192 |    8152 |         1
```

```
123864 | RowDescriptionContext            |      8192 |    1344 |          1
123864 | smgr relation context            |      8192 |     296 |          1
123864 | Table function arguments         |      8192 |     352 |          1
123864 | TopTransactionContext            |      8192 |     632 |          1
123864 | TransactionAbortContext          |     32768 |     296 |          1
123864 | WAL record construction          |     50216 |   43904 |          1
123864 | hash table                       |     65536 |   52744 |          6
123864 | Relation metadata                |    191488 |  124240 |         87
104992 | Miscellaneous                    |      9280 |    7728 |          3
104992 | Aurora File Context              |      8192 |     376 |          1
104992 | Aurora WAL Context               |      8192 |     296 |          1
104992 ||Autovacuum Launcher              |      8192 |     296 |          1
104992 | Autovacuum database list         |     16384 |     744 |          2
104992 | CacheMemoryContext               |    262144 |  140288 |          1
104992 | Catalog tuple context            |      8192 |     296 |          1
104992 | GWAL record construction         |      1024 |     832 |          1
104992 | MdSmgr                           |      8192 |     296 |          1
104992 | PortalMemory                     |      8192 |     296 |          1
104992 | RelCache hash table entries      |      8192 |     296 |          1
104992 | smgr relation context            |      8192 |     296 |          1
104992 | Autovacuum start worker (tmp)    |      8192 |     296 |          1
104992 | TopTransactionContext            |     16384 |     592 |          2
104992 | TransactionAbortContext          |     32768 |     296 |          1
104992 | WAL record construction          |     50216 |   43904 |          1
104992 | hash table                       |     49152 |   34024 |          4
(39 rows)
```

일부 제한된 키워드는 숨겨지고 출력은 다음과 같이 표시됩니다.

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
  5482 | anonymous                        |      8192 |     456 |          1
  5482 | anonymous                        |      8192 |     296 |          1
```

aurora_stat_optimized_reads_cache

이 함수는 계층형 캐시 통계를 보여줍니다.

구문

```
aurora_stat_optimized_reads_cache()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- `total_size` - 최적화된 총 읽기 캐시 크기입니다.
- `used_size` - 최적화된 읽기 캐시에서 사용된 페이지 크기입니다.

사용 노트

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.4 이상의 15 버전
- 14.9 이상의 14 버전

예시

다음 예는 r6gd.8xlarge 인스턴스의 출력을 보여줍니다.

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

추적된 모든 실행 계획에 대해 하나의 행을 반환합니다.

구문

```
aurora_stat_plans(
    showtext
```

```
)
```

인수

- showtext - 쿼리와 계획 텍스트를 표시합니다. 유효한 값은 NULL, true 또는 false입니다. True인 경우 쿼리와 계획 텍스트를 표시합니다.

반환 타입

`aurora_stat_statements`의 모든 열과 다음의 추가 열을 포함하는 추적된 각 계획의 행 하나를 반환합니다.

- planid – 계획 식별자
- explain_plan — 계획 텍스트 설명
- plan_type:
  - `no plan` - 계획이 캡처되지 않음
  - `estimate` - 예상 비용을 포함하여 캡처된 계획
  - `actual` - EXPLAIN ANALYZE로 캡처된 계획
- plan_captured_time - 계획이 마지막으로 캡처된 시간

사용 노트

계획을 추적하려면 `aurora_compute_plan_id`가 활성화되어 있어야 하고 `pg_stat_statements`가 `shared_preload_libraries`에 있어야 합니다.

사용 가능한 계획 수는 `pg_stat_statements.max` 파라미터에 설정된 값에 따라 제어됩니다. `aurora_compute_plan_id`가 활성화되면 `aurora_stat_plans`에서 지정된 값까지 계획을 추적할 수 있습니다.

이 함수는 Aurora PostgreSQL 버전 14.10 및 15.5 이상 모든 버전의 릴리스부터 사용할 수 있습니다.

예시

아래 예제에서는 쿼리 식별자 -5471422286312252535에 대한 두 개의 계획이 캡처되고 해당 문 통계는 planid에 의해 추적됩니다.

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
```

```
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                 explain_plan
---------+--------------------+-------------+-------------------------------
+------------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                                      +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                                      +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                                +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                                      +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

논리적 wal 캐시의 카운터를 재설정합니다.

구문

특정 슬롯을 재설정하는 방법

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

모든 슬롯을 재설정하는 방법

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

인수

NULL 또는 `slot_name`

반환 타입

상태 메시지, 텍스트 문자열

- 논리적 Wal 캐시 카운터 재설정 - 성공 메시지입니다. 함수가 성공하면 이 텍스트가 반환됩니다.
- 복제 슬롯을 찾을 수 없습니다. 다시 시도하세요. - 오류 메시지입니다. 함수가 실패하면 이 텍스트가 반환됩니다.

사용 노트

이 함수는 다음 버전에서 사용할 수 있습니다.

- Aurora PostgreSQL 14.5 이상
- Aurora PostgreSQL 버전 13.8 이상
- Aurora PostgreSQL 버전 12.12 이상
- Aurora PostgreSQL 버전 11.17 이상

예시

다음 예제에서는 `aurora_stat_reset_wal_cache` 함수를 사용하여 `test_results`라는 슬롯을 재설정한 다음, 존재하지 않는 슬롯의 재설정을 시도합니다.

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
--------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
-----------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_resource_usage

모든 Aurora PostgreSQL 백엔드 프로세스에 대한 백엔드 리소스 지표 및 cpu 사용량으로 구성된 실시간 리소스 사용률을 보고합니다.

구문

```
aurora_stat_resource_usage()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- pid - 프로세스 식별자
- allocated_memory - 프로세스별로 할당된 총 메모리(바이트)
- used_memory - 프로세스별로 실제로 사용된 메모리(바이트)
- cpu_usage_percent - 프로세스의 CPU 사용률(%)

사용 노트

이 함수는 각 Aurora PostgreSQL 백엔드 프로세스의 백엔드 리소스 사용량을 표시합니다.

이 함수는 다음 Aurora PostgreSQL 버전 이상에서 사용할 수 있습니다.

- Aurora PostgreSQL 17.5 이상 17 버전
- Aurora PostgreSQL 16.9 이상 16 버전
- Aurora PostgreSQL 15.13 이상 15 버전
- Aurora PostgreSQL 14.18 이상 14 버전
- Aurora PostgreSQL 13.21 이상 13 버전

예시

다음 예에서는 `aurora_stat_resource_usage` 함수의 출력을 보여줍니다.

```
=> select * from aurora_stat_resource_usage();
 pid  | allocated_memory | used_memory |    cpu_usage_percent
------+------------------+-------------+-----------------------
```

```
   666 |          1074032 |        333544 |   0.00729274882897963
   667 |           787312 |        287360 | 0.0029263928146372746
   668 |          3076776 |       1563488 |  0.006013116835953961
   684 |           803744 |        307480 |  0.002226855426881142
  2401 |          1232992 |        943144 |                     0
   647 |             8000 |           944 |   0.48853387812429855
   659 |           319344 |        243000 | 0.0004135602076683591
   663 |           262000 |        185736 |  0.008181301476644002
   664 |             9024 |          1216 |   0.10992313082653653
 (9 rows)
```

aurora_stat_statements

모든 pg_stat_statements 열을 표시하고 끝에 열을 더 추가합니다.

구문

```
aurora_stat_statements(showtext boolean)
```

인수

showtext 부울

반환 타입

모든 pg_stat_statements 열과 다음과 같은 추가 열이 포함된 SETOF 레코드. pg_stat_statements 열에 대한 자세한 내용은 pg_stat_statements 섹션을 참조하세요.

pg_stat_statements_reset()을 사용하여 이 함수의 통계를 재설정할 수 있습니다.

- storage_blks_read - 이 문이 Aurora 스토리지에서 읽은 총 공유 블록 수입니다.
- orcache_blks_hit - 이 문에 의한 최적화된 읽기 캐시 적중률의 총 수입니다.
- storage_blk_read_time - track_io_timing이 활성화된 경우 Aurora 스토리지에서 공유된 블록을 읽는 데 문이 소요한 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.
- local_blk_read_time - track_io_timing이 활성화된 경우 로컬 블록을 읽는 데 문이 소요한 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.

- orcache_blk_read_time - track_io_timing이 활성화된 경우 최적화된 읽기 캐시에서 공유된 블록을 읽는 데 문이 소요한 총 시간을 밀리초 단위로 추적합니다. 활성화되지 않은 경우 값은 0입니다. 자세한 내용은 track_io_timing을 참조하세요.
- total_plan_peakmem – 이 문에 대한 모든 직접 호출의 계획 단계 중 피크 메모리 값의 총계입니다. 문을 계획하는 동안 평균 피크 메모리를 확인하려면 이 값을 직접 호출 수로 나누세요.
- min_plan_peakmem – 이 문에 대한 모든 직접 호출을 계획하는 동안 나타난 최소 피크 메모리 값입니다.
- max_plan_peakmem – 이 문에 대한 모든 직접 호출을 계획하는 동안 나타난 최대 피크 메모리 값입니다.
- total_exec_peakmem – 이 문에 대한 모든 직접 호출의 실행 단계 중 피크 메모리 값의 총계입니다. 문을 실행하는 동안 평균 피크 메모리를 확인하려면 이 값을 직접 호출 수로 나누세요.
- min_exec_peakmem – 이 문에 대한 모든 직접 호출을 실행하는 동안 나타난 최소 피크 메모리 값입니다.
- max_exec_peakmem – 이 문에 대한 모든 직접 호출을 실행하는 동안 나타난 최대 피크 메모리 값입니다.

> **Note**
>
> total_plan_peakmen, min_plan_peakmem, max_plan_peakmem은 pg_stat_statements.track_planning 설정이 켜져 있을 때만 모니터링됩니다.

사용 노트

aurora_stat_statements() 함수를 사용하려면 shared_preload_libraries 파라미터에 pg_stat_statements 확장을 포함해야 합니다.

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.4 이상의 15 버전
- 14.9 이상의 14 버전

피크 메모리를 표시하는 열은 다음 버전에서 사용할 수 있습니다.

- 16.3 이상 버전
- 15.7 이상 버전

- 14.12 이상 버전

예시

다음 예는 모든 pg_stat_statements 열을 포함하고 끝에 새 열 11개를 추가하는 방법을 보여줍니다.

```
=> select * from aurora_stat_statements(true) where query like 'with window_max%';
-[ RECORD 1 ]----------
+------------------------------------------------------------------------------------------
userid                  | 16409
dbid                    | 5
toplevel                | t
queryid                 | -8347523682669847482
query                   | with window_max as (select custid, max(scratch) over (order by
 scratch rows between $1 preceding
and $2 following) wmax from ts) select sum(wmax), max(custid) from window_max
plans                   | 0
total_plan_time         | 0
min_plan_time           | 0
max_plan_time           | 0
mean_plan_time          | 0
stddev_plan_time        | 0
calls                   | 4
total_exec_time         | 254.105121
min_exec_time           | 57.503164000000005
max_exec_time           | 68.687418
mean_exec_time          | 63.52628025
stddev_exec_time        | 5.150765359979643
rows                    | 4
shared_blks_hit         | 200192
shared_blks_read        | 0
shared_blks_dirtied     | 0
shared_blks_written     | 0
local_blks_hit          | 0
local_blks_read         | 0
local_blks_dirtied      | 0
local_blks_written      | 0
temp_blks_read          | 0
temp_blks_written       | 0
blk_read_time           | 0
blk_write_time          | 0
temp_blk_read_time      | 0
temp_blk_write_time     | 0
```

```
wal_records            | 0
wal_fpi                | 0
wal_bytes              | 0
jit_functions          | 0
jit_generation_time    | 0
jit_inlining_count     | 0
jit_inlining_time      | 0
jit_optimization_count | 0
jit_optimization_time  | 0
jit_emission_count     | 0
jit_emission_time      | 0
storage_blks_read      | 0
orcache_blks_hit       | 0
storage_blk_read_time  | 0
local_blk_read_time    | 0
orcache_blk_read_time  | 0
total_plan_peakmem     | 0
min_plan_peakmem       | 0
max_plan_peakmem       | 0
total_exec_peakmem     | 6356224
min_exec_peakmem       | 1589056
max_exec_peakmem       | 1589056
```

aurora_stat_system_waits

Aurora PostgreSQL DB 인스턴스에 대한 대기 이벤트 정보를 보고합니다.

구문

```
aurora_stat_system_waits()
```

인수

없음

반환 타입

SETOF 레코드

사용 노트

이 함수는 현재 연결된 DB 인스턴스에 의해 생성된 각 대기 이벤트의 누적 대기 횟수와 누적 대기 시간을 반환합니다.

반환되는 레코드 세트에는 다음 필드가 포함됩니다.

- `type_id` - 대기 이벤트 유형의 ID입니다.
- `event_id` - 대기 이벤트의 ID입니다.
- `waits` - 대기 이벤트가 발생한 횟수입니다.
- `wait_time` - 이 이벤트를 기다리는 데 소요된 총 시간(마이크로초)입니다.

이 함수에서 반환되는 통계는 DB 인스턴스가 다시 시작될 때 재설정됩니다.

예시

다음 예제에서는 `aurora_stat_system_waits` 함수 호출의 결과를 보여줍니다.

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+--------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

다음 예는 이 함수를 `aurora_stat_wait_event` 및 `aurora_stat_wait_type`과 함께 사용하여 가독성이 더 높은 결과를 생성하는 방법을 보여줍니다.

```
=> SELECT type_name,
             event_name,
             waits,
             wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();
```

```
 type_name |        event_name        | waits   | wait_time
-----------+-------------------------+----------+--------------
 LWLock    | XidGenLock              |       11 |        12864
 LWLock    | ProcArrayLock           |      501 |       174473
 LWLock    | buffer_content          |    53171 |     23641847
 LWLock    | rdsutils                |        2 |        12764
 Lock      | tuple                   |    75686 |   2033956052
 Lock      | transactionid           |  1765147 |  47267583409
 Activity  | AutoVacuumMain          |   136868 |  56305604538
 Activity  | BgWriterHibernate       |     7486 |  55266949471
 Activity  | BgWriterMain            |     7487 |   1508909964
.
.
.
 IO        | SLRURead                |        3 |        11756
 IO        | WALWrite                | 52544463 |    388850428
 IO        | XactSync                |   187073 |    597041642
 IO        | ClogRead                |        2 |        47729
 IO        | OutboundCtrlRead        |        1 |          888
 IO        | OutboundCtrlWrite       |        2 |           64
```

aurora_stat_wait_event

Aurora PostgreSQL에 대해 지원되는 대기 이벤트를 모두 나열합니다. Aurora PostgreSQL 대기 이벤트에 대한 자세한 내용은 Amazon Aurora PostgreSQL 대기 이벤트 섹션을 참조하세요.

구문

```
aurora_stat_wait_event()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- type_id - 대기 이벤트 유형의 ID입니다.
- - 대기 이벤트의 ID입니다.
- type_name - 대기 유형 이름

- event_name - 대기 이벤트 이름

사용 노트

ID 대신 이벤트 유형을 사용한 이벤트 이름을 보려면 이 함수를 `aurora_stat_wait_type` 및 `aurora_stat_system_waits`와 같은 다른 함수와 함께 사용합니다. 이 함수에서 반환되는 대기 이벤트 이름은 `aurora_wait_report` 함수에서 반환되는 이름과 동일합니다.

예시

다음 예제에서는 `aurora_stat_wait_event` 함수 호출의 결과를 보여줍니다.

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |                event_name
---------+-----------+------------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

다음 예에서는 `aurora_stat_wait_type`과 `aurora_stat_wait_event`를 함께 사용하여 유형 이름과 이벤트 이름을 반환함으로써 가독성을 높입니다.

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |                  event_name
---------+-----------+---------+-----------+------------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
       1 | LWLock    |       1 |  16777218 | OidGenLock
       1 | LWLock    |       1 |  16777219 | XidGenLock
       1 | LWLock    |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock      |       3 |  50331648 | relation
       3 | Lock      |       3 |  50331649 | extend
       3 | Lock      |       3 |  50331650 | page
       3 | Lock      |       3 |  50331651 | tuple
.
.
.
      10 | IO        |      10 | 167772214 | TimelineHistorySync
      10 | IO        |      10 | 167772215 | TimelineHistoryWrite
      10 | IO        |      10 | 167772216 | TwophaseFileRead
      10 | IO        |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN       |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

Aurora PostgreSQL에 대해 지원되는 대기 유형을 모두 나열합니다.

구문

```
aurora_stat_wait_type()
```

인수

없음

반환 타입

다음 열이 있는 SETOF 레코드:

- type_id - 대기 이벤트 유형의 ID입니다.
- type_name - 대기 유형 이름

사용 노트

ID 대신 대기 이벤트 유형을 사용한 대기 이벤트 이름을 보려면 이 함수를 aurora_stat_wait_event 및 aurora_stat_system_waits와 같은 다른 함수와 함께 사용합니다. 이 함수에서 반환되는 대기 유형 이름은 aurora_wait_report 함수에서 반환되는 이름과 동일합니다.

예시

다음 예제에서는 aurora_stat_wait_type 함수 호출의 결과를 보여줍니다.

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

aurora_version

Amazon Aurora PostgreSQL 호환 버전 번호의 문자열 값을 반환합니다.

구문

```
aurora_version()
```

인수

없음

반환 타입

CHAR 또는 VARCHAR 문자열

사용 노트

이 함수는 Amazon Aurora PostgreSQL 호환 버전 데이터베이스 엔진의 버전을 표시합니다. 버전 번호는 *major*.*minor*.*patch* 형식의 문자열로 반환됩니다. Aurora PostgreSQL 버전 번호에 대한 자세한 내용은 Aurora 버전 번호 단원을 참조하세요.

Aurora PostgreSQL DB 클러스터의 유지 관리 기간을 설정하여 마이너 버전 업그레이드를 적용할 시기를 선택할 수 있습니다. 자세한 방법은 Amazon Aurora DB 클러스터 유지 관리을 참조하세요.

Aurora PostgreSQL 버전 13.3, 12.8, 11.13, 10.18 및 다른 모든 이후 버전의 릴리스부터 Aurora 버전 번호는 PostgreSQL 버전 번호를 따릅니다. 모든 Aurora PostgreSQL 릴리스에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 Amazon Aurora PostgreSQL 업데이트를 참조하세요.

예시

다음 예제에서는 PostgreSQL 12.7, Aurora PostgreSQL 릴리스 4.2를 실행 중인 Aurora PostgreSQL DB 클러스터에서 aurora_version 함수를 호출한 후 Aurora PostgreSQL 버전 13.3을 실행 중인 클러스터에서 같은 함수를 실행한 결과를 보여줍니다.

```
=> SELECT * FROM aurora_version();
aurora_version
-----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
-----------------
 13.3.0
```

이 예제에서는 다양한 옵션과 함께 함수를 사용하여 Aurora PostgreSQL 버전에 대한 자세한 정보를 얻는 방법을 보여줍니다. 이 예제에는 PostgreSQL 버전 번호와 구별되는 Aurora 버전 번호가 있습니다.

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)

=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
---------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
------------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

다음 예제에서는 이전 예제에서와 동일한 옵션을 가진 함수를 사용합니다. 이 예제에는 PostgreSQL 버전 번호와 구별되는 Aurora 버전 번호가 없습니다.

```
=> SHOW SERVER_VERSION;
server_version
----------------
 13.3
```

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
---------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-------------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

Aurora 클러스터 볼륨의 논리적 미리 쓰기 로그(WAL) 스트림에서 레코드의 시작을 식별하는 데 사용되는 로그 시퀀스 번호(LSN)를 반환합니다.

구문

```
aurora_volume_logical_start_lsn()
```

인수

없음

반환 타입

`pg_lsn`

사용 노트

이 함수는 주어진 Aurora 클러스터 볼륨의 논리적 WAL 스트림에서 레코드의 시작 부분을 식별합니다. 논리적 복제와 Aurora 고속 복제를 사용하여 메이저 버전 업그레이드를 수행하는 동안 이 함수를 사용하여 스냅샷 또는 데이터베이스 클론이 생성된 LSN을 확인할 수 있습니다. 그런 다음 논리적 복제를 사용하여 LSN 이후에 기록된 최신 데이터를 지속적으로 스트리밍하고 게시자에서 구독자로 변경 내용을 동기화할 수 있습니다.

메이저 버전 업그레이드에 논리적 복제를 사용하는 방법에 대한 자세한 내용은 논리적 복제를 사용하여 Aurora PostgreSQL에 대한 메이저 버전 업그레이드 수행 섹션을 참조하세요.

이 함수는 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- 15.2 이상의 15 버전
- 14.3 이상의 14 버전
- 13.6 이상의 13 버전
- 12.10 이상의 12 버전
- 11.15 이상의 11 버전
- 10.20 이상의 10 버전

예시

다음 쿼리를 사용하여 로그 시퀀스 번호(LSN)를 얻을 수 있습니다.

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

이 함수는 일정 기간 동안 대기 이벤트 활동을 표시합니다.

구문

```
aurora_wait_report([time])
```

인수

시간(선택 사항)

시간(초)입니다. 기본값은 10초입니다.

반환 타입

다음 열이 있는 SETOF 레코드:

- type_name - 대기 유형 이름
- event_name - 대기 이벤트 이름
- wait - 대기 횟수
- wait_time - 대기 시간(밀리초)
- ms_per_wait - 대기 수당 한 평균 밀리초
- waits_per_xact - 트랜잭션 수 하나당 평균 대기
- ms_per_wait - 트랜잭션 수당 평균 밀리초

사용 노트

이 함수는 PostgreSQL 9.6.6 이상 버전과 호환되는 Aurora PostgreSQL 릴리스 1.1부터 사용할 수 있습니다.

이 함수를 사용하려면 먼저 Aurora PostgreSQL`aurora_stat_utils` 확장을 다음과 같이 생성해야 합니다.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

사용할 수 있는 Aurora PostgreSQL 확장 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 [Amazon Aurora PostgreSQL 확장 버전]을 참조하세요.

이 함수는 aurora_stat_system_waits () 함수와 pg_stat_database PostgreSQL 통계 뷰에서 얻은 통계 데이터의 두 스냅샷을 비교하여 인스턴스 수준 대기 이벤트를 계산합니다.

`aurora_stat_system_waits()` 및 `pg_stat_database`에 대한 자세한 내용은 PostgreSQL 설명서에서 [통계 수집기]를 참조하세요.

실행 시 이 함수는 초기 스냅샷을 찍고 지정된 시간(초)을 기다린 다음 두 번째 스냅샷을 찍습니다. 이 함수는 두 스냅샷을 비교하고 차이를 반환합니다. 이 차이는 해당 시간 간격의 인스턴스 활동을 나타냅니다.

라이터 인스턴스에서 함수는 커밋된 트랜잭션 수와 TPS(초당 트랜잭션 수)도 표시합니다. 이 함수는 인스턴스 수준에서 정보를 반환하며 인스턴스의 모든 데이터베이스를 포함합니다.

예시

이 예제에서는 aurora_wait_report 함수를 사용할 수 있도록 aurora_stat_utils 확장을 만드는 방법을 보여줍니다.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

이 예에서는 10초에 대한 대기 보고서를 확인하는 방법을 보여 줍니다.

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
```

```
 IO         | DataFileExtend     |      1 |       0.02 |       0.018 |            0.03 |
   0.001
```

이 예에서는 60초에 대한 대기 보고서를 확인하는 방법을 보여 줍니다.

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits   | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |       30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
```

```
IO         | RelationMapRead        |      72 |      5.16 |       0.072 |
0.05 |       0.003
LWLock     | ProcArrayLock          |       1 |      2.01 |       2.008 |
0.00 |       0.001
IO         | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
0.00 |       0.000
IO         | DataFileExtend         |       1 |      0.02 |       0.018 |
0.00 |       0.000
IO         | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Amazon Aurora PostgreSQL parameters

Amazon Aurora DB 클러스터는 다른 Amazon RDS DB 인스턴스와 마찬가지로 DB 파라미터 그룹의 파라미터를 사용하여 관리합니다. 그러나 Amazon Aurora는 Aurora DB 클러스터에 여러 DB 인스턴스가 있다는 점에서 Amazon RDS와 다릅니다. 다음과 같이 Amazon Aurora DB 클러스터를 관리하는 데 사용하는 일부 파라미터는 엔진 클러스터에 적용되는 반면, 또 다른 일부 파라미터는 DB 클러스터의 지정된 DB 인스턴스에만 적용됩니다.

- DB 클러스터 파라미터 그룹 - DB 클러스터 파라미터 그룹에는 Aurora DB 클러스터 전체에 적용되는 엔진 구성 파라미터 세트가 포함됩니다. 예를 들어 클러스터 캐시 관리는 DB 클러스터 파라미터 그룹의 일부인 `apg_ccm_enabled` 파라미터로 제어되는 Aurora DB 클러스터의 기능입니다. DB 클러스터 파라미터 그룹에는 클러스터를 구성하는 DB 인스턴스에 대한 DB 파라미터 그룹의 기본 설정도 포함되어 있습니다.
- DB 파라미터 그룹 - DB 파라미터 그룹은 해당 엔진 유형의 특정 DB 인스턴스에 적용되는 엔진 구성 값의 집합입니다. PostgreSQL DB 엔진용 DB 파라미터 그룹은 RDS for PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터에서 사용됩니다. 이 구성 설정은 메모리 버퍼 크기와 같은 Aurora 클러스터 내의 DB 인스턴스 사이에서 변화할 수 있는 속성에 적용됩니다.

클러스터 수준의 파라미터는 DB 클러스터 파라미터 그룹에서 관리됩니다. 인스턴스 수준의 파라미터는 DB 파라미터 그룹에서 관리됩니다. Amazon RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 파라미터를 관리할 수 있습니다. 클러스터 수준 파라미터와 인스턴스 수준 파라미터를 관리하기 위한 명령은 서로 다릅니다.

- DB 클러스터 파라미터 그룹에서 클러스터 수준 파라미터를 관리하려면 [modify-db-cluster-parameter-group](#) AWS CLI 명령을 사용합니다.
- DB 클러스터의 DB 인스턴스에 대한 DB 파라미터 그룹에서 인스턴스 수준 파라미터를 관리하려면 [modify-db-parameter-group](#) AWS CLI 명령을 사용합니다.

AWS CLI에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 사용을 참조하세요.

파라미터 그룹에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 단원을 참조하세요.

Aurora PostgreSQL DB 클러스터 및 DB 파라미터 보기

AWS Management Console에서 RDS for PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터에 대해 사용 가능한 모든 기본 파라미터 그룹을 볼 수 있습니다. 모든 DB 엔진과 DB 클러스터 유형 및 버전에 대한 기본 파라미터 그룹이 각 AWS 리전에 대해 나열됩니다. 모든 사용자 정의 파라미터 그룹도 나열됩니다.

AWS Management Console에서 보는 대신 AWS CLI 또는 Amazon RDS API를 사용하여 DB 클러스터 파라미터 그룹 및 DB 파라미터 그룹에 포함된 파라미터를 나열할 수도 있습니다. 예를 들어 DB 클러스터 파라미터 그룹의 파라미터를 나열하려면 다음과 같이 describe-db-cluster-parameters AWS CLI 명령을 사용합니다.

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

이 명령은 각 파라미터에 대한 자세한 JSON 설명을 반환합니다. 반환되는 정보의 양을 줄이기 위해 `--query` 옵션을 사용하여 원하는 사항을 지정할 수 있습니다. 예를 들어 다음과 같이 기본 Aurora PostgreSQL 12 DB 클러스터 파라미터 그룹에 대해 파라미터 이름, 설명 및 허용되는 값을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Windows의 경우:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Aurora DB 클러스터 파라미터 그룹에는 DB 인스턴스 파라미터 그룹과 지정된 Aurora DB 엔진의 기본 값이 포함됩니다. 다음과 같이 describe-db-parameters AWS CLI 명령을 사용하여 동일한 기본 Aurora PostgreSQL 기본 파라미터 그룹에서 DB 파라미터 목록을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Windows의 경우:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

앞의 명령은 쿼리에 지정된 설명 및 기타 세부 정보와 함께 DB 클러스터 또는 DB 파라미터 그룹의 파라미터 목록을 반환합니다. 다음은 응답의 예입니다.

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

다음은 Aurora PostgreSQL 버전 14의 기본 DB 클러스터 파라미터 및 DB 파라미터 값이 들어 있는 표입니다.

Aurora PostgreSQL 클러스터 수준 파라미터

AWS 관리 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 특정 Aurora PostgreSQL 버전에서 사용 가능한 클러스터 수준 파라미터를 볼 수 있습니다. RDS 콘솔의 Aurora PostgreSQL DB 클러스터 파라미터 그룹에서 파라미터 보기에 대한 자세한 정보는 [Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 값 보기](#) 섹션을 참조하세요.

일부 클러스터 수준 파라미터는 일부 버전에서 사용할 수 없으며 일부는 더 이상 사용되지 않습니다. 특정 Aurora PostgreSQL 버전의 파라미터 보기에 대한 자세한 내용은 [Aurora PostgreSQL DB 클러스터 및 DB 파라미터 보기](#) 섹션을 참조하세요.

예를 들어, 다음 테이블에는 Aurora PostgreSQL 버전 14의 기본 DB 클러스터 파라미터 그룹에서 사용할 수 있는 파라미터가 나와 있습니다. 사용자 정의 DB 파라미터 그룹을 지정하지 않고 Aurora PostgreSQL DB 클러스터를 생성하는 경우 `default.aurora-postgresql14`, `default.aurora-postgresql13` 등과 같이 선택한 버전에 대한 기본 Aurora DB 클러스터 파라미터 그룹을 사용하여 DB 클러스터가 생성됩니다.

동일한 기본 DB 클럿스터 그룹에 대한 DB 인스턴스 파라미터 목록에 내한 내용은 [Aurora PostgreSQL 인스턴스 수준 파라미터](#) 섹션을 참조하세요.

파라미터 이름	설명	Default
ansi_constraint_trigger_ordering	ANSI SQL 표준과 호환되도록 제약 조건 트리거의 실행 순서를 변경합니다.	–
ansi_force_foreign_key_checks	계단식 삭제 또는 계단식 업데이트와 같은 참조 작업이 작업에 대해 존재하는 다양한 트리거 컨텍스트에 관계없이 항상 발생하는지 확인합니다.	–
ansi_qualified_update_set_target	UPDATE ... SET 문에서 테이블 및 스키마 한정자를 지원합니다.	–
apg_ccm_enabled	클러스터에 대한 클러스터 캐시 관리를 사용 설정하거나 사용 중지합니다.	–
apg_enable_batch_mode_function_execution	배치 모드 기능을 사용하여 한 번에 일련의 행을 처리합니다.	–

파라미터 이름	설명	Default
apg_enable_correlated_any_transform	가능한 경우 플래너가 상관 관계가 있는 ANY 하위 링크(IN/NOT IN 하위 쿼리)를 JOIN으로 변환할 수 있습니다.	–
apg_enable_function_migration	플래너가 적합한 스칼라 함수를 FROM 절로 마이그레이션할 수 있습니다.	–
apg_enable_not_in_transform	플래너가 가능한 경우 NOT IN 하위 쿼리를 ANTI JOIN으로 변환할 수 있습니다.	–
apg_enable_remove_redundant_inner_joins	플래너가 중복 내부 조인을 제거할 수 있습니다.	–
apg_enable_semijoin_push_down	해시 조인에 세미조인 필터를 사용할 수 있습니다.	–
apg_plan_mgmt.capture_plan_baselines	캡처 계획 기준 모드입니다. 수동(manual) - 모든 SQL 문에 대해 계획 캡처 사용 설정, 끄기(off) - 계획 캡처 사용 중지, 자동(automatic) - 자격 기준을 충족하는 pg_stat_statements의 문에 계획 캡처 사용.	끄기
apg_plan_mgmt.max_databases	apg_plan_mgmt를 사용하여 쿼리를 관리할 수 있는 최대 데이터베이스 수를 설정합니다.	10
apg_plan_mgmt.max_plans	apg_plan_mgmt로 캐시할 수 있는 최대 계획 수를 설정합니다.	10000
apg_plan_mgmt.plan_retention_period	계획이 자동으로 삭제되기 전에 계획의 last_used 이후 최대 일수입니다.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	미승인 계획이 실행될 예상 총 계획 비용입니다.	0
apg_plan_mgmt.use_plan_baselines	관리형 문에 대해 승인된 계획이나 고정된 계획만 사용합니다.	false

파라미터 이름	설명	Default
application_name	애플리케이션 이름이 통계 및 로그에 표시되도록 설정합니다.	–
array_nulls	배열에서 NULL 요소의 입력을 사용 설정합니다.	–
aurora_compute_plan_id	쿼리 실행 계획을 모니터링하여 현재 데이터베이스 로드에 기여하는 실행 계획을 감지하고 시간 경과에 따른 실행 계획의 성능 통계를 추적합니다. 자세한 내용은 Aurora PostgreSQL용 쿼리 실행 계획 모니터링을 참조하세요.	on
authentication_timeout	(초) 클라이언트 인증을 완료하는 데 허용되는 최대 시간을 설정합니다.	–
auto_explain.log_analyze	계획 로깅에 EXPLAIN ANALYZE를 사용합니다.	–
auto_explain.log_buffers	버퍼 사용량을 로그합니다.	–
auto_explain.log_format	계획 로깅에 사용할 EXPLAIN 형식입니다.	–
auto_explain.log_min_duration	계획이 로그되는 최소 실행 시간을 설정합니다.	–
auto_explain.log_nested_statements	중첩 문을 로그합니다.	–
auto_explain.log_timing	행 개수뿐만 아니라 타이밍 데이터를 수집합니다.	–
auto_explain.log_triggers	계획에 트리거 통계를 포함합니다.	–

파라미터 이름	설명	Default
auto_explain.log_verbose	계획 로깅에 EXPLAIN VERBOSE를 사용합니다.	–
auto_explain.sample_rate	처리할 쿼리의 비율입니다.	–
autovacuum	autovacuum 서브프로세스를 시작합니다.	–
autovacuum_analyze_scale_factor	분석 전 튜플 삽입 업데이트 또는 삭제의 수(릴튜플의 비율)입니다.	0.05
autovacuum_analyze_threshold	분석 전 최소 튜플 삽입 업데이트 또는 삭제 횟수입니다.	–
autovacuum_freeze_max_age	트랜잭션 ID 랩어라운드를 방지하기 위한 테이블의 autovacuum 기간을 지정합니다.	–
autovacuum_max_workers	autovacuum 작업자 프로세스를 동시에 실행할 수 있는 최대 수를 설정합니다.	GREATEST(DBInstanceClassMemory/64371566592,3)
autovacuum_multixact_freeze_max_age	Multixact 랩어라운드를 방지하기 위해 테이블을 자동 정리할 multixact 기간입니다.	–
autovacuum_naptime	(초) 자동 정리 실행 간 절전 시간을 지정합니다.	5
autovacuum_vacuum_cost_delay	(밀리초) 자동 정리에 대한 정리 비용 지연 시간(밀리초)입니다.	5
autovacuum_vacuum_cost_limit	autovacuum에서 지연 시간 이전에 이용 가능한 vacuum 코스트 값을 지정합니다.	GREATEST(log(DBInstanceClassMemory/21474836480)*600,200)
autovacuum_vacuum_insert_scale_factor	정리 전 튜플 삽입 횟수(릴튜플의 비율)입니다.	–

파라미터 이름	설명	Default
autovacuum_vacuum_insert_threshold	정리 전 최소 튜플 삽입 횟수 또는 -1(삽입 정리 사용 중지)입니다.	–
autovacuum_vacuum_scale_factor	정리 전 튜플 업데이트 또는 삭제 횟수(릴튜플의 비율)입니다.	0.1
autovacuum_vacuum_threshold	정리 전 최소 튜플 업데이트 또는 삭제 횟수입니다.	–
autovacuum_work_mem	(kB) 각 자동 정리 작업자 프로세스에서 사용할 최대 메모리를 설정합니다.	GREATEST(DBInstanceClassMemory/32768,131072)
babelfishpg_tds.default_server_name	Babelfish 서버의 기본 이름입니다.	Microsoft SQL Server
babelfishpg_tds.listen_addresses	TDS를 수신할 호스트 이름이나 IP 주소를 설정합니다.	*
babelfishpg_tds.port	서버가 수신하는 TDS TCP 포트를 설정합니다.	1433
babelfishpg_tds.tds_debug_log_level	TDS에서 로깅 수준을 설정합니다. 0을 설정하면 로깅이 사용 중지됩니다.	1
babelfishpg_tds.tds_default_numeric_precision	엔진에서 지정하지 않은 경우 TDS 열 메타데이터에서 전송할 숫자 유형의 기본 정밀도를 설정합니다.	38
babelfishpg_tds.tds_default_numeric_scale	엔진에서 지정하지 않은 경우 TDS 열 메타데이터에서 전송할 숫자 유형의 기본 배율을 설정합니다.	8
babelfishpg_tds.tds_default_packet_size	연결 중인 모든 SQL Server 클라이언트의 기본 패킷 크기를 설정합니다.	4096

파라미터 이름	설명	Default
babelfishpg_tds.tds_default_protocol_version	연결 중인 모든 클라이언트에 대한 기본 TDS 프로토콜 버전을 설정합니다.	DEFAULT
babelfishpg_tds.tds_ssl_encrypt	SSL 암호화 옵션을 설정합니다.	0
babelfishpg_tds.tds_ssl_max_protocol_version	TDS 세션에 사용할 최대 SSL/TLS 프로토콜 버전을 설정합니다.	TLSv1.2
babelfishpg_tds.tds_ssl_min_protocol_version	TDS 세션에 사용할 최소 SSL/TLS 프로토콜 버전을 설정합니다.	Aurora PostgreSQL 버전 16의 TLSv1.2, Aurora PostgreSQL 버전 16 이전 버전의 경우 TLSv1
babelfishpg_tsql.default_locale	CREATE COLLATION에 의해 생성된 데이터 정렬에 사용할 기본 로캘입니다.	en-US
babelfishpg_tsql.migration_mode	여러 사용자 데이터베이스가 지원되는지 여부를 정의합니다.	Aurora PostgreSQL 버전 16의 multi-db, Aurora PostgreSQL 버전 16 이전 버전의 경우 single-db
babelfishpg_tsql.server_collation_name	기본 서버 콜레이션 이름입니다.	sql_latin1_general_cp1_ci_as
babelfishpg_tsql.version	@@VERSION 변수의 출력을 설정합니다.	기본값
backend_flush_after	(8Kb) 이전에 수행된 쓰기가 디스크로 플러시된 이후의 페이지 수입니다.	–
backslash_quote	문자열 리터럴에서 \\ 허용 여부를 설정합니다.	–

파라미터 이름	설명	Default
backtrace_functions	이러한 함수의 오류에 대한 백트레이스를 로깅합니다.	–
bytea_output	바이트의 출력 형식을 설정합니다.	–
check_function_bodies	CREATE FUNCTION 중 함수 본문을 검사합니다.	–
client_connection_check_interval	쿼리를 실행하는 동안 연결 끊김 검사 사이의 시간 간격을 설정합니다.	–
client_encoding	클라이언트 문자 세트 인코딩을 설정합니다.	UTF8
client_min_messages	클라이언트에게 보여지는 메시지 수준을 설정합니다.	–
compute_query_id	쿼리 식별자를 계산합니다.	auto
config_file	서버 기본 구성 파일을 설정합니다.	/rdsdbdata/config/postgresql.conf
constraint_exclusion	planner가 제약 조건을 사용하여 쿼리를 최적화하도록 활성화합니다.	–
cpu_index_tuple_cost	인덱스 스캔 중 각 인덱스 항목을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cpu_operator_cost	각 연산자 또는 함수 호출을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cpu_tuple_cost	각 튜플(행)을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cron.database_name	pg_cron 메타데이터 테이블을 저장하도록 데이터베이스를 설정합니다.	postgres
cron.log_run	모든 작업 실행을 job_run_details 테이블에 로그합니다.	켜짐

파라미터 이름	설명	Default
cron.log_statement	실행 전에 모든 cron 문을 로깅합니다.	끄기
cron.max_running_jobs	동시에 실행할 수 있는 최대 작업 수입니다.	5
cron.use_background_workers	pg_cron에 대한 백그라운드 작업자를 사용 설정합니다.	켜짐
cursor_tuple_fraction	플래너가 예상하는 검색할 커서 행의 분수 값을 설정합니다.	–
data_directory	서버의 데이터 디렉터리를 설정합니다.	/rdsdbdata/db
datestyle	날짜와 시간 값에 대한 표시 형식을 설정합니다.	–
db_user_namespace	데이터베이스별 사용자 이름을 사용 설정합니다.	–
deadlock_timeout	(밀리초) 교착 상태 여부를 확인하기 전 잠금 대기 시간을 설정합니다.	–
debug_pretty_print	구문과 실행 계획 트리를 들여쓰기 하여 표시합니다.	–
debug_print_parse	각 쿼리의 구문 분석 트리를 로그합니다.	–
debug_print_plan	각 쿼리의 실행 계획을 로그합니다.	–
debug_print_rewritten	각 쿼리에서 재작성된 구문 분석 트리를 로그합니다.	–
default_statistics_target	기본 통계 대상을 설정합니다.	–
default_tablespace	테이블과 인덱스를 생성할 기본 테이블스페이스를 설정합니다.	–

파라미터 이름	설명	Default
default_toast_compression	압축 가능한 값에 대한 기본 압축 방법을 설정합니다.	–
default_transaction_deferrable	새로운 트랜잭션의 기본 deferrable 상태를 설정합니다.	–
default_transaction_isolation	새로운 트랜잭션마다 트랜잭션 격리 수준을 설정합니다.	–
default_transaction_read_only	새로운 트랜잭션의 기본 읽기 전용 상태를 설정합니다.	–
effective_cache_size	(8kB) 디스크 캐시 크기에 대한 플래너의 가정을 설정합니다.	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	디스크 하위 시스템에서 효율적으로 동시에 처리할 수 있는 요청 수를 지정합니다.	–
enable_async_append	플래너가 비동기 추가 계획을 사용할 수 있도록 합니다.	–
enable_bitmapscan	플래너가 비트맵 스캔 계획을 사용할 수 있도록 합니다.	–
enable_gathermerge	플래너가 수집 병합 계획을 사용할 수 있도록 합니다.	–
enable_hashagg	플래너가 해시된 집계 계획을 사용할 수 있도록 합니다.	–
enable_hashjoin	플래너가 해시 조인 계획을 사용할 수 있도록 합니다.	–
enable_incremental_sort	플래너가 증분 정렬 단계를 사용할 수 있도록 합니다.	–

파라미터 이름	설명	Default
enable_indexonlyscan	플래너가 인덱스 전용 스캔 계획을 사용할 수 있도록 합니다.	–
enable_indexscan	플래너가 인덱스 스캔 계획을 사용할 수 있도록 합니다.	–
enable_material	플래너가 구체화를 사용할 수 있도록 합니다.	–
enable_memoize	플래너가 메모이제이션을 사용할 수 있도록 합니다.	–
enable_mergejoin	플래너가 병합 조인 계획을 사용할 수 있도록 합니다.	–
enable_nestloop	플래너가 중첩 루프 조인 계획을 사용할 수 있도록 합니다.	–
enable_parallel_append	플래너가 병렬 추가 계획을 사용할 수 있도록 합니다.	–
enable_parallel_hash	플래너가 병렬 해시 계획을 사용할 수 있도록 합니다.	–
enable_partition_pruning	계획 시간 및 실행 시간 파티션 정리를 사용 설정합니다.	–
enable_partitionwise_aggregate	파티션별 집계 및 그룹화를 사용 설정합니다.	–
enable_partitionwise_join	파티션별 조인을 사용 설정합니다.	–
enable_seqscan	플래너가 순차적 스캔 계획을 사용할 수 있도록 합니다.	–
enable_sort	플래너가 명시적 정렬 단계를 사용할 수 있도록 합니다.	–

파라미터 이름	설명	Default
enable_tidscan	플래너가 TID 스캔 계획을 사용할 수 있도록 합니다.	–
escape_string_warning	일반 문자열 리터럴의 백슬래시 이스케이프에 대해 경고합니다.	–
exit_on_error	오류 발생 시 세션을 종료합니다.	–
extra_float_digits	부동 소수점으로 표시할 자릿수를 설정합니다.	–
force_parallel_mode	병렬 쿼리 기능을 강제로 사용합니다.	–
from_collapse_limit	서브 쿼리가 축소되지 않는 FROM 목록 크기를 설정합니다.	–
geqo	유전적 쿼리 최적화를 활성화합니다.	–
geqo_effort	GEQO: 다른 GEQO 파라미터의 기본값을 설정하는 데 사용됩니다.	–
geqo_generations	GEQO: 알고리즘의 반복 횟수입니다.	–
geqo_pool_size	GEQO: 모집단의 개체 수입니다.	–
geqo_seed	GEQO: 무작위 경로 선택을 위한 시드(seed)를 지정합니다.	–
geqo_selection_bias	GEQO: 모집단 내 선택적 압력을 지정합니다.	–
geqo_threshold	GEQO가 사용되는 FROM 항목의 임계값을 설정합니다.	–
gin_fuzzy_search_limit	정확한 GIN 기준 검색에 허용되는 최대 결과 수를 설정합니다.	–
gin_pending_list_limit	(kB) GIN 인덱스에 대해 보류 중인 목록의 최대 크기를 설정합니다.	–

파라미터 이름	설명	Default
hash_mem_multiplier	해시 테이블에 사용할 work_mem의 배수입니다.	–
hba_file	서버 hba 구성 파일을 설정합니다.	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	쿼리 충돌을 피하기 위해 상시 대기에서 프라이머리로 피드백을 허용합니다.	켜짐
huge_pages	DB 인스턴스가 공유 버퍼에서 사용하는 것과 같은 대규모 연속 메모리 청크로 작업할 때 오버헤드를 줄입니다. t3.medium, db.t3.large, db.t4g.medium, db.t4g.large 인스턴스 클래스를 제외한 모든 DB 인스턴스 클래스에서 기본적으로 켜져 있습니다.	on
ident_file	서버 ident 구성 파일을 설정합니다.	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(밀리초) 유휴 트랜잭션의 최대 허용 기간을 설정합니다.	86400000
idle_session_timeout	열린 트랜잭션 내에 있지 않으면서 지정된 시간보다 오랫동안 유휴 상태(즉, 클라이언트 쿼리를 기다리는 중)인 세션을 종료합니다.	–
intervalstyle	간격 값에 대한 표시 형식을 설정합니다.	–
join_collapse_limit	JOIN 구문이 결합되지 않는 FROM 목록 크기를 설정합니다.	–

파라미터 이름	설명	Default
krb_caseins_users	GSSAPI(일반 보안 서비스 API) 사용자 이름을 대소문자를 구분하지 않고 처리할지(true) 여부를 설정합니다. 기본적으로 이 파라미터는 false로 설정되므로, Kerberos는 사용자 이름이 대소문자를 구분할 것으로 예상합니다. 자세한 내용은 PostgreSQL 설명서에서 GSSAPI 인증을 참조하십시오.	false
lc_messages	메시지 표시 언어를 설정합니다.	–
lc_monetary	통화 금액의 형식으로 사용할 로캘을 설정합니다.	–
lc_numeric	숫자의 형식으로 사용할 로캘을 설정합니다.	–
lc_time	날짜와 시간 값의 형식으로 사용할 로캘을 설정합니다.	–
listen_addresses	수신할 호스트 이름이나 IP 주소를 설정합니다.	*
lo_compat_privileges	대형 객체에 대한 권한 검사를 위해 이전 버전과의 호환성 모드를 사용 설정합니다.	0
log_autovacuum_min_duration	(밀리초) 자동 정리 작업이 기록되는 최소 실행 시간을 설정합니다.	10000
log_connections	성공한 연결을 모두 기록합니다.	–
log_destination	서버 로그 출력의 대상을 설정합니다.	stderr
log_directory	로그 파일의 대상 디렉터리를 설정합니다.	/rdsdbdata/log/error
log_disconnections	지속 시간을 포함해 세션 종료를 기록합니다.	–
log_duration	완료된 개별 SQL 문의 지속 시간을 기록합니다.	–
log_error_verbosity	기록된 메시지의 세부 사항을 설정합니다.	–

파라미터 이름	설명	Default
log_executor_stats	실행기 성능 통계를 서버 로그에 기록합니다.	–
log_file_mode	로그 파일에 대한 파일 권한을 설정합니다.	0644
log_filename	로그 파일의 이름 패턴을 설정합니다.	postgresql.log.%Y-%m-%d-%H%M
logging_collector	하위 프로세스를 시작하여 stderr 출력 및/또는 csvlog를 로그 파일로 캡처합니다.	1
log_hostname	연결 로그에 호스트 이름을 기록합니다.	0
logical_decoding_work_mem	(kB) 디스크로 유출되기 전에 각 내부 재정렬 버퍼에서 이 정도의 메모리를 사용할 수 있습니다.	–
log_line_prefix	각 로그 행에 접두사가 붙은 정보를 제어합니다.	%t:%r:%u@%d:%p]:
log_lock_waits	오랜 잠금 대기 시간을 기록합니다.	–
log_min_duration_sample	(밀리초) 문 샘플이 로그되는 최소 실행 시간을 설정합니다. 샘플링은 log_statement_sample_rate에 의해 결정됩니다.	–
log_min_duration_statement	(밀리초) 문이 기록되는 최소 실행 시간을 설정합니다.	–
log_min_error_statement	이 수준 이상으로 오류 원인이 되는 모든 문을 로그합니다.	–
log_min_messages	기록되는 메시지 수준을 설정합니다.	–
log_parameter_max_length	(B) 문을 로그할 때 로그된 파라미터 값을 처음 N바이트로 제한합니다.	–
log_parameter_max_length_on_error	(B) 오류를 보고할 때 로그된 파라미터 값을 처음 N바이트로 제한합니다.	–

파라미터 이름	설명	Default
log_parser_stats	구문 분석기 성능 통계를 서버 로그에 기록합니다.	–
log_planner_stats	planner 성능 통계를 서버 로그에 기록합니다.	–
log_replication_commands	각 복제 명령을 로그합니다.	–
log_rotation_age	(분) N분 후 자동 로그 파일 회전이 수행됩니다.	60
log_rotation_size	(kB) N킬로바이트 후 자동 로그 파일 회전이 수행됩니다.	100000
log_statement	기록할 문 유형을 설정합니다.	–
log_statement_sample_rate	로그할 log_min_duration_sample을 초과하는 문의 비율입니다.	–
log_statement_stats	누적 성능 통계를 서버 로그에 기록합니다.	–
log_temp_files	(kB) 이 킬로바이트 수치보다 큰 임시 파일의 사용을 로그합니다.	–
log_timezone	로그 메시지에 사용할 표준 시간대를 설정합니다.	UTC
log_transaction_sample_rate	새 트랜잭션에 대해 로그할 트랜잭션 비율을 설정합니다.	–
log_truncate_on_rotation	로그 순환 중에 동일한 이름의 기존 로그 파일을 잘라냅니다.	0
maintenance_io_concurrency	유지 관리 작업에 사용되는 effective_io_concurrency의 변형입니다.	1

파라미터 이름	설명	Default
maintenance_work_mem	(kB) 유지 관리 작업에 사용할 최대 메모리를 설정합니다.	GREATEST(DBInstanceClassMemory/63963136*1024, 65536)
max_connections	동시에 접속할 수 있는 최대 수를 설정합니다.	LEAST(DBInstanceClassMemory/9531392, 5000)
max_files_per_process	서버 프로세스마다 파일을 동시에 열 수 있는 최대 수를 설정합니다.	–
max_locks_per_transaction	하나의 트랜잭션에서 사용할 수 있는 최대 잠금 횟수를 설정합니다.	64
max_logical_replication_workers	최대 논리적 복제 작업자 프로세스 수입니다.	–
max_parallel_maintenance_workers	유지 관리 작업당 최대 병렬 프로세스 수를 설정합니다.	–
max_parallel_workers	한 번에 활성화할 수 있는 최대 병렬 작업자 수를 설정합니다.	GREATEST($DBInstanceVCPU/2, 8)
max_parallel_workers_per_gather	실행기 노드당 최대 병렬 프로세스 수를 설정합니다.	–
max_pred_locks_per_page	페이지당 최대 술어 잠금 튜플 수를 설정합니다.	–
max_pred_locks_per_relation	관계당 최대 술어 잠금 페이지 및 튜플 수를 설정합니다.	–
max_pred_locks_per_transaction	하나의 트랜잭션에서 사용할 수 있는 최대 술어(predicate) 잠금 횟수를 설정합니다.	–

파라미터 이름	설명	Default
max_prepared_transactions	트랜잭션을 동시에 준비할 수 있는 최대 수를 설정합니다.	0
max_replication_slots	서버에서 지원할 수 있는 최대 복제 슬롯 수를 설정합니다.	20
max_slot_wal_keep_size	(MB) 복제 슬롯은 실패한 것으로 표시되고 디스크에서 WAL이 이 정도의 공간을 차지하는 경우 삭제 또는 재활용을 위해 세그먼트가 해제됩니다.	–
max_stack_depth	(kB) 최대 스택 깊이(킬로바이트)를 설정합니다.	6144
max_standby_streaming_delay	(밀리초) 상시 대기 서버가 스트리밍된 WAL 데이터를 처리할 때 쿼리 취소까지 걸리는 최대 지연 시간을 설정합니다.	14000
max_sync_workers_per_subscription	구독당 최대 동기화 작업자 수	2
max_wal_senders	동시에 실행되는 최대 WAL 발신자 프로세스 수를 설정합니다.	10
max_worker_processes	최대 동시 작업자 프로세스 수를 설정합니다.	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) 시작 시 예약된 동적 공유 메모리의 양입니다.	–
min_parallel_index_scan_size	(8kB) 병렬 스캔을 위한 최소 인덱스 데이터 양을 설정합니다.	–
min_parallel_table_scan_size	(8kB) 병렬 스캔을 위한 최소 테이블 데이터 양을 설정합니다.	–

파라미터 이름	설명	Default
old_snapshot_threshold	(분) 스냅샷이 너무 오래되어 스냅샷이 생성된 후 변경된 페이지를 읽을 수 없게 될 때까지의 시간입니다.	–
orafce.nls_date_format	오라클의 날짜 출력 동작을 에뮬레이션합니다.	–
orafce.timezone	sysdate 함수에 사용되는 시간대를 지정합니다.	–
parallel_leader_participation	수집 및 수집 병합이 하위 계획도 실행할지 여부를 제어합니다.	–
parallel_setup_cost	병렬 쿼리에 대한 작업자 프로세스 시작 비용의 플래너 추정치를 설정합니다.	–
parallel_tuple_cost	작업자에서 마스터 백엔드로 각 튜플(행)을 전달하는 비용의 플래너 추정치를 설정합니다.	–
password_encryption	암호를 암호화합니다.	–
pgaudit.log	세션 감사 로깅에 의해 로그될 문의 클래스를 지정합니다.	–
pgaudit.log_catalog	문의 모든 관계가 pg_catalog에 있는 경우 세션 로깅을 사용하도록 지정합니다.	–
pgaudit.log_level	로그 항목에 사용할 로그 수준을 지정합니다.	–
pgaudit.log_parameter	감사 로깅에 문과 함께 전달된 파라미터가 포함되도록 지정합니다.	–
pgaudit.log_relation	세션 감사 로깅에서 SELECT 또는 DML 문에서 참조되는 각 관계(TABLE, VIEW 등)에 대해 별도의 로그 항목을 생성해야 하는지 여부를 지정합니다.	–

파라미터 이름	설명	Default
pgaudit.log_statement_once	로깅에 문/하위 문 조합에 대한 첫 번째 로그 항목이 있는 문 텍스트 및 파라미터를 포함할지 아니면 모든 항목이 있는 문 텍스트 및 파라미터를 포함할지를 지정합니다.	–
pgaudit.role	객체 감사 로깅에 사용할 마스터 역할을 지정합니다.	–
pg_bigm.enable_recheck	전체 텍스트 검색의 내부 프로세스인 재검사를 수행할지 여부를 지정합니다.	켜짐
pg_bigm.gin_key_limit	전체 텍스트 검색에 사용할 검색 키워드의 최대 2그램 수를 지정합니다.	0
pg_bigm.last_update	pg_bigm 모듈의 마지막 업데이트 날짜를 보고합니다.	2013.11.22
pg_bigm.similarity_limit	유사성 검색에 사용되는 최소 임계값을 지정합니다.	0.3
pg_hint_plan.debug_print	힌트 구문 분석 결과를 로그합니다.	–
pg_hint_plan.enable_hint	플래너가 쿼리 앞의 힌트 설명에 지정된 계획을 사용하도록 합니다.	–
pg_hint_plan.enable_hint_table	테이블 조회를 사용하여 플래너가 힌트를 얻지 못하도록 합니다.	–
pg_hint_plan.message_level	디버그 메시지의 메시지 수준입니다.	–
pg_hint_plan.parse_messages	구문 분석 오류의 메시지 수준입니다.	–
pglogical.batch_inserts	가능한 경우 배치 삽입	–

파라미터 이름	설명	Default
pglogical.conflict_log_level	해결된 충돌을 로깅하는 데 사용되는 로그 수준을 설정합니다.	–
pglogical.conflict_resolution	해결 가능한 충돌에 대한 충돌 해결에 사용되는 방법을 설정합니다.	–
pglogical.extra_connection_options	모든 피어 노드 연결에 추가할 연결 옵션	–
pglogical.synchronous_commit	pglogical 특정 동기 커밋 값입니다.	–
pglogical.use_spi	변경 사항 적용에 하위 수준 API 대신 SPI 사용	–
pgtle.clientauth_databases_to_skip	clientauth 기능을 사용하기 위해 건너뛰어야 할 데이터베이스 목록입니다.	–
pgtle.clientauth_db_name	clientauth 기능에 사용할 데이터베이스를 제어합니다.	–
pgtle.clientauth_num_parallel_workers	clientauth 기능에 사용되는 백그라운드 워커 수입니다.	–
pgtle.clientauth_users_to_skip	clientauth 기능을 사용하기 위해 건너뛰어야 할 사용자 목록입니다.	–
pgtle.enable_clientauth	clientauth 기능을 활성화합니다.	–
pgtle.passcheck_db_name	클러스터 전체 passcheck 기능에 사용할 데이터베이스를 설정합니다.	–
pg_prewarm.autoprewarm	자동 사전 웜 작업자를 시작합니다.	–
pg_prewarm.autoprewarm_interval	공유 버퍼의 덤프 간격을 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.block_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.block_threshold	블록 유사성 함수에 사용되는 임계값을 설정합니다.	–
pg_similarity.block_tokenizer	블록 유사성 함수의 토크나이저를 설정합니다.	–
pg_similarity.cosine_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.cosine_threshold	코사인 유사성 함수에 사용되는 임계값을 설정합니다.	–
pg_similarity.cosine_tokenizer	코사인 유사성 함수의 토크나이저를 설정합니다.	–
pg_similarity.dice_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.dice_threshold	주사위 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.dice_tokenizer	주사위 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.euclidean_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.euclidean_threshold	유클리드 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.euclidean_tokenizer	유클리드 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.hamming_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.hamming_threshold	블록 유사성 지수에 사용되는 임계값을 설정합니다.	–
pg_similarity.jaccard_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jaccard_threshold	자카드 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.jaccard_tokenizer	자카드 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.jaro_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jaro_threshold	자로 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.jarowinkler_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jarowinkler_threshold	자로윙클러 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.levenshtein_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.levenshtein_threshold	레벤슈타인 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.matching_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.matching_threshold	일치 계수 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.matching_tokenizer	일치 계수 유사성 측정에 대한 토크나이저를 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.mongeelkan_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.mongeelkan_threshold	몽게-엘칸 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.mongeelkan_tokenizer	몽게-엘칸 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.nw_gap_penalty	니들만 브니쉬 유사성 측정에 사용되는 갭 페널티를 설정합니다.	–
pg_similarity.nw_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.nw_threshold	니들레만-운쉬 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.overlap_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.overlap_threshold	겹침 계수 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.overlap_tokenizer	겹침 계수 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.qgram_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.qgram_threshold	Q-그램 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.qgram_tokenizer	Q-그램 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.swg_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.swg_threshold	스미스-워터맨-고토 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.sw_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.sw_threshold	스미스-워터맨 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_stat_statements.max	pg_stat_statements에서 추적하는 최대 문 수를 설정합니다.	–
pg_stat_statements.save	서버 종료 시 pg_stat_statements 통계를 저장합니다.	–
pg_stat_statements.track	pg_stat_statements로 추적할 문을 선택합니다.	–
pg_stat_statements.track_planning	pg_stat_statements로 계획 기간을 추적할지 여부를 선택합니다.	–
pg_stat_statements.track_utility	pg_stat_statements로 유틸리티 명령을 추적할지 여부를 선택합니다.	–
plan_cache_mode	사용자 정의 또는 일반 계획의 플래너 선택을 제어합니다.	–
포트	서버가 수신하는 TCP 포트를 설정합니다.	EndPointPort
postgis.gdal_enabled_drivers	Postgres 9.3.5 이상에서 PostGIS와 함께 사용되는 GDAL 드라이버를 사용 설정하거나 사용 중지합니다.	ENABLE_ALL
quote_all_identifiers	SQL 조각 생성 시 모든 식별자에 인용 부호를 추가합니다.	–
random_page_cost	비순차적으로 가져온 디스크 페이지에 대한 플래너의 예상 비용을 설정합니다.	–

파라미터 이름	설명	Default
rdkit.dice_threshold	주사위 유사도의 하한 임계값입니다. 유사도가 임계값보다 낮은 분자는 # 연산에서 유사하지 않습니다.	–
rdkit.do_chiral_sss	하부 구조 일치에서 입체 화학을 고려해야 합니다. false인 경우 하위 구조 일치에 입체 화학 정보가 사용되지 않습니다.	–
rdkit.tanimoto_threshold	타니모토 유사도의 하한 임계값입니다. 유사도가 임계값보다 낮은 분자는 % 연산에서 유사하지 않습니다.	–
rds.accepted_password_auth_method	로컬에 저장된 암호를 사용하여 연결을 강제로 인증합니다.	md5+scram
rds.adaptive_autovacuum	적응형 자동 정리를 사용 설정/사용 중지하는 RDS 파라미터입니다.	1
rds.babelfish_status	Babelfish for Aurora PostgreSQL을 사용 설정/사용 중지하는 RDS 파라미터입니다.	끄기
rds.enable_plan_management	apg_plan_mgmt 확장을 사용 설정하거나 사용 중지합니다.	0

파라미터 이름	설명	Default
SHOw extensions	RDS에서 제공하는 확장 목록입니다.	address_standardizer, address_standardizer_data_us, apg_plan_mgmt, aurora_stat_utils, amcheck, autoinc, aws_commons, aws_ml, aws_s3, aws_lambda, bool_plperl, bloom, btree_gin, btree_gist, citext, cube, dblink, dict_int, dict_xsyn, earthdistance, fuzzystrmatch, hll, hstore, hstore_plperl, insert_username, intagg, intarray, ip4r, isn, jsonb_plperl, lo, log_fdw, ltree, moddatetime, old_snapshot, oracle_fdw, orafce, pgaudit, pgcrypto, pglogical, pgrouting, pgrowlocks, pgstattuple, pgtap, pg_bigm, pg_buffercache, pg_cron, pg_freespacemap, pg_hint_plan, pg_partman, pg_prewarm, pg_proctab, pg_repack, pg_simila

파라미터 이름	설명	Default
		rity, pg_stat_statements, pg_trgm, pg_visibility, plcoffee, plls, plperl, plpgsql, plprofiler, pltcl, plv8, postgis, postgis_tiger_geocoder, postgis_raster, postgis_topology, postgres_fdw, prefix, rdkit, rds_tools, refint, sslinfo, tablefunc, tds_fdw, test_parser, tsm_system_rows, tsm_system_time, unaccent, uuid-ossp
rds.force_admin_logging_level	고객 데이터베이스의 RDS 관리자 사용자 작업에 대한 로그 메시지를 참조하세요.	–
rds.force_autovacuum_logging_level	자동 정리 작업과 관련된 로그 메시지를 참조하세요.	경고
rds.force_ssl	SSL 연결을 강제 설정합니다.	0

파라미터 이름	설명	Default
rds.global_db_rpo	(초) 위반 시 사용자 커밋을 차단하는 복구 시점 목표 임계값(초)입니다. **Important** 이 파라미터는 Aurora PostgreSQL 기반 글로벌 데이터베이스를 위한 것입니다. 비글로벌 데이터베이스의 경우 기본값을 그대로 둡니다. 이 파라미터에 대한 자세한 내용은 the section called “RPO(Aurora PostgreSQL) 관리” 섹션을 참조하세요.	–
rds.logical_replication	논리적 디코딩을 사용 설정합니다.	0
rds.logically_replicate_unlogged_tables	로깅되지 않은 테이블은 논리적으로 복제됩니다.	1
rds.log_retention_period	Amazon RDS가 N분보다 오래된 PostgreSQL 로그를 삭제합니다.	4320
rds.pg_stat_ramdisk_size	통계 램디스크의 크기(MB)입니다. 값이 0이 아니면 램디스크가 설정됩니다. 이 파라미터는 Aurora PostgreSQL 14 이하 버전에서만 사용할 수 있습니다.	0
rds.rds_superuser_reserved_connections	rds_superuser용으로 예약된 연결 슬롯 수를 설정합니다. 이 파라미터는 버전 15 이상에서만 사용할 수 있습니다. 자세한 내용은 PostgreSQL 설명서의 reserved connections를 참조하세요.	2
rds.restrict_password_commands	암호 관련 명령을 rds_password 멤버로 제한합니다.	–
rds.superuser_variables	rds_superuser 수정 문을 승격시키는 수퍼 사용자 전용 변수 목록입니다.	session_replication_role

파라미터 이름	설명	Default
recovery_init_sync_method	충돌 복구 전에 데이터 디렉터리를 동기화하는 방법을 설정합니다.	syncfs
remove_temp_files_after_crash	백엔드 충돌 후 임시 파일을 제거합니다.	0
restart_after_crash	백엔드 충돌 후 서버를 다시 초기화합니다.	–
row_security	행 보안을 사용 설정합니다.	–
search_path	스키마로 한정되지 않은 이름의 스키마 검색 순서를 설정합니다.	–
seq_page_cost	순차적으로 가져온 디스크 페이지에 대한 플래너의 예상 비용을 설정합니다.	–
session_replication_role	트리거 및 다시 쓰기 규칙에 대한 세션 동작을 설정합니다.	–
shared_buffers	(8kB) 서버에서 사용하는 공유 메모리 버퍼 수를 설정합니다.	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	서버에 미리 로드할 공유 라이브러리를 나열합니다.	pg_stat_statements
ssl	SSL 연결을 활성화합니다.	1
ssl_ca_file	SSL 서버 권한 파일의 위치입니다.	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 서버 인증서 파일의 위치입니다.	/rdsdbdata/rds-metadata/server-cert.pem
ssl_ciphers	보안 연결에 사용할 수 있는 TLS 암호 목록을 설정합니다.	–

파라미터 이름	설명	Default
ssl_crl_dir	SSL 인증서 취소 목록 디렉토리의 위치입니다.	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL 서버 프라이빗 키 파일의 위치입니다.	/rdsdbdata/rds-metadata/server-key.pem
ssl_max_protocol_version	허용되는 최대 SSL/TLS 프로토콜 버전을 설정합니다.	–
ssl_min_protocol_version	허용되는 최소 SSL/TLS 프로토콜 버전을 설정합니다.	TLSv1.2
standard_conforming_strings	... 문자열에서 백슬래시가 리터럴로 처리됩니다.	–
statement_timeout	(밀리초) 모든 문의 최대 허용 기간을 설정합니다.	–
stats_temp_directory	임시 통계 파일을 지정된 디렉터리에 씁니다.	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	슈퍼 사용자용으로 예약된 연결 슬롯 수를 설정합니다.	3
synchronize_seqscans	동기 방식의 순차적 스캔을 사용 설정합니다.	–
synchronous_commit	현재 트랜잭션 동기화 수준을 설정합니다.	켜짐
tcp_keepalives_count	TCP keepalive의 최대 재전송 횟수를 지정합니다.	–
tcp_keepalives_idle	(초) TCP keepalive의 실행 간격을 지정합니다.	–
tcp_keepalives_interval	(초) TCP keepalive의 재전송 간격을 지정합니다.	–

파라미터 이름	설명	Default
temp_buffers	(8kB) 각 세션에서 사용하는 임시 버퍼의 최대 수를 설정합니다.	–
temp_file_limit	명시적 임시 테이블에 사용되는 공간을 제외하고 주어진 PostgreSQL 프로세스가 임시 파일에 사용할 수 있는 총 디스크 공간(KB)을 제한합니다.	-1
temp_tablespaces	임시 테이블 및 정렬 파일에 사용할 테이블스페이스를 설정합니다.	–
timezone	타임스탬프를 표시 및 해석할 시간대를 설정합니다.	UTC
track_activities	명령 실행에 대한 정보를 수집합니다.	–
track_activity_query_size	pg_stat_activity.current_query에 예약되는 크기(바이트)를 설정합니다.	4096
track_commit_timestamp	트랜잭션 커밋 시간을 수집합니다.	–
track_counts	데이터베이스 작업에 관한 통계를 수집합니다.	–
track_functions	데이터베이스 작업에 관한 함수 수준 통계를 수집합니다.	pl
track_io_timing	데이터베이스 I/O 작업에 대한 시간 통계를 수집합니다.	1
track_wal_io_timing	WAL I/O 활동에 관한 시간 통계를 수집합니다.	–
transform_null_equals	expr=NULL을 expr IS NULL로 처리합니다.	–
update_process_title	프로세스 제목을 업데이트하여 활성 SQL 명령을 표시합니다.	–
vacuum_cost_delay	(밀리초) 정리 비용 지연 시간(밀리초)입니다.	–

파라미터 이름	설명	Default
vacuum_cost_limit	지연 시간 이전에 이용 가능한 vacuum 코스트 값을 지정합니다.	–
vacuum_cost_page_dirty	vacuum으로 페이지 변경 시 부과되는 vacuum 코스트를 지정합니다.	–
vacuum_cost_page_hit	버퍼 캐시에서 발견되는 페이지에 대한 vacuum 코스트를 지정합니다.	–
vacuum_cost_page_miss	버퍼 캐시에서 발견되지 않는 페이지에 대한 vacuum 코스트를 지정합니다.	0
vacuum_defer_cleanup_age	VACUUM 및 HOT 정리를 연기해야 하는 트랜잭션 수입니다(있는 경우).	–
vacuum_failsafe_age	VACUUM에서 안전 장치가 작동하여 랩어라운드 정전을 방지해야 하는 기간입니다.	1200000000
vacuum_freeze_min_age	VACUUM에서 테이블 행을 고정해야 하는 최소 기간입니다.	–
vacuum_freeze_table_age	VACUUM에서 전체 테이블을 스캔하여 튜플을 고정해야 하는 기간입니다.	–
vacuum_multixact_failsafe_age	VACUUM에서 안전 장치가 작동하여 랩어라운드 정전을 방지해야 하는 Multixact 기간입니다.	1200000000
vacuum_multixact_freeze_min_age	VACUUM에서 테이블 행의 MultiXactId를 고정해야 하는 최소 기간입니다.	–
vacuum_multixact_freeze_table_age	VACUUM에서 전체 테이블을 스캔하여 튜플을 고정해야 하는 multixact 기간입니다.	–
wal_buffers	(8kB) WAL을 위해 공유 메모리에서 사용할 디스크 페이지 버퍼 수를 설정합니다.	–

파라미터 이름	설명	Default
wal_receiver_create_temp_slot	영구 슬롯이 구성되지 않은 경우 WAL 수신기에서 임시 복제 슬롯을 생성할지 여부를 설정합니다.	0
wal_receiver_status_interval	(초) WAL 수신기 상태 보고 간 최대 간격을 프라이머리로 설정합니다.	–
wal_receiver_timeout	(밀리초) 프라이머리로부터 데이터를 수신할 최대 대기 시간을 설정합니다.	30000
wal_sender_timeout	(밀리초) WAL 복제를 기다리는 최대 시간을 설정합니다.	–
work_mem	(kB) 쿼리 작업 공간에 사용할 최대 메모리를 설정합니다.	–
xmlbinary	XML에서 바이너리 값의 인코딩 방식을 설정합니다.	–
xmloption	암시적 구문 분석 및 직렬화 작업에서 XML 데이터를 문서 또는 내용 조각으로 간주할지 여부를 설정합니다.	–

Aurora PostgreSQL 인스턴스 수준 파라미터

AWS 관리 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 특정 Aurora PostgreSQL 버전에서 사용 가능한 인스턴스 수준 파라미터를 볼 수 있습니다. RDS 콘솔의 Aurora PostgreSQL DB 파라미터 그룹에서 파라미터 보기에 대한 자세한 정보는 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 값 보기 섹션을 참조하세요.

일부 인스턴스 수준 파라미터는 일부 버전에서 사용할 수 없으며 일부는 더 이상 사용되지 않습니다. 특정 Aurora PostgreSQL 버전의 파라미터 보기에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터 및 DB 파라미터 보기 섹션을 참조하세요.

예를 들어, 다음 테이블에는 Aurora PostgreSQL DB 클러스터의 특정 DB 클러스터에 적용되는 파라미터가 나와 있습니다. 이 목록은 `--db-parameter-group-name` 값에 대해 `default.aurora-postgresql14`과 함께 describe-db-parameters AWS CLI 명령을 실행하여 생성되었습니다.

동일한 기본 DB 파라미터 그룹에 대한 DB 클러스터 파라미터 목록에 대한 내용은 Aurora PostgreSQL 클러스터 수준 파라미터 단원을 참조하세요.

파라미터 이름	설명	Default
apg_enable_batch_mode_function_execution	배치 모드 기능을 사용하여 한 번에 일련의 행을 처리합니다.	–
apg_enable_correlated_any_transform	가능한 경우 플래너가 상관 관계가 있는 ANY 하위 링크(IN/NOT IN 하위 쿼리)를 JOIN으로 변환할 수 있습니다.	–
apg_enable_function_migration	플래너가 적합한 스칼라 함수를 FROM 절로 마이그레이션할 수 있습니다.	–
apg_enable_not_in_transform	플래너가 가능한 경우 NOT IN 하위 쿼리를 ANTI JOIN으로 변환할 수 있습니다.	–
apg_enable_remove_redundant_inner_joins	플래너가 중복 내부 조인을 제거할 수 있습니다.	–
apg_enable_semijoin_push_down	해시 조인에 세미조인 필터를 사용할 수 있습니다.	–

파라미터 이름	설명	Default
apg_plan_mgmt.capture_plan_baselines	캡처 계획 기준 모드입니다. 수동(manual) - 모든 SQL 문에 대해 계획 캡처 사용 설정, 끄기(off) - 계획 캡처 사용 중지, 자동(automatic) - 자격 기준을 충족하는 pg_stat_statements의 문에 계획 캡처 사용.	끄기
apg_plan_mgmt.max_databases	apg_plan_mgmt를 사용하여 쿼리를 관리할 수 있는 최대 데이터베이스 수를 설정합니다.	10
apg_plan_mgmt.max_plans	apg_plan_mgmt로 캐시할 수 있는 최대 계획 수를 설정합니다.	10000
apg_plan_mgmt.plan_retention_period	계획이 자동으로 삭제되기 전에 계획의 last_used 이후 최대 일수입니다.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	미승인 계획이 실행될 예상 총 계획 비용입니다.	0
apg_plan_mgmt.use_plan_baselines	관리형 문에 대해 승인된 계획이나 고정된 계획만 사용합니다.	false
application_name	애플리케이션 이름이 통계 및 로그에 표시되도록 설정합니다.	–
aurora_compute_plan_id	쿼리 실행 계획을 모니터링하여 현재 데이터베이스 로드에 기여하는 실행 계획을 감지하고 시간 경과에 따른 실행 계획의 성능 통계를 추적합니다. 자세한 내용은 Aurora PostgreSQL용 쿼리 실행 계획 모니터링을 참조하세요.	on
aurora_temp_space_size	(MB) 지원되는 인스턴스 클래스가 있는 Aurora I/O 최적화 클러스터에서 최적화된 읽기 지원 임시 객체에 할당된 공간의 크기를 설정합니다.	DBInstanceClassMemory/524288
authentication_timeout	(초) 클라이언트 인증을 완료하는 데 허용되는 최대 시간을 설정합니다.	–

파라미터 이름	설명	Default
auto_explain.log_analyze	계획 로깅에 EXPLAIN ANALYZE를 사용합니다.	–
auto_explain.log_buffers	버퍼 사용량을 로그합니다.	–
auto_explain.log_format	계획 로깅에 사용할 EXPLAIN 형식입니다.	–
auto_explain.log_min_duration	계획이 로그되는 최소 실행 시간을 설정합니다.	–
auto_explain.log_nested_statements	중첩 문을 로그합니다.	–
auto_explain.log_timing	행 개수뿐만 아니라 타이밍 데이터를 수집합니다.	–
auto_explain.log_triggers	계획에 트리거 통계를 포함합니다.	–
auto_explain.log_verbose	계획 로깅에 EXPLAIN VERBOSE를 사용합니다.	–
auto_explain.sample_rate	처리할 쿼리의 비율입니다.	–
babelfishpg_tds.listen_addresses	TDS를 수신할 호스트 이름이나 IP 주소를 설정합니다.	*
babelfishpg_tds.tds_debug_log_level	TDS에서 로깅 수준을 설정합니다. 0을 설정하면 로깅이 사용 중지됩니다.	1
backend_flush_after	(8Kb) 이전에 수행된 쓰기가 디스크로 플러시된 이후의 페이지 수입니다.	–
bytea_output	바이트의 출력 형식을 설정합니다.	–

파라미터 이름	설명	Default
check_function_bodies	CREATE FUNCTION 중 함수 본문을 검사합니다.	–
client_connection_check_interval	쿼리를 실행하는 동안 연결 끊김 검사 사이의 시간 간격을 설정합니다.	–
client_min_messages	클라이언트에게 보여지는 메시지 수준을 설정합니다.	–
config_file	서버 기본 구성 파일을 설정합니다.	/rdsdbdata/config/postgresql.conf
constraint_exclusion	planner가 제약 조건을 사용하여 쿼리를 최적화하도록 활성화합니다.	–
cpu_index_tuple_cost	인덱스 스캔 중 각 인덱스 항목을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cpu_operator_cost	각 연산자 또는 함수 호출을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cpu_tuple_cost	각 튜플(행)을 처리하는 데 따른 플래너의 예상 비용을 설정합니다.	–
cron.database_name	pg_cron 메타데이터 테이블을 저장하도록 데이터베이스를 설정합니다.	postgres
cron.log_run	모든 작업 실행을 job_run_details 테이블에 로그합니다.	켜짐
cron.log_statement	실행 전에 모든 cron 문을 로깅합니다.	끄기
cron.max_running_jobs	동시에 실행할 수 있는 최대 작업 수입니다.	5
cron.use_background_workers	pg_cron에 대한 백그라운드 작업자를 사용 설정합니다.	켜짐

파라미터 이름	설명	Default
cursor_tuple_fraction	플래너가 예상하는 검색할 커서 행의 분수 값을 설정합니다.	–
db_user_namespace	데이터베이스별 사용자 이름을 사용 설정합니다.	–
deadlock_timeout	(밀리초) 교착 상태 여부를 확인하기 전 잠금 대기 시간을 설정합니다.	–
debug_pretty_print	구문과 실행 계획 트리를 들여쓰기 하여 표시합니다.	–
debug_print_parse	각 쿼리의 구문 분석 트리를 로그합니다.	–
debug_print_plan	각 쿼리의 실행 계획을 로그합니다.	–
debug_print_rewritten	각 쿼리에서 재작성된 구문 분석 트리를 로그합니다.	–
default_statistics_target	기본 통계 대상을 설정합니다.	–
default_transaction_deferrable	새로운 트랜잭션의 기본 deferrable 상태를 설정합니다.	–
default_transaction_isolation	새로운 트랜잭션마다 트랜잭션 격리 수준을 설정합니다.	–
default_transaction_read_only	새로운 트랜잭션의 기본 읽기 전용 상태를 설정합니다.	–
effective_cache_size	(8kB) 디스크 캐시 크기에 대한 플래너의 가정을 설정합니다.	SUM(DBInstanceClassMemory/12038,-50003
effective_io_concurrency	디스크 하위 시스템에서 효율적으로 동시에 처리할 수 있는 요청 수를 지정합니다.	–

파라미터 이름	설명	Default
enable_async_append	플래너가 비동기 추가 계획을 사용할 수 있도록 합니다.	–
enable_bitmapscan	플래너가 비트맵 스캔 계획을 사용할 수 있도록 합니다.	–
enable_gathermerge	플래너가 수집 병합 계획을 사용할 수 있도록 합니다.	–
enable_hashagg	플래너가 해시된 집계 계획을 사용할 수 있도록 합니다.	–
enable_hashjoin	플래너가 해시 조인 계획을 사용할 수 있도록 합니다.	–
enable_incremental_sort	플래너가 증분 정렬 단계를 사용할 수 있도록 합니다.	–
enable_indexonlyscan	플래너가 인덱스 전용 스캔 계획을 사용할 수 있도록 합니다.	–
enable_indexscan	플래너가 인덱스 스캔 계획을 사용할 수 있도록 합니다.	–
enable_material	플래너가 구체화를 사용할 수 있도록 합니다.	–
enable_memoize	플래너가 메모이제이션을 사용할 수 있도록 합니다.	–
enable_mergejoin	플래너가 병합 조인 계획을 사용할 수 있도록 합니다.	–
enable_nestloop	플래너가 중첩 루프 조인 계획을 사용할 수 있도록 합니다.	–
enable_parallel_append	플래너가 병렬 추가 계획을 사용할 수 있도록 합니다.	–

파라미터 이름	설명	Default
enable_parallel_hash	플래너가 병렬 해시 계획을 사용할 수 있도록 합니다.	–
enable_partition_pruning	계획 시간 및 실행 시간 파티션 정리를 사용 설정합니다.	–
enable_partitionwise_aggregate	파티션별 집계 및 그룹화를 사용 설정합니다.	–
enable_partitionwise_join	파티션별 조인을 사용 설정합니다.	–
enable_seqscan	플래너가 순차적 스캔 계획을 사용할 수 있도록 합니다.	–
enable_sort	플래너가 명시적 정렬 단계를 사용할 수 있도록 합니다.	–
enable_tidscan	플래너가 TID 스캔 계획을 사용할 수 있도록 합니다.	–
escape_string_warning	일반 문자열 리터럴의 백슬래시 이스케이프에 대해 경고합니다.	–
exit_on_error	오류 발생 시 세션을 종료합니다.	–
force_parallel_mode	병렬 쿼리 기능을 강제로 사용합니다.	–
from_collapse_limit	서브 쿼리가 축소되지 않는 FROM 목록 크기를 설정합니다.	–
geqo	유전적 쿼리 최적화를 활성화합니다.	–
geqo_effort	GEQO: 다른 GEQO 파라미터의 기본값을 설정하는 데 사용됩니다.	–
geqo_generations	GEQO: 알고리즘의 반복 횟수입니다.	–

파라미터 이름	설명	Default
geqo_pool_size	GEQO: 모집단의 개체 수입니다.	–
geqo_seed	GEQO: 무작위 경로 선택을 위한 시드(seed)를 지정합니다.	–
geqo_selection_bias	GEQO: 모집단 내 선택적 압력을 지정합니다.	–
geqo_threshold	GEQO가 사용되는 FROM 항목의 임계값을 설정합니다.	–
gin_fuzzy_search_limit	정확한 GIN 기준 검색에 허용되는 최대 결과 수를 설정합니다.	–
gin_pending_list_limit	(kB) GIN 인덱스에 대해 보류 중인 목록의 최대 크기를 설정합니다.	–
hash_mem_multiplier	해시 테이블에 사용할 work_mem의 배수입니다.	–
hba_file	서버 hba 구성 파일을 설정합니다.	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	쿼리 충돌을 피하기 위해 상시 대기에서 프라이머리로 피드백을 허용합니다.	켜짐
ident_file	서버 ident 구성 파일을 설정합니다.	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(밀리초) 유휴 트랜잭션의 최대 허용 기간을 설정합니다.	86400000
idle_session_timeout	열린 트랜잭션 내에 있지 않으면서 지정된 시간보다 오랫동안 유휴 상태(즉, 클라이언트 쿼리를 기다리는 중)인 세션을 종료합니다.	–
join_collapse_limit	JOIN 구문이 결합되지 않는 FROM 목록 크기를 설정합니다.	–

파라미터 이름	설명	Default
lc_messages	메시지 표시 언어를 설정합니다.	–
listen_addresses	수신할 호스트 이름이나 IP 주소를 설정합니다.	*
lo_compat_privileges	대형 객체에 대한 권한 검사를 위해 이전 버전과의 호환성 모드를 사용 설정합니다.	0
log_connections	성공한 연결을 모두 기록합니다.	–
log_destination	서버 로그 출력의 대상을 설정합니다.	stderr
log_directory	로그 파일의 대상 디렉터리를 설정합니다.	/rdsdbdata/log/error
log_disconnections	지속 시간을 포함해 세션 종료를 기록합니다.	–
log_duration	완료된 개별 SQL 문의 지속 시간을 기록합니다.	–
log_error_verbosity	기록된 메시지의 세부 사항을 설정합니다.	–
log_executor_stats	실행기 성능 통계를 서버 로그에 기록합니다.	–
log_file_mode	로그 파일에 대한 파일 권한을 설정합니다.	0644
log_filename	로그 파일의 이름 패턴을 설정합니다.	postgresql.log.%Y-%m-%d-%H%M
logging_collector	하위 프로세스를 시작하여 stderr 출력 및/또는 csvlog를 로그 파일로 캡처합니다.	1
log_hostname	연결 로그에 호스트 이름을 기록합니다.	0
logical_decoding_work_mem	(kB) 디스크로 유출되기 전에 각 내부 재정렬 버퍼에서 이 정도의 메모리를 사용할 수 있습니다.	–
log_line_prefix	각 로그 행에 접두사가 붙은 정보를 제어합니다.	%t:%r:%u@%d:%p]:
log_lock_waits	오랜 잠금 대기 시간을 기록합니다.	–

파라미터 이름	설명	Default
log_min_duration_sample	(밀리초) 문 샘플이 로그되는 최소 실행 시간을 설정합니다. 샘플링은 log_statement_sample_rate에 의해 결정됩니다.	–
log_min_duration_statement	(밀리초) 문이 기록되는 최소 실행 시간을 설정합니다.	–
log_min_error_statement	이 수준 이상으로 오류 원인이 되는 모든 문을 로그합니다.	–
log_min_messages	기록되는 메시지 수준을 설정합니다.	–
log_parameter_max_length	(B) 문을 로그할 때 로그된 파라미터 값을 처음 N바이트로 제한합니다.	–
log_parameter_max_length_on_error	(B) 오류를 보고할 때 로그된 파라미터 값을 처음 N바이트로 제한합니다.	–
log_parser_stats	구문 분석기 성능 통계를 서버 로그에 기록합니다.	–
log_planner_stats	planner 성능 통계를 서버 로그에 기록합니다.	–
log_replication_commands	각 복제 명령을 로그합니다.	–
log_rotation_age	(분) N분 후 자동 로그 파일 교체가 수행됩니다.	60
log_rotation_size	(kB) N킬로바이트 후 자동 로그 파일 교체가 수행됩니다.	100000
log_statement	기록할 문 유형을 설정합니다.	–
log_statement_sample_rate	로그할 log_min_duration_sample을 초과하는 문의 비율입니다.	–
log_statement_stats	누적 성능 통계를 서버 로그에 기록합니다.	–

파라미터 이름	설명	Default
log_temp_files	(kB) 이 킬로바이트 수치보다 큰 임시 파일의 사용을 로그합니다.	–
log_timezone	로그 메시지에 사용할 표준 시간대를 설정합니다.	UTC
log_truncate_on_rotation	로그 순환 중에 동일한 이름의 기존 로그 파일을 잘라냅니다.	0
maintenance_io_concurrency	유지 관리 작업에 사용되는 effective_io_concurrency의 변형입니다.	1
maintenance_work_mem	(kB) 유지 관리 작업에 사용할 최대 메모리를 설정합니다.	GREATEST(DBInstanceClassMemory/63963136*1024, 65536
max_connections	동시에 접속할 수 있는 최대 수를 설정합니다.	LEAST(DBInstanceClassMemory/9531392, 5000
max_files_per_process	서버 프로세스마다 파일을 동시에 열 수 있는 최대 수를 설정합니다.	–
max_locks_per_transaction	하나의 트랜잭션에서 사용할 수 있는 최대 잠금 횟수를 설정합니다.	64
max_parallel_maintenance_workers	유지 관리 작업당 최대 병렬 프로세스 수를 설정합니다.	–
max_parallel_workers	한 번에 활성화할 수 있는 최대 병렬 작업자 수를 설정합니다.	GREATEST($DBInstanceVCPU/2, 8
max_parallel_workers_per_gather	실행기 노드당 최대 병렬 프로세스 수를 설정합니다.	–

파라미터 이름	설명	Default
max_pred_locks_per_page	페이지당 최대 술어 잠금 튜플 수를 설정합니다.	–
max_pred_locks_per_relation	관계당 최대 술어 잠금 페이지 및 튜플 수를 설정합니다.	–
max_pred_locks_per_transaction	하나의 트랜잭션에서 사용할 수 있는 최대 술어(predicate) 잠금 횟수를 설정합니다.	–
max_slot_wal_keep_size	(MB) 복제 슬롯은 실패한 것으로 표시되고 디스크에서 WAL이 이 정도의 공간을 차지하는 경우 삭제 또는 재활용을 위해 세그먼트가 해제됩니다.	–
max_stack_depth	(kB) 최대 스택 깊이(킬로바이트)를 설정합니다.	6144
max_standby_streaming_delay	(밀리초) 상시 대기 서버가 스트리밍된 WAL 데이터를 처리할 때 쿼리 취소까지 걸리는 최대 지연 시간을 설정합니다.	14000
max_worker_processes	최대 동시 작업자 프로세스 수를 설정합니다.	GREATEST($DBInstanceVCPU*2,8
min_dynamic_shared_memory	(MB) 시작 시 예약된 동적 공유 메모리의 양입니다.	–
min_parallel_index_scan_size	(8kB) 병렬 스캔을 위한 최소 인덱스 데이터 양을 설정합니다.	–
min_parallel_table_scan_size	(8kB) 병렬 스캔을 위한 최소 테이블 데이터 양을 설정합니다.	–
old_snapshot_threshold	(분) 스냅샷이 너무 오래되어 스냅샷이 생성된 후 변경된 페이지를 읽을 수 없게 될 때까지의 시간입니다.	–

파라미터 이름	설명	Default
parallel_leader_pa rticipation	수집 및 수집 병합이 하위 계획도 실행할지 여부를 제어합니다.	–
parallel_setup_cost	병렬 쿼리에 대한 작업자 프로세스 시작 비용의 플래너 추정치를 설정합니다.	–
parallel_tuple_cost	작업자에서 마스터 백엔드로 각 튜플(행)을 전달하는 비용의 플래너 추정치를 설정합니다.	–
pgaudit.log	세션 감사 로깅에 의해 로그될 문의 클래스를 지정합니다.	–
pgaudit.log_catalog	문의 모든 관계가 pg_catalog에 있는 경우 세션 로깅을 사용하도록 지정합니다.	–
pgaudit.log_level	로그 항목에 사용할 로그 수준을 지정합니다.	–
pgaudit.log_parameter	감사 로깅에 문과 함께 전달된 파라미터가 포함되도록 지정합니다.	–
pgaudit.log_relation	세션 감사 로깅에서 SELECT 또는 DML 문에서 참조되는 각 관계(TABLE, VIEW 등)에 대해 별도의 로그 항목을 생성해야 하는지 여부를 지정합니다.	–
pgaudit.log_statem ent_once	로깅에 문/하위 문 조합에 대한 첫 번째 로그 항목이 있는 문 텍스트 및 파라미터를 포함할지 아니면 모든 항목이 있는 문 텍스트 및 파라미터를 포함할지를 지정합니다.	–
pgaudit.role	객체 감사 로깅에 사용할 마스터 역할을 지정합니다.	–
pg_bigm.enable_rec heck	전체 텍스트 검색의 내부 프로세스인 재검사를 수행할지 여부를 지정합니다.	켜짐

파라미터 이름	설명	Default
pg_bigm.gin_key_limit	전체 텍스트 검색에 사용할 검색 키워드의 최대 2그램 수를 지정합니다.	0
pg_bigm.last_update	pg_bigm 모듈의 마지막 업데이트 날짜를 보고합니다.	2013.11.22
pg_bigm.similarity_limit	유사성 검색에 사용되는 최소 임계값을 지정합니다.	0.3
pg_hint_plan.debug_print	힌트 구문 분석 결과를 로그합니다.	–
pg_hint_plan.enable_hint	플래너가 쿼리 앞의 힌트 설명에 지정된 계획을 사용하도록 합니다.	–
pg_hint_plan.enable_hint_table	테이블 조회를 사용하여 플래너가 힌트를 얻지 못하도록 합니다.	–
pg_hint_plan.message_level	디버그 메시지의 메시지 수준입니다.	–
pg_hint_plan.parse_messages	구문 분석 오류의 메시지 수준입니다.	–
pglogical.batch_inserts	가능한 경우 배치 삽입	–
pglogical.conflict_log_level	해결된 충돌을 로깅하는 데 사용되는 로그 수준을 설정합니다.	–
pglogical.conflict_resolution	해결 가능한 충돌에 대한 충돌 해결에 사용되는 방법을 설정합니다.	–
pglogical.extra_connection_options	모든 피어 노드 연결에 추가할 연결 옵션	–
pglogical.synchronous_commit	pglogical 특정 동기 커밋 값입니다.	–

파라미터 이름	설명	Default
pglogical.use_spi	변경 사항 적용에 하위 수준 API 대신 SPI 사용	–
pg_similarity.block_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.block_threshold	블록 유사성 함수에 사용되는 임계값을 설정합니다.	–
pg_similarity.block_tokenizer	블록 유사성 함수의 토크나이저를 설정합니다.	–
pg_similarity.cosine_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.cosine_threshold	코사인 유사성 함수에 사용되는 임계값을 설정합니다.	–
pg_similarity.cosine_tokenizer	코사인 유사성 함수의 토크나이저를 설정합니다.	–
pg_similarity.dice_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.dice_threshold	주사위 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.dice_tokenizer	주사위 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.euclidean_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.euclidean_threshold	유클리드 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.euclidean_tokenizer	유클리드 유사성 측정에 대한 토크나이저를 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.hamming_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.hamming_threshold	블록 유사성 지수에 사용되는 임계값을 설정합니다.	–
pg_similarity.jaccard_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jaccard_threshold	자카드 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.jaccard_tokenizer	자카드 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.jaro_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jaro_threshold	자로 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.jarowinkler_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.jarowinkler_threshold	자로윙클러 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.levenshtein_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.levenshtein_threshold	레벤슈타인 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.matching_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.matching_threshold	일치 계수 측정에 사용되는 임계값을 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.matching_tokenizer	일치 계수 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.mongeelkan_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.mongeelkan_threshold	몽게-엘칸 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.mongeelkan_tokenizer	몽게-엘칸 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.nw_gap_penalty	니들만 브니쉬 유사성 측정에 사용되는 갭 페널티를 설정합니다.	–
pg_similarity.nw_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.nw_threshold	니들레만-운쉬 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.overlap_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.overlap_threshold	겹침 계수 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.overlap_tokenizer	겹침 계수 유사성 측정에 대한 토크나이저를 설정합니다.	–
pg_similarity.qgram_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.qgram_threshold	Q-그램 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.qgram_tokenizer	Q-그램 측정에 대한 토크나이저를 설정합니다.	–

파라미터 이름	설명	Default
pg_similarity.swg_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.swg_threshold	스미스-워터맨-고토 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_similarity.sw_is_normalized	결과 값이 정규화되는지 여부를 설정합니다.	–
pg_similarity.sw_threshold	스미스-워터맨 유사성 측정에 사용되는 임계값을 설정합니다.	–
pg_stat_statements.max	pg_stat_statements에서 추적하는 최대 문 수를 설정합니다.	–
pg_stat_statements.save	서버 종료 시 pg_stat_statements 통계를 저장합니다.	–
pg_stat_statements.track	pg_stat_statements로 추적할 문을 선택합니다.	–
pg_stat_statements.track_planning	pg_stat_statements로 계획 기간을 추적할지 여부를 선택합니다.	–
pg_stat_statements.track_utility	pg_stat_statements로 유틸리티 명령을 추적할지 여부를 선택합니다.	–
postgis.gdal_enabled_drivers	Postgres 9.3.5 이상에서 PostGIS와 함께 사용되는 GDAL 드라이버를 사용 설정하거나 사용 중지합니다.	ENABLE_ALL
quote_all_identifiers	SQL 조각 생성 시 모든 식별자에 인용 부호를 추가합니다.	–
random_page_cost	비순차적으로 가져온 디스크 페이지에 대한 플래너의 예상 비용을 설정합니다.	–

파라미터 이름	설명	Default
rds.enable_memory_management	여유 메모리 부족으로 인한 안정성 문제와 데이터베이스 재시작을 방지하는 Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5 및 상위 버전의 메모리 관리 기능을 개선합니다. 자세한 내용은 Aurora PostgreSQL의 향상된 메모리 관리 섹션을 참조하세요.	True
rds.force_admin_logging_level	고객 데이터베이스의 RDS 관리자 사용자 작업에 대한 로그 메시지를 참조하세요.	–
rds.log_retention_period	Amazon RDS가 N분보다 오래된 PostgreSQL 로그를 삭제합니다.	4320
rds.memory_allocation_guard	여유 메모리 부족으로 인한 안정성 문제와 데이터베이스 재시작을 방지하는 Aurora PostgreSQL 11.21, 12.16, 13.12, 14.9, 15.4 및 하위 버전의 메모리 관리 기능을 개선합니다. 자세한 내용은 Aurora PostgreSQL의 향상된 메모리 관리 섹션을 참조하세요.	False
rds.pg_stat_ramdisk_size	통계 램디스크의 크기(MB)입니다. 값이 0이 아니면 램디스크가 설정됩니다.	0
rds.rds_superuser_reserved_connections	rds_superuser용으로 예약된 연결 슬롯 수를 설정합니다. 이 파라미터는 버전 15 이상에서만 사용할 수 있습니다. 자세한 내용은 PostgreSQL 설명서의 reserved connections를 참조하세요.	2
rds.superuser_variables	rds_superuser 수정 문을 승격시키는 수퍼 사용자 전용 변수 목록입니다.	session_replication_role
remove_temp_files_after_crash	백엔드 충돌 후 임시 파일을 제거합니다.	0
restart_after_crash	백엔드 충돌 후 서버를 다시 초기화합니다.	–
row_security	행 보안을 사용 설정합니다.	–

파라미터 이름	설명	Default
search_path	스키마로 한정되지 않은 이름의 스키마 검색 순서를 설정합니다.	–
seq_page_cost	순차적으로 가져온 디스크 페이지에 대한 플래너의 예상 비용을 설정합니다.	–
session_replication_role	트리거 및 다시 쓰기 규칙에 대한 세션 동작을 설정합니다.	–
shared_buffers	(8kB) 서버에서 사용하는 공유 메모리 버퍼 수를 설정합니다.	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	서버에 미리 로드할 공유 라이브러리를 나열합니다.	pg_stat_statements
ssl_ca_file	SSL 서버 권한 파일의 위치입니다.	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 서버 인증서 파일의 위치입니다.	/rdsdbdata/rds-metadata/server-cert.pem
ssl_crl_dir	SSL 인증서 취소 목록 디렉토리의 위치입니다.	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL 서버 프라이빗 키 파일의 위치입니다.	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	... 문자열에서 백슬래시가 리터럴로 처리됩니다.	–
statement_timeout	(밀리초) 모든 문의 최대 허용 기간을 설정합니다.	–
stats_temp_directory	임시 통계 파일을 지정된 디렉터리에 씁니다.	/rdsdbdata/db/pg_stat_tmp

파라미터 이름	설명	Default
superuser_reserved_connections	슈퍼 사용자용으로 예약된 연결 슬롯 수를 설정합니다.	3
synchronize_seqscans	동기 방식의 순차적 스캔을 사용 설정합니다.	–
tcp_keepalives_count	TCP keepalive의 최대 재전송 횟수를 지정합니다.	–
tcp_keepalives_idle	(초) TCP keepalive의 실행 간격을 지정합니다.	–
tcp_keepalives_interval	(초) TCP keepalive의 재전송 간격을 지정합니다.	–
temp_buffers	(8kB) 각 세션에서 사용하는 임시 버퍼의 최대 수를 설정합니다.	–
temp_file_limit	명시적 임시 테이블에 사용되는 공간을 제외하고 주어진 PostgreSQL 프로세스가 임시 파일에 사용할 수 있는 총 디스크 공간(KB)을 제한합니다.	-1
temp_tablespaces	임시 테이블 및 정렬 파일에 사용할 테이블스페이스를 설정합니다.	–
track_activities	명령 실행에 대한 정보를 수집합니다.	–
track_activity_query_size	pg_stat_activity.current_query에 예약되는 크기(바이트)를 설정합니다.	4096
track_counts	데이터베이스 작업에 관한 통계를 수집합니다.	–
track_functions	데이터베이스 작업에 관한 함수 수준 통계를 수집합니다.	pl
track_io_timing	데이터베이스 I/O 작업에 대한 시간 통계를 수집합니다.	1

파라미터 이름	설명	Default
transform_–_equals	expr=–을 expr IS –로 처리합니다.	–
update_process_title	프로세스 제목을 업데이트하여 활성 SQL 명령을 표시합니다.	–
wal_receiver_status_interval	(초) WAL 수신기 상태 보고 간 최대 간격을 프라이머리로 설정합니다.	–
work_mem	(kB) 쿼리 작업 공간에 사용할 최대 메모리를 설정합니다.	–
xmlbinary	XML에서 바이너리 값의 인코딩 방식을 설정합니다.	–
xmloption	암시적 구문 분석 및 직렬화 작업에서 XML 데이터를 문서 또는 내용 조각으로 간주할지 여부를 설정합니다.	–

Amazon Aurora PostgreSQL 대기 이벤트

다음은 Aurora PostgreSQL의 일반적인 대기 이벤트입니다. 대기 이벤트 및 Aurora PostgreSQL 클러스터 튜닝에 대해 자세히 알아보려면 섹션을 참조하세요.Aurora PostgreSQL의 대기 이벤트를 사용한 튜닝.

Activity:ArchiverMain

아카이버 프로세스가 활동을 기다리고 있습니다.

Activity:AutoVacuumMain

autovacuum 시작 관리자 프로세스가 활동을 기다리고 있습니다.

Activity:BgWriterHibernate

백그라운드 라이터 프로세스가 활동을 기다리는 동안 최대 절전 모드로 전환됩니다.

Activity:BgWriterMain

백그라운드 라이터 프로세스가 활동을 기다리고 있습니다.

Activity:CheckpointerMain

체크포인터 프로세스가 활동을 기다리고 있습니다.

Activity:LogicalApplyMain

논리적 복제 적용 프로세스가 활동을 기다리고 있습니다.

Activity:LogicalLauncherMain

논리적 복제 시작 관리자 프로세스가 활동을 기다리고 있습니다.

Activity:PgStatMain

통계 수집기 프로세스가 활동을 기다리고 있습니다.

Activity:RecoveryWalAll

프로세스가 복구 시 스트림에서 미리 쓰기 로그(WAL)를 기다리고 있습니다.

Activity:RecoveryWalStream

스타트업 프로세스는 스트리밍 복구 중에 미리 쓰기 로그(WAL)가 도착할 때까지 대기 중입니다.

Activity:SysLoggerMain

syslogger 프로세스가 활동을 기다리고 있습니다.

Activity:WalReceiverMain

미리 쓰기 로그(WAL) 수신기 프로세스가 활동을 기다리고 있습니다.

Activity:WalSenderMain

미리 쓰기 로그(WAL) 수신기 프로세스가 활동을 기다리고 있습니다.

Activity:WalWriterMain

미리 쓰기 로그(WAL) 라이터 프로세스가 활동을 기다리고 있습니다.

BufferPin:BufferPin

버퍼에서 독점 핀을 획득하기 위해 프로세스가 기다리고 있습니다.

Client:GSSOpenServer

일반 보안 서비스 애플리케이션 인터페이스(GSSAPI) 세션을 설정하는 동안 프로세스가 클라이언트에서 데이터를 읽기를 기다리고 있습니다.

Client:ClientRead

백엔드 프로세스가 PostgreSQL 클라이언트에서 데이터를 수신하기 위해 대기 중입니다. 자세한 내용은 Client:ClientRead 섹션을 참조하세요.

Client:ClientWrite

백엔드 프로세스가 PostgreSQL 클라이언트로 더 많은 데이터를 발신하기 위해 대기 중입니다. 자세한 내용은 Client:ClientWrite 섹션을 참조하세요.

Client:LibPQWalReceiverConnect

프로세스가 원격 서버에 연결을 생성하기 위해 미리 쓰기 로그 수신기(WAL)에서 대기 중입니다.

Client:LibPQWalReceiverReceive

프로세스가 원격 서버에서 데이터를 수신하기 위해 미리 쓰기 로그 수신기(WAL)에서 대기 중입니다.

Client:SSLOpenServer

프로세스가 연결을 시도하는 동안 보안 소켓 레이어(SSL)를 기다리고 있습니다.

Client:WalReceiverWaitStart

스타타트업 프로세스가 스트리밍 복제를 위한 초기 데이터를 전송하기를 기다리고 있습니다.

Client:WalSenderWaitForWAL

프로세스가 WAL 발신자 프로세스에서 미리 쓰기 로그(WAL)가 플러시될 때까지 기다리고 있습니다.

Client:WalSenderWriteData

WAL 발신자 프로세스에서 미리 쓰기 로그 (WAL) 수신자의 응답을 처리할 때 프로세스가 모든 활동을 기다리고 있습니다.

CPU

백엔드 프로세스가 CPU에서 활성 상태이거나 CPU를 기다리고 있습니다. 자세한 내용은 CPU 섹션을 참조하세요.

Extension:extension

백엔드 프로세스가 확장 또는 모듈에 의해 정의된 조건을 기다리고 있습니다.

IO:AuroraEnhancedLogicalWALRead

백엔드 프로세스는 변경 데이터 캡처(CDC) 볼륨에서 로그 레코드를 가져오는 것입니다.

IO:AuroraOptimizedReadsCacheRead

공유 메모리에서 페이지를 사용할 수 없기 때문에 프로세스가 최적화된 읽기 계층형 캐시에서 읽기를 기다리고 있습니다.

IO:AuroraOptimizedReadsCacheSegmentTruncate

프로세스가 최적화된 읽기 계층형 캐시 세그먼트 파일이 잘릴 때까지 기다리고 있습니다.

IO:AuroraOptimizedReadsCacheWrite

백그라운드 라이터 프로세스가 최적화된 읽기 계층형 캐시에 쓰기를 기다리고 있습니다.

IO:AuroraStorageLogAllocate

세션이 메타데이터를 할당하고 트랜잭션 로그 쓰기를 준비하고 있습니다.

IO:BufFileRead

작업에 작업 메모리 파라미터로 정의된 양보다 많은 메모리가 필요한 경우 엔진은 디스크에 임시 파일을 생성합니다. 이 대기 이벤트는 작업 중 임시 파일에서 읽을 때 발생합니다. 자세한 내용은 IO:BufFileRead 및 IO:BufFileWrite 섹션을 참조하세요.

IO:BufFileWrite

작업에 작업 메모리 파라미터로 정의된 양보다 많은 메모리가 필요한 경우 엔진은 디스크에 임시 파일을 생성합니다. 이 대기 이벤트는 연산자가 임시 파일에서 쓸 때 발생합니다. 자세한 내용은 IO:BufFileRead 및 IO:BufFileWrite 섹션을 참조하세요.

IO:ControlFileRead

프로세스가 `pg_control` 파일에서 읽기를 기다리고 있습니다.

IO:ControlFileSync

프로세스가 `pg_control` 파일에서 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:ControlFileSyncUpdate

프로세스가 `pg_control` 파일이 내구성 있는 스토리지에 도달하도록 업데이트를 기다리고 있습니다.

IO:ControlFileWrite

프로세스가 `pg_control` 파일에서 쓰기를 기다리고 있습니다.

IO:ControlFileWriteUpdate

프로세스가 `pg_control` 파일 업데이트를 위해 쓰기를 기다리고 있습니다.

IO:CopyFileRead

파일 복사 작업 중에 프로세스가 읽기를 기다리고 있습니다.

IO:CopyFileWrite

파일 복사 작업 중에 프로세스가 쓰기를 기다리고 있습니다.

IO:DataFileExtend

프로세스가 관계식 데이터 파일이 확장될 때까지 기다리고 있습니다.

IO:DataFileFlush

프로세스가 관계식 데이터 파일이 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:DataFileImmediateSync

프로세스가 관계식 데이터 파일이 내구성 있는 스토리지에 즉시 동기화될 때까지 기다리고 있습니다.

IO:DataFilePrefetch

프로세스가 관계식 데이터 파일에서 비동기 프리페치를 기다리고 있습니다.

IO:DataFileSync

프로세스가 내구성 있는 스토리지에 도달하기 위해 관계 데이터 파일의 변경 사항을 기다리고 있습니다.

IO:DataFileRead

백엔드 프로세스가 공유 버퍼에서 페이지를 찾으려고 시도했지만 찾지 못했으므로 저장소에서 읽었습니다. 자세한 내용은 IO:DataFileRead 섹션을 참조하세요.

IO:DataFileTruncate

프로세스가 관계식 데이터 파일이 잘릴 때까지 기다리고 있습니다.

IO:DataFileWrite

프로세스가 관계식 데이터 파일에 쓰기를 기다리고 있습니다.

IO:DSMFillZeroWrite

프로세스가 동적 공유 메모리 백업 파일에 0바이트를 쓰기를 기다리고 있습니다.

IO:LockFileAddToDataDirRead

프로세스가 데이터 디렉토리 잠금 파일에 행을 추가하는 동안 읽기를 기다리고 있습니다.

IO:LockFileAddToDataDirSync

프로세스가 데이터 디렉토리 잠금 파일에 줄을 추가하는 동안 데이터가 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:LockFileAddToDataDirWrite

프로세스가 데이터 디렉토리 잠금 파일에 행을 추가하는 동안 쓰기를 기다리고 있습니다.

IO:LockFileCreateRead

프로세스가 데이터 디렉토리 잠금 파일을 만드는 동안 읽기를 기다리고 있습니다.

IO:LockFileCreateSync

프로세스가 데이터 디렉토리 잠금 파일을 만드는 동안 데이터가 내구성 있는 스토리지에 도달하기를 기다리고 있습니다

IO:LockFileCreateWrite

프로세스가 데이터 디렉토리 잠금 파일을 만드는 동안 쓰기를 기다리고 있습니다.

IO:LockFileReCheckDataDirRead

프로세스가 데이터 디렉토리 잠금 파일을 다시 확인하는 동안 읽기를 기다리고 있습니다.

IO:LogicalRewriteCheckpointSync

프로세스가 체크포인트 중에 내구성 있는 스토리지에 도달하기 위해 논리적 재쓰기 매핑을 기다리고 있습니다.

IO:LogicalRewriteMappingSync

프로세스가 논리적 재작성 중에 데이터가 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:LogicalRewriteMappingWrite

프로세스가 논리적 재작성 중에 매핑 데이터의 쓰기를 기다리고 있습니다.

IO:LogicalRewriteSync

프로세스가 논리적 재작성 매핑이 내구성 있는 스토리지에 도착할 때까지 기다리고 있습니다.

IO:LogicalRewriteTruncate

프로세스가 논리적 재작성 중에 매핑 데이터의 절단을 기다리고 있습니다.

IO:LogicalRewriteWrite

프로세스가 논리적 재작성 매핑 쓰기를 기다리고 있습니다.

IO:RelationMapRead

프로세스가 관계식 맵 파일의 읽기를 기다리고 있습니다.

IO:RelationMapSync

프로세스가 관계식 맵 파일이 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:RelationMapWrite

프로세스가 관계식 맵 파일에 대한 쓰기를 기다리고 있습니다.

IO:ReorderBufferRead

재정렬 버퍼 관리 중에 프로세스가 읽기를 기다리고 있습니다.

IO:ReorderBufferWrite

재정렬 버퍼 관리 중에 프로세스가 읽기를 기다리고 있습니다.

IO:ReorderLogicalMappingRead

재정렬 버퍼 관리 중에 프로세스가 읽기를 기다리고 있습니다.

IO:ReplicationSlotRead

프로세스가 복제 슬롯 제어 파일에서 읽기를 기다리고 있습니다.

IO:ReplicationSlotRestoreSync

프로세스가 복제 슬롯 제어 파일을 메모리에 복원하는 동안 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:ReplicationSlotSync

프로세스가 복제 슬롯 제어 파일이 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:ReplicationSlotWrite

프로세스가 복제 슬롯 제어 파일에 쓰기를 기다리고 있습니다.

IO:SLRUFlushSync

프로세스가 체크포인트 또는 데이터베이스 종료 중 가장 오래전에 사용된 단순(SLRU) 데이터가 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:SLRURead

프로세스가 가장 오래전에 사용된 단순(SLRU) 페이지의 읽기를 기다리고 있습니다.

IO:SLRUSync

프로세스는 페이지 쓰기 후 가장 오래전에 사용된 단순(SLRU) 데이터가 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:SLRUWrite

프로세스가 가장 오래전에 사용된 단순(SLRU) 페이지의 쓰기를 기다리고 있습니다.

IO:SnapbuildRead

프로세스가 직렬화된 기록 카탈로그 스냅샷의 읽기를 기다리고 있습니다.

IO:SnapbuildSync

프로세스가 직렬화된 기록 카탈로그 스냅샷이 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:SnapbuildWrite

프로세스가 직렬화된 기록 카탈로그 스냅샷의 쓰기를 기다리고 있습니다.

IO:TimelineHistoryFileSync

프로세스가 스트리밍 복제를 통해 수신된 타임라인 기록 파일이 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:TimelineHistoryFileWrite

프로세스가 스트리밍 복제를 통해 수신된 타임라인 기록 파일의 쓰기를 기다리고 있습니다.

IO:TimelineHistoryRead

프로세스가 타임라인 히스토리 파일의 읽기를 기다리고 있습니다.

IO:TimelineHistorySync

프로세스가 새로 생성된 타임라인 기록 파일이 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:TimelineHistoryWrite

프로세스가 새로 생성된 타임라인 기록 파일의 쓰기를 기다리고 있습니다.

IO:TwophaseFileRead

프로세스가 2단계 상태 파일의 읽기를 기다리고 있습니다.

IO:TwophaseFileSync

프로세스가 2단계 상태 파일이 내구성 있는 스토리지에 도달하기를 기다리고 있습니다.

IO:TwophaseFileWrite

프로세스가 2단계 상태 파일의 쓰기를 기다리고 있습니다.

IO:WALBootstrapSync

프로세스가 부트스트래핑 중에 미리 쓰기 로그(WAL)가 내구성 있는 스토리지에 도달할 때까지 대기 중입니다.

IO:WALBootstrapWrite

프로세스가 부트스트래핑 중에 미리 쓰기 로그(WAL) 페이지의 쓰기를 대기 중입니다.

IO:WALCopyRead

프로세스는 기존 세그먼트를 복사하여 새 미리 쓰기 로그(WAL) 세그먼트를 만들 때 읽기를 기다리고 있습니다.

IO:WALCopySync

프로세스는 기존 세그먼트를 복사하여 생성한 새 미리 쓰기 로그(WAL)가 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:WALCopyWrite

프로세스는 기존 세그먼트를 복사하여 새 미리 쓰기 로그(WAL) 세그먼트를 만들 때 쓰기를 기다리고 있습니다.

IO:WALInitSync

프로세스가 새로 초기화된 미리 쓰기 로그(WAL) 파일이 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:WALInitWrite

프로세스가 새 미리 쓰기 로그(WAL) 파일을 초기화하는 동안 쓰기를 기다리고 있습니다.

IO:WALRead

프로세스가 미리 쓰기 로그(WAL) 파일에서 읽기를 기다리고 있습니다.

IO:WALSenderTimelineHistoryRead

프로세스가 WAL 발신자 타임라인 명령 중에 타임라인 기록 파일에서 읽기를 기다리고 있습니다.

IO:WALSync

프로세스가 미리 쓰기 로그(WAL) 파일이 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:WALSyncMethodAssign

프로세스는 새로운 미리 쓰기 로그(WAL) 동기화 방법을 할당하면서 데이터가 내구성 있는 스토리지에 도달할 때까지 기다리고 있습니다.

IO:WALWrite

프로세스가 미리 쓰기 로그(WAL) 파일에서 읽기를 기다리고 있습니다.

IO:XactSync

백엔드 프로세스는 Aurora 스토리지 하위 시스템이 일반 트랜잭션의 커밋 또는 준비된 트랜잭션의 커밋 또는 롤백을 확인할 때까지 기다리고 있습니다. 자세한 내용은 IO:XactSync 섹션을 참조하세요.

IPC:AuroraLogicalSchemaUpdate

두 백엔드 프로세스가 스키마 캐시에 동일한 항목을 삽입하려고 시도하고 있습니다. 한 프로세스는 계속되고 다른 프로세스는 완료될 때까지 기다립니다.

IPC:AuroraOptimizedReadsCacheWriteStop

프로세스는 백그라운드 라이터가 최적화된 읽기 계층형 캐시에 쓰기를 중지하기를 기다리고 있습니다.

IPC:BackupWaitWalArchive

프로세스가 백업이 성공적으로 아카이빙되는 데 필요한 미리 쓰기 로그(WAL) 파일을 기다리고 있습니다.

IPC:BgWorkerShutdown

프로세스가 백그라운드 작업자가 종료될 때까지 기다리고 있습니다.

IPC:BgWorkerStartup

프로세스가 백그라운드 작업자가 시작할 때까지 기다리고 있습니다.

IPC:BtreePage

프로세스가 병렬 B-트리 스캔을 계속 사용할 수 있도록 하는 데 필요한 페이지 번호를 기다리고 있습니다.

IPC:CheckpointDone

프로세스가 체크포인트가 완료될 때까지 기다리고 있습니다.

IPC:CheckpointStart

프로세스가 체크포인트가 시작될 때까지 기다리고 있습니다.

IPC:ClogGroupUpdate

프로세스가 그룹 리더가 트랜잭션이 끝날 때 트랜잭션 상태를 업데이트하기를 기다리고 있습니다.

IPC:DamRecordTxAck

백엔드 프로세스에서 데이터베이스 활동 스트림 이벤트를 생성했으며 이벤트가 지속될 때까지 기다리고 있습니다. 자세한 내용은 IPC:DamRecordTxAck 섹션을 참조하세요.

IPC:ExecuteGather

프로세스가 Gather 계획 노드를 실행하는 동안 자식 프로세스의 활동을 기다리고 있습니다.

IPC:Hash/Batch/Allocating

프로세스가 선택된 병렬 해시 참여자가 해시 테이블을 할당하기를 기다리고 있습니다.

IPC:Hash/Batch/Electing

프로세스가 해시 테이블을 할당하기 위해 병렬 해시 참여자를 선택합니다.

IPC:Hash/Batch/Loading

프로세스가 다른 병렬 해시 참여자가 해시 테이블 로딩을 마칠 때까지 기다리고 있습니다.

IPC:Hash/Batch/Allocating

프로세스가 선택된 병렬 해시 참여자가 초기 해시 테이블을 할당하기를 기다리고 있습니다.

IPC:Hash/Build/Electing

프로세스가 초기 해시 테이블을 할당하기 위해 병렬 해시 참여자를 선택합니다.

IPC:Hash/Build/HashingInner

프로세스가 다른 병렬 해시 참여자가 내부 관계 해싱을 완료할 때까지 기다리고 있습니다.

IPC:Hash/Build/HashingOuter

프로세스가 다른 병렬 해시 참여자가 외부 관계 파티셔닝을 마칠 때까지 기다리고 있습니다.

IPC:해시/성장 배치/할당

프로세스는 선출된 병렬 해시 참여자가 더 많은 배치를 할당하기를 기다리고 있습니다.

IPC:Hash/GrowBatches/Deciding

프로세스에서 병렬 해시 참여자를 선택하여 향후 배치 증가를 결정합니다.

IPC:Hash/GrowBatches/Electing

프로세스에서 병렬 해시 참여자를 선택하여 더 많은 배치를 할당합니다.

IPC:Hash/GrowBatches/Finishing

프로세스는 선출된 병렬 해시 참여자가 향후 배치 증가를 결정할 때까지 기다리고 있습니다.

IPC:Hash/GrowBatches/Repartitioning

프로세스가 다른 병렬 해시 참여자가 다시 파티셔닝을 완료할 때까지 기다리고 있습니다.

IPC:Hash/GrowBuckets/Allocating

프로세스가 선택된 병렬 해시 참여자가 더 많은 버킷 할당을 완료할 때까지 기다리고 있습니다.

IPC:Hash/GrowBuckets/Electing

프로세스가 더 많은 버킷을 할당하기 위해 병렬 해시 참여자를 선택합니다.

IPC:Hash/GrowBuckets/Reinserting

프로세스가 다른 병렬 해시 참가자가 새 버킷에 튜플 삽입을 마칠 때까지 기다리고 있습니다.

IPC:HashBatchAllocate

프로세스가 선택된 병렬 해시 참여자가 해시 테이블을 할당하기를 기다리고 있습니다.

IPC:HashBatchElect

프로세스가 해시 테이블을 할당하기 위해 병렬 해시 참여자를 선택하기를 기다리고 있습니다.

IPC:HashBatchLoad

프로세스가 다른 병렬 해시 참여자가 해시 테이블 로딩을 마칠 때까지 기다리고 있습니다.

IPC:HashBuildAllocate

프로세스가 선택된 병렬 해시 참여자가 초기 해시 테이블을 할당하기를 기다리고 있습니다.

IPC:HashBuildElect

프로세스가 초기 해시 테이블을 할당하기 위해 병렬 해시 참여자를 선택하기를 기다리고 있습니다.

IPC:HashBuildHashInner

프로세스가 다른 병렬 해시 참여자가 내부 관계 해싱을 완료할 때까지 기다리고 있습니다.

IPC:'HashBuildHashOuter

프로세스가 다른 병렬 해시 참여자가 외부 관계 파티셔닝을 마칠 때까지 기다리고 있습니다.

IPC:HashGrowBatchesAllocate

프로세스는 선출된 병렬 해시 참여자가 더 많은 배치를 할당하기를 기다리고 있습니다.

IPC:'HashGrowBatchesDecide

프로세스가 병렬 해시 참가자를 선출하여 향후 배치 증가를 결정하기를 기다리고 있습니다.

IPC:HashGrowBatchesElect

프로세스가 더 많은 배치를 할당하기 위해 병렬 해시 참여자를 선택하기를 기다리고 있습니다.

IPC:HashGrowBatchesFinish

프로세스는 선출된 병렬 해시 참여자가 향후 배치 증가를 결정할 때까지 기다리고 있습니다.

IPC:HashGrowBatchesRepartition

프로세스가 다른 병렬 해시 참여자가 다시 파티셔닝을 완료할 때까지 기다리고 있습니다.

IPC:HashGrowBucketsAllocate

프로세스가 선택된 병렬 해시 참여자가 더 많은 버킷 할당을 완료할 때까지 기다리고 있습니다.

IPC:HashGrowBucketsElect

프로세스가 더 많은 배치를 할당하기 위해 병렬 해시 참여자를 선택하기를 기다리고 있습니다.

IPC:HashGrowBucketsReinsert

프로세스가 다른 병렬 해시 참가자가 새 버킷에 튜플 삽입을 마칠 때까지 기다리고 있습니다.

IPC:LogicalSyncData

프로세스가 논리적 복제 원격 서버가 초기 테이블 동기화를 위해 데이터를 보낼 때까지 대기 중입니다.

IPC:LogicalSyncStateChange

프로세스가 논리적 복제 원격 서버가 상태를 변경하기를 기다리고 있습니다.

IPC:MessageQueueInternal

프로세스가 다른 프로세스가 공유 메시지 대기열에 연결될 때까지 기다리고 있습니다.

IPC:MessageQueuePutMessage

프로세스가 공유 메시지 대기열에 프로토콜 메시지를 쓰기 위해 대기 중입니다.

IPC:MessageQueueReceive

프로세스가 공유 메시지 대기열에서 바이트를 수신하기 위해 대기 중입니다.

IPC:MessageQueueSend

프로세스가 공유 메시지 대기열로 바이트를 전송하기 위해 대기 중입니다.

IPC:ParallelBitmapScan

프로세스가 병렬 비트맵 스캔이 초기화되기를 기다리고 있습니다.

IPC:ParallelCreateIndexScan

프로세스가 병렬 CREATE INDEX 작업자가 힙 스캔을 완료할 때까지 기다리고 있습니다.

IPC:ParallelFinish

프로세스가 병렬 작업자가 컴퓨팅을 완료할 때까지 기다리고 있습니다.

IPC:ProcArrayGroupUpdate

그룹 리더가 트랜잭션 종료 시 트랜잭션 ID를 지울 때까지 프로세스가 대기 중입니다.

IPC:ProcSignalBarrier

프로세스가 모든 백엔드에서 배리어 이벤트가 처리되기를 기다리고 있습니다.

IPC:Promote

프로세스가 대기 프로모션을 기다리고 있습니다.

IPC:RecoveryConflictSnapshot

프로세스가 베큠 클린업을 위해 복구 충돌 해결을 기다리고 있습니다.

IPC:RecoveryConflictTablespace

프로세스가 테이블스페이스를 삭제하기 위한 복구 충돌 해결을 기다리고 있습니다.

IPC:RecoveryPause

프로세스가 복구가 재개될 때까지 기다리고 있습니다.

IPC:ReplicationOriginDrop

프로세스가 복제 원본이 비활성화되어 삭제할 수 있도록 기다리고 있습니다.

IPC:ReplicationSlotDrop

프로세스가 복제 슬롯이 비활성화되어 삭제할 수 있도록 기다리고 있습니다.

IPC:SafeSnapshot

프로세스가 읽기 전용 연기 가능 트랜잭션에 대해 유효한 스냅샷을 얻기 위해 기다리고 있습니다.

IPC:SyncRep

프로세스가 동기식 복제 중에 원격 서버로부터의 확인을 기다리고 있습니다.

IPC:XactGroupUpdate

그룹 리더가 트랜잭션 종료 시 트랜잭션 상태를 업데이트할 때까지 프로세스가 대기 중입니다.

Lock:advisory

백엔드 프로세스에서 권고 잠금을 요청하고 기다리고 있습니다. 자세한 내용은 Lock:advisory 섹션을 참조하세요.

Lock:extend

백엔드 프로세스는 관계식을 확장할 수 있도록 잠금이 해제되기를 기다리고 있습니다. 한 번에 하나의 백엔드 프로세스만 관계를 확장할 수 있기 때문에 이 잠금이 필요합니다. 자세한 내용은 Lock:extend 섹션을 참조하세요.

Lock:frozenid

프로세스가 `pg_database.datfrozenxid`와 `pg_database.datminmxid`의 업데이트를 위해 대기 중입니다.

Lock:object

프로세스가 비관계 데이터베이스 객체에 대한 잠금을 얻기 위해 기다리고 있습니다.

Lock:page

프로세스가 관계 페이지에서 잠금을 받기 위해 기다리고 있습니다.

Lock:Relation

백엔드 프로세스가 다른 트랜잭션에 의해 잠긴 관계에 대한 잠금을 획득하기 위해 기다리고 있습니다. 자세한 내용은 Lock:Relation 섹션을 참조하세요.

Lock:spectoken

투기적 삽입 잠금을 얻기 위해 프로세스가 기다리고 있습니다.

Lock:speculative token

투기적 삽입 잠금을 얻기 위해 프로세스가 기다리고 있습니다.

Lock:transactionid

트랜잭션이 행 수준 잠금을 대기 중입니다. 자세한 내용은 Lock:transactionid 섹션을 참조하세요.

Lock:tuple

백엔드 프로세스는 튜플에 대한 잠금을 얻기 위해 기다리고 있으며 다른 백엔드 프로세스는 동일한 튜플에서 충돌하는 잠금을 유지합니다. 자세한 내용은 Lock:tuple 섹션을 참조하세요.

Lock:userlock

프로세스가 사용자 잠금을 받기 위해 기다리고 있습니다.

Lock:virtualxid

프로세스가 가상 트랜잭션 ID 잠금을 받기 위해 기다리고 있습니다.

LWLock:AddinShmemInit

프로세스가 공유 메모리에서 확장 프로그램의 공간 할당을 관리하기 위해 기다리고 있습니다.

LWLock:AddinShmemInitLock

프로세스가 공유 메모리의 공간 할당을 관리하기 위해 기다리고 있습니다.

LWLock:async

프로세스가 비동기(알림) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:AsyncCtlLock

프로세스가 공유 알림 상태를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:AsyncQueueLock

프로세스가 알림 메시지를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:AuroraOptimizedReadsCacheMapping

프로세스가 데이터 블록을 최적화된 읽기 계층형 캐시의 페이지와 연결하기 위해 기다리고 있습니다.

LWLock:AutoFile

프로세스가 `postgresql.auto.conf` 파일을 업데이트하기 위해 대기 중입니다.

LWLock:AutoFileLock

프로세스가 `postgresql.auto.conf` 파일을 업데이트하기 위해 대기 중입니다.

LWLock:Autovacuum

프로세스가 autovacuum 작업자의 현재 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:AutovacuumLock

autovacuum 작업자 또는 시작 관리자가 autovacuum 작업자의 현재 상태를 업데이트하거나 읽기를 기다리고 있습니다.

LWLock:AutovacuumSchedule

프로세스가 autovacuum용으로 선택된 테이블에 여전히 베큠이 필요한지 확인하기 위해 기다리고 있습니다.

Lwlock:AutovacuumScheduleLock

프로세스가 베큠을 하기로 선택한 테이블에 여전히 베큠이 필요한지 확인하기 위해 기다리고 있습니다.

LWLock:BackendRandomLock

프로세스가 난수를 생성하기 위해 기다리고 있습니다.

LWLock:BackgroundWorker

프로세스가 백그라운드 작업자 상태를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:BackgroundWorkerLock

프로세스가 백그라운드 작업자 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:BtreeVacuum

프로세스가 B-트리 인덱스에 대한 베큠 관련 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:BtreeVacuumLock

프로세스가 B-트리 인덱스에 대한 베큠 관련 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:buffer_content

백엔드 프로세스가 공유 메모리 버퍼의 내용에 대한 경량 잠금을 획득하기 위해 대기 중입니다. 자세한 내용은 LWLock:buffer_content (BufferContent) 섹션을 참조하세요.

LWLock:buffer_mapping

백엔드 프로세스가 데이터 블록을 공유 버퍼 풀의 버퍼와 연결하기 위해 기다리고 있습니다. 자세한 내용은 LWLock:buffer_mapping 섹션을 참조하세요.

LWLock:BufferIO

백엔드 프로세스에서 페이지를 공유 메모리로 읽으려고 합니다. 프로세스는 다른 프로세스가 해당 페이지에 대한 I/O를 완료할 때까지 기다리고 있습니다. 자세한 내용은 LWLock:BufferIO(IPC:BufferIO) 섹션을 참조하세요.

LWLock:Checkpoint

프로세스가 체크포인트 시작을 기다리고 있습니다.

LWLock:CheckpointLock

프로세스가 체크포인트 수행을 기다리고 있습니다.

LWLock:CheckpointerComm

프로세스가 `fsync` 요청 관리를 기다리고 있습니다.

LWLock:CheckpointerCommLock

프로세스가 `fsync` 요청 관리를 기다리고 있습니다.

LWLock:clog

프로세스가 클록(트랜잭션 상태) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:CLogControlLock

프로세스가 트랜잭션 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:CLogTruncationLock

프로세스가 `txid_status`를 실행하거나 사용 가능한 가장 오래된 트랜잭션 ID를 업데이트 하기 위해 대기 중입니다.

LWLock:commit_timestamp

프로세스가 커밋 타임스탬프 버퍼에서 I/O를 기다리고 있습니다.

LWLock:CommitTs

프로세스가 트랜잭션 커밋 타임스탬프에 대한 마지막 값 세트를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:CommitTsBuffer

프로세스가 커밋 타임스탬프를 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:CommitTsControlLock

프로세스가 트랜잭션 커밋 타임스탬프를 읽거나 업데이트하기를 기다리고 있습니다.

LWLock:CommitTsLock

프로세스가 트랜잭션 타임스탬프에 대한 마지막 값 세트를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:CommitTsSLRU

프로세스가 커밋 타임스탬프에 대해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.

LWLock:ControlFile

프로세스가 `pg_control` 파일의 읽기 또는 업데이트 또는 새 미리 쓰기 로그(WAL) 생성을 위해 대기 중입니다.

LWLock:ControlFileLock

프로세스가 제어 파일을 읽거나 업데이트하거나 새 미리 쓰기 로그(WAL) 파일을 만들기 위해 대기 중입니다.

LWLock:DynamicSharedMemoryControl

프로세스가 동적 공유 메모리 할당 정보를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:DynamicSharedMemoryControlLock

프로세스가 동적 공유 메모리 상태를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:lock_manager

백엔드 프로세스가 백엔드 프로세스에 대한 잠금을 추가하거나 검사하기 위해 기다리고 있습니다. 또는 병렬 쿼리에 사용되는 잠금 그룹에 참여하거나 종료하기를 기다리고 있습니다. 자세한 내용은 LWLock:lock_manager 섹션을 참조하세요.

LWLock:LockFastPath

프로세스가 프로세스의 빠른 경로 잠금 정보를 읽거나 업데이트하기 위해 기다리고 있습니다.

LWLock:LogicalRepWorker

프로세스가 논리적 복제 작업자의 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:LogicalRepWorkerLock

프로세스가 논리적 복제 작업자에 대한 작업이 완료될 때까지 기다리고 있습니다.

LWLock:LogicalSchemaCache

프로세스가 스키마 캐시를 수정했습니다.

LWLock:multixact_member

프로세스가 multixact_member 버퍼에서 I/O를 기다리고 있습니다.

LWLock:multixact_offset

프로세스가 멀티액트 오프셋 버퍼에서 I/O를 기다리고 있습니다.

LWLock:MultiXactGen

프로세스가 공유 멀티액트 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:MultiXactGenLock

프로세스가 공유 멀티액트 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:MultiXactMemberBuffer

프로세스가 멀티액트 멤버를 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다. 자세한 내용은 LWLock:MultiXact 섹션을 참조하세요.

LWLock:MultiXactMemberControlLock

프로세스가 멀티액트 멤버 매핑을 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:MultiXactMemberSLRU

프로세스가 멀티액트 멤버를 위해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다. 자세한 내용은 LWLock:MultiXact 섹션을 참조하세요.

LWLock:MultiXactOffsetBuffer

프로세스가 멀티액트 오프셋을 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다. 자세한 내용은 LWLock:MultiXact 섹션을 참조하세요.

LWLock:MultiXactOffsetControlLock

프로세스가 멀티액트 오프셋 매핑을 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:MultiXactOffsetSLRU

프로세스가 멀티액트 오프셋을 위해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다. 자세한 내용은 LWLock:MultiXact 섹션을 참조하세요.

LWLock:MultiXactTruncation

프로세스가 멀티액트 정보를 읽거나 절단하기 위해 대기 중입니다.

LWLock:MultiXactTruncationLock

프로세스가 멀티액트 정보를 읽거나 절단하기 위해 대기 중입니다.

LWLock:NotifyBuffer

프로세스가 NOTIFY 메시지에 대해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:NotifyQueue

프로세스가 NOTIFY 메시지를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:NotifyQueueTail

프로세스가 NOTIFY 메시지 스토리지에 대한 제한을 업데이트하기 위해 대기 중입니다.

LWLock:NotifyQueueTailLock

프로세스가 알림 메시지 스토리지에 대한 제한을 업데이트하기 위해 대기 중입니다.

LWLock:NotifySLRU

프로세스가 NOTIFY 메시지에 대해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.

LWLock:OidGen

프로세스가 새 객체 ID(OID)를 할당하기 위해 대기 중입니다.

LWLock:OidGenLock

프로세스가 객체 ID(OID)를 할당하거나 지정하기 위해 대기 중입니다.

LWLock:oldserxid

프로세스가 oldserxid 버퍼에서 I/O를 대기 중인 중입니다.

LWLock:OldSerXidLock

프로세스가 충돌하는 직렬화 가능 트랜잭션을 읽거나 기록하기 위해 기다리고 있습니다.

LWLock:OldSnapshotTimeMap

프로세스가 이전 스냅샷 제어 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:OldSnapshotTimeMapLock

프로세스가 이전 스냅샷 제어 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:parallel_append

병렬 추가 계획 실행 중에 프로세스가 다음 하위 계획을 선택하기를 기다리고 있습니다.

LWLock:parallel_hash_join

프로세스가 병렬 해시 계획 실행 중에 메모리 청크 또는 업데이트 카운터를 할당하거나 교환하기 위해 대기 중입니다.

LWLock:parallel_query_dsa

프로세스가 병렬 쿼리에 대한 동적 공유 메모리 할당에 대한 잠금을 기다리고 있습니다.

LWLock:ParallelAppend

프로세스가 병렬 추가 계획 실행 중에 다음 하위 계획을 선택하기를 기다리고 있습니다.

Lwlock:ParallelHashJoin

프로세스가 병렬 해시 조인에 대한 계획 실행 중에 작업자를 동기화하기 위해 대기 중입니다.

Lwlock:ParallelQueryDSA

프로세스가 병렬 쿼리에 대한 동적 공유 메모리 할당을 기다리고 있습니다.

Lwlock:PerSessionDSA

프로세스가 병렬 쿼리에 대한 동적 공유 메모리 할당을 기다리고 있습니다.

Lwlock:PerSessionRecordType

프로세스가 복합 유형에 대한 병렬 쿼리의 정보에 액세스하기 위해 대기 중입니다.

Lwlock:PerSessionRecordTypmod

프로세스가 익명 레코드 유형을 식별하는 유형 수정자에 대한 병렬 쿼리의 정보에 액세스하기 위해 대기 중입니다.

Lwlock:PerXactPredicateList

프로세스가 병렬 쿼리 중에 현재 직렬화 가능 트랜잭션에 의해 유지되는 술어 잠금 목록에 액세스하기 위해 대기 중입니다.

Lwlock:predicate_lock_manager

프로세스가 술어 잠금 정보를 추가하거나 검사하기 위해 대기 중입니다.

Lwlock:PredicateLockManager

프로세스가 직렬화 가능 트랜잭션에서 사용되는 술어 잠금 정보에 액세스하기 위해 대기 중입니다.

Lwlock:proc

프로세스가 빠른 경로 잠금 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:ProcArray

프로세스가 공유 프로세스별 데이터 구조에 액세스하기 위해 대기 중입니다(일반적으로 스냅샷을 가져오거나 세션의 트랜잭션 ID를 보고하기 위함).

LWLock:ProcArrayLock

프로세스가 트랜잭션이 끝날 때 스냅샷을 얻거나 트랜잭션 ID를 지우기를 기다리고 있습니다.

LWLock:RelationMapping

프로세스가 `pg_filenode.map` 파일의 읽기 또는 업데이트를 기다리고 있습니다(특정 시스템 카탈로그의 파일 노드 지정을 추적하는 데 사용됨).

LWLock:RelationMappingLock

프로세스가 카탈로그-파일 노드 매핑을 저장하는 데 사용되는 관계식 맵 파일을 업데이트하기 위해 대기 중입니다.

LWLock:RelCacheInit

프로세스가 `pg_internal.init` 파일의 읽기 또는 업데이트를 기다리고 있습니다(관계식 캐시 초기화 파일).

LWLock:RelCacheInitLock

프로세스가 관계식 캐시 초기화 파일을 읽거나 쓰기를 기다리고 있습니다.

LWLock:replication_origin

프로세스가 복제 진행률을 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:replication_slot_io

프로세스가 복제 슬롯에서 I/O를 대기 중입니다.

LWLock:ReplicationOrigin

프로세스가 복제 원본을 생성, 삭제 또는 사용하기 위해 대기 중입니다.

LWLock:ReplicationOriginLock

프로세스가 복제 원본을 설정, 삭제 또는 사용하기 위해 대기 중입니다.

LWLock:ReplicationOriginState

프로세스가 하나의 복제 원본의 진행 상황을 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:ReplicationSlotAllocation

프로세스가 복제 슬롯을 할당하거나 해제하기 위해 대기 중입니다.

LWLock:ReplicationSlotAllocationLock

프로세스가 복제 슬롯을 할당하거나 해제하기 위해 대기 중입니다.

LWLock:ReplicationSlotControl

프로세스가 복제 슬롯 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:ReplicationSlotControlLock

프로세스가 해당 복제 슬롯 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:ReplicationSlotIO

프로세스가 복제 슬롯에서 I/O를 대기 중입니다.

LWLock:SerialBuffer

프로세스가 직렬화 가능한 트랜잭션 충돌을 위해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:SerializableFinishedList

프로세스가 완료된 직렬화 가능 트랜잭션 목록에 액세스하기 위해 대기 중입니다.

LWLock:SerializableFinishedListLock

프로세스가 완료된 직렬화 가능 트랜잭션 목록에 액세스하기 위해 대기 중입니다.

LWLock:SerializablePredicateList

프로세스가 트랜잭션에 의해 유지되는 술어 잠금 목록에 액세스하기 위해 대기 중입니다.

LWLock:SerializablePredicateLockListLock

프로세스가 직렬화 가능 트랜잭션에 의해 유지되는 잠금 목록에 대한 작업을 수행하기 위해 대기 중입니다.

LWLock:SerializableXactHash

프로세스가 직렬화 가능 트랜잭션에 대한 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:SerializableXactHashLock

프로세스가 직렬화 가능 트랜잭션에 대한 정보를 검색하거나 저장하기 위해 대기 중입니다.

LWLock:SerialSLRU

프로세스가 직렬화 가능한 트랜잭션 충돌을 위해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스스하기 위해 대기 중입니다.

LWLock:SharedTidBitmap

프로세스가 병렬 비트맵 인덱스 스캔 중에 공유 튜플 식별자(TID) 비트맵에 액세스하기 위해 대기 중입니다.

LWLock:SharedTupleStore

프로세스가 병렬 쿼리 중에 공유 튜플 스토어에 액세스하기 위해 대기 중입니다.

LWLock:ShmemIndex

프로세스가 공유 메모리의 공간 할당을 관리하기 위해 대기 중입니다.

LWLock:ShmemIndexLock

프로세스가 공유 메모리의 공간 할당을 관리하기 위해 대기 중입니다.

LWLock:SInvalRead

프로세스가 공유 카탈로그 무효화 대기열에서 메시지를 검색하기 위해 대기 중입니다.

LWLock:SInvalReadLock

프로세스가 공유 무효화 대기열에서 메시지를 검색하거나 제거하기 위해 대기 중입니다.

LWLock:SInvalWrite

프로세스가 공유 카탈로그 무효화 대기열에서 메시지를 추가하기 위해 대기 중입니다.

LWLock:SInvalWriteLock

프로세스가 공유 무효화 대기열에 메시지를 추가하기 위해 대기 중입니다.

LWLock:subtrans

프로세스가 하위 트랜잭션 버퍼의 I/O를 기다리고 있습니다.

LWLock:SubtransBuffer

프로세스가 하위 트랜잭션에 대해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:SubtransControlLock

프로세스가 하위 트랜잭션 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:SubtransSLRU

프로세스가 하위 트랜잭션에 대해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.

LWLock:SyncRep

프로세스가 동기 복제 상태에 대한 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:SyncRepLock

프로세스가 동기 복제에 대한 정보를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:SyncScan

프로세스가 동기화된 테이블 스캔의 시작 위치를 선택하기 위해 대기 중입니다.

LWLock:SyncScanLock

프로세스가 동기화된 스캔을 위해 테이블에서 검사의 시작 위치를 가져오기 위해 대기 중입니다.

LWLock:TablespaceCreate

프로세스가 테이블스페이스를 생성하거나 삭제하기 위해 대기 중입니다.

LWLock:TablespaceCreateLock

프로세스가 해당 테이블스페이스를 생성하거나 삭제하기 위해 대기 중입니다.

LWLock:tbm

프로세스가 트리 비트맵(TBM) 에서 공유 이터레이터 잠금을 기다리고 있습니다.

LWLock:TwoPhaseState

프로세스가 준비된 트랜잭션 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:TwoPhaseStateLock

프로세스가 준비된 트랜잭션 상태를 읽거나 업데이트하기 위해 대기 중입니다.

LWLock:wal_insert

프로세스가 미리 쓰기 로그(WAL)를 메모리 버퍼에 삽입하기 위해 대기 중입니다.

LWLock:WALBufMapping

프로세스가 미리 쓰기 로그(WAL) 버퍼의 페이지를 대체하기 위해 대기 중입니다.

LWLock:WALBufMappingLock

프로세스가 미리 쓰기 로그(WAL) 버퍼의 페이지를 대체하기 위해 대기 중입니다.

LWLock:WALInsert

프로세스가 미리 쓰기 로그(WAL)를 메모리 버퍼에 삽입하기 위해 대기 중입니다.

LWLock:WALWrite

프로세스가 미리 쓰기 로그(WAL) 버퍼가 디스크에 기록될 때까지 대기 중입니다.

LWLock:WALWriteLock

프로세스가 미리 쓰기 로그(WAL) 버퍼가 디스크에 기록될 때까지 대기 중입니다.

LWLock:WrapLimitsVacuum

프로세스가 트랜잭션 ID 및 멀티액트 소비에 대한 한도를 업데이트하기 위해 기다리고 있습니다.

LWLock:WrapLimitsVacuumLock

프로세스가 트랜잭션 ID 및 멀티액트 소비에 대한 한도를 업데이트하기 위해 기다리고 있습니다.

LWLock:XactBuffer

프로세스가 트랜잭션 상태에 대해 가장 오래전에 사용된 단순(SLRU) 버퍼에서 I/O를 기다리고 있습니다.

LWLock:XactSLRU

프로세스가 트랜잭션 상태에 대해 가장 오래전에 사용된 단순(SLRU) 캐시에 액세스하기 위해 대기 중입니다.

LWLock:XactTruncation

프로세스가 pg_xact_status를 실행하거나 사용 가능한 가장 오래된 트랜잭션 ID를 업데이트하기 위해 대기 중입니다.

LWLock:XidGen

프로세스가 새 트랜잭션 ID를 할당하기 위해 대기 중입니다.

LWLock:XidGenLock

프로세스가 트랜잭션 ID를 할당하거나 배정하기 위해 대기 중입니다.

Timeout:BaseBackupThrottle

프로세스가 작업 제한 중 기본 백업 중에 대기 중입니다.

Timeout:PgSleep

백엔드 프로세스는 pg_sleep 함수를 호출하고 절전 시간 초과가 만료될 때까지 기다리고 있습니다. 자세한 내용은 Timeout:PgSleep 섹션을 참조하세요.

Timeout:RecoveryApplyDelay

프로세스가 지연 설정으로 인한 복구 중에 미리 쓰기 로그(WAL)를 적용하기 위해 대기 중입니다.

Timeout:RecoveryRetrieveRetryInterval

프로세스가 소스(pg_wal, 아카이브 또는 스트림)에서 미리 쓰기 로그 (WAL) 데이터를 사용할 수 없어 복구가 진행되는 동안 대기 중입니다.

Timeout:VacuumDelay

프로세스가 비용 기반 베큠 지연 지점에서 대기 중입니다.

PostgreSQL 대기 이벤트의 전체 목록을 보려면 PostgreSQL 설명서의 통계 수집기 > 대기 이벤트 테이블을 참조하세요.

Amazon Aurora PostgreSQL에 대한 데이터베이스 엔진 업데이트

다음에서 Amazon Aurora PostgreSQL 엔진 버전 릴리스 및 업데이트에 대한 정보를 확인할 수 있습니다. Aurora PostgreSQL 엔진을 업그레이드하는 방법에 대한 정보도 확인할 수 있습니다. Aurora 릴리스에 대한 자세한 내용은 Amazon Aurora 섹션을 참조하세요.

> **Tip**
> 블루/그린 배포를 사용하면 DB 클러스터 업그레이드에 필요한 가동 중지를 최소화할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 블루/그린 배포 사용 단원을 참조하십시오.

주제

- Amazon Aurora PostgreSQL 버전 식별
- Amazon Aurora PostgreSQL 릴리스 및 엔진 버전
- Amazon Aurora PostgreSQL의 확장 버전
- Amazon Aurora PostgreSQL DB 클러스터 업그레이드
- Aurora PostgreSQL 장기 지원(LTS) 릴리스 사용

Amazon Aurora PostgreSQL 버전 식별

Amazon Aurora는 Aurora에 일반적이며 모든 Aurora DB 클러스터에서 사용할 수 있는 특정 기능을 포함합니다. Aurora는 Aurora가 지원하는 특정 데이터베이스 엔진에 특정한 다른 기능을 포함합니다. 이러한 기능들은 Aurora PostgreSQL 같이 해당 데이터베이스 엔진을 사용하는 Aurora DB 클러스터에만 제공됩니다.

Aurora 데이터베이스 릴리스에는 일반적으로 데이터베이스 엔진 버전 번호와 Aurora 버전 번호의 두 가지 버전 번호가 있습니다. Aurora PostgreSQL 릴리스에 Aurora 버전 번호가 있는 경우 Amazon Aurora PostgreSQL 릴리스 및 엔진 버전 목록에서 엔진 버전 번호 뒤에 포함됩니다.

주제

- Aurora 버전 번호
- PostgreSQL 엔진 버전 번호

Aurora 버전 번호

Aurora 버전 번호는 *major.minor.patch* 이름 지정 체계를 사용합니다. Aurora 패치 버전에는 릴리스 후 마이너 버전에 추가된 중요한 버그 수정이 포함되어 있습니다. Amazon Aurora 메이저, 마이너 및 패치 릴리스에 대한 자세한 내용은 Amazon Aurora 메이저 버전, Amazon Aurora 마이너 버전 및Amazon Aurora 패치 버전 섹션을 참조하세요.

다음 SQL 쿼리를 사용하여 Aurora PostgreSQL DB 인스턴스의 Aurora 버전 번호를 확인할 수 있습니다.

```
postgres=> SELECT aurora_version();
```

PostgreSQL 버전 13.3, 12.8, 11.13, 10.18 및 기타 모든 이후 버전의 릴리스부터 Aurora 버전 번호는 PostgreSQL 엔진 버전에 더 가깝게 맞춰집니다. 예를 들어 Aurora PostgreSQL 13.3 DB 클러스터를 쿼리하면 다음이 반환됩니다.

```
aurora_version
----------------
 13.3.1
(1 row)
```

Aurora PostgreSQL 10.14 DB 클러스터와 같은 이전 릴리스는 다음과 유사한 버전 번호를 반환합니다.

```
aurora_version
----------------
 2.7.3
(1 row)
```

PostgreSQL 엔진 버전 번호

PostgreSQL 10부터 PostgreSQL 데이터베이스 엔진 버전은 모든 릴리스에 대해 *major.minor* 번호 지정 체계를 사용합니다. 몇 가지 예로 PostgreSQL 10.18, PostgreSQL 12.7 및 PostgreSQL 13.3이 있습니다.

PostgreSQL 10 이전 릴리스에서는 처음 두 자리가 메이저 버전 번호를 구성하고 세 번째 숫자가 마이너 버전을 나타내는 *major.major.minor* 번호 지정 체계를 사용합니다. 예를 들어 PostgreSQL 9.6은 메이저 버전이며 마이너 버전 9.6.21 또는 9.6.22는 세 번째 숫자로 표시됩니다.

> **Note**
>
> PostgreSQL 엔진 버전 9.6은 더 이상 지원되지 않습니다. 업그레이드하려면 Amazon Aurora PostgreSQL DB 클러스터 업그레이드 섹션을 참조하세요. 버전 정책 및 출시 일정은 Amazon Aurora 메이저 버전을 사용할 수 있는 기간 단원을 참조하세요.

다음 SQL 쿼리를 사용하여 PostgreSQL 데이터베이스 엔진 버전 번호를 확인할 수 있습니다.

```
postgres=> SELECT version();
```

Aurora PostgreSQL 13.3 DB 클러스터의 경우 결과는 다음과 같습니다.

```
version
-------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Amazon Aurora PostgreSQL 릴리스 및 엔진 버전

Amazon Aurora PostgreSQL 호환 버전 릴리스는 정기적으로 업데이트됩니다. 업데이트는 시스템 유지 관리 기간 중에 Aurora PostgreSQL DB 클러스터에 적용됩니다. 업데이트 적용 시기는 업데이트 유형, AWS 리전 및 DB 클러스터의 유지 관리 기간 설정에 따라 다릅니다. 나열된 릴리스에는 대다수 PostgreSQL 버전 번호와 Amazon Aurora 버전 번호가 모두 포함되어 있습니다. 그러나 PostgreSQL 버전 13.3, 12.8, 11.13, 10.18의 릴리스부터 다른 모든 이후 버전에서는 Aurora 버전 번호가 사용되지 않습니다. Aurora PostgreSQL 데이터베이스의 버전 번호를 확인하려면 Amazon Aurora PostgreSQL 버전 식별 단원을 참조하세요.

확장 및 모듈에 대한 자세한 내용은 Amazon Aurora PostgreSQL의 확장 버전 단원을 참조하세요.

> **Note**
>
> Amazon Aurora 버전 정책 및 출시 일정에 대한 자세한 내용은 Amazon Aurora 메이저 버전을 사용할 수 있는 기간 섹션을 참조하세요.

Amazon Aurora에 대한 지원 및 기타 정책에 대한 내용은 Amazon RDS FAQ를 참조하세요.

AWS 리전에서 사용할 수 있는 Aurora PostgreSQL DB 엔진 버전을 확인하려면 describe-db-engine-versions AWS CLI 명령을 사용합니다. 예:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

AWS 리전 목록은 Aurora PostgreSQL 리전 가용성 단원을 참조하세요.

Aurora PostgreSQL에서 사용 가능한 PostgreSQL 버전에 대한 자세한 내용은 Aurora PostgreSQL릴리스 정보를 참조하세요.

Amazon Aurora PostgreSQL의 확장 버전

Aurora PostgreSQL DB 클러스터에서 사용할 수 있도록 다양한 PostgreSQL 확장을 설치하고 구성할 수 있습니다. 예를 들어 PostgreSQL `pg_partman` 확장을 사용하여 테이블 파티션의 생성 및 유지 관리를 자동화할 수 있습니다. 이 확장과 Aurora PostgreSQL에 사용할 수 있는 기타 확장에 대한 자세한 내용은 확장 및 외부 데이터 래퍼 작업을 참조하세요.

Aurora PostgreSQL에서 지원되는 PostgreSQL 확장에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 Amazon Aurora PostgreSQL의 확장 버전을 참조하세요.

Amazon Aurora PostgreSQL DB 클러스터 업그레이드

Amazon Aurora는 광범위한 테스트를 거친 후에만 PostgreSQL 데이터베이스 엔진의 새 버전을 AWS 리전에 공개합니다. Aurora PostgreSQL DB 클러스터를 리전에서 사용할 수 있는 경우 새 버전으로 업그레이드할 수 있습니다.

DB 클러스터가 현재 실행 중인 Aurora PostgreSQL 버전에 따라 새 릴리스로 업그레이드하는 작업은 마이너 업그레이드일 수도, 메이저 업그레이드일 수도 있습니다. 예를 들어, Aurora PostgreSQL 11.15 DB 클러스터를 Aurora PostgreSQL 13.6으로 업그레이드하는 것은 메이저 버전 업그레이드입니다. Aurora PostgreSQL 13.3 DB 클러스터를 Aurora PostgreSQL 13.7로 업그레이드하는 작업은 마이너 버전 업그레이드입니다. 다음 주제에는 두 유형의 업그레이드를 수행하는 방법에 대한 정보가 나와 있습니다.

목차

Aurora PostgreSQL 업그레이드 프로세스 개요

메이저 버전과 마이너 버전 업그레이드의 차이점은 다음과 같습니다.

마이너 버전 업그레이드 및 패치

마이너 버전 업그레이드 및 패치에는 기존 애플리케이션과 역호환되는 변경 사항만 포함됩니다. 마이너 버전 업그레이드 및 패치는 Aurora PostgreSQL이 테스트하고 승인한 후에만 사용할 수 있습니다.

Aurora는 자동으로 마이너 버전 업그레이드를 적용할 수 있습니다. 새로운 Aurora PostgreSQL DB 클러스터를 만들면 마이너 버전 업그레이드 활성화 옵션이 기본적으로 선택되어 있습니다. 이 옵션을 수동으로 끄지 않는 한 Aurora는 예정된 유지 관리 기간에 마이너 버전 자동 업그레이드를 주기적으로 적용합니다. 자동 마이너 버전 업그레이드(AmVU) 옵션과 AmVU를 사용하도록 Aurora DB 클러스터를 수정하는 방법에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하세요.

Aurora PostgreSQL DB 클러스터에 대해 마이너 버전 자동 업그레이드가 사용 설정되지 않은 경우 Aurora PostgreSQL이 새 마이너 버전으로 자동으로 업그레이드되지 않습니다. 대신 AWS 리전에 새 마이너 버전이 출시될 때 Aurora PostgreSQL DB 클러스터에서 이전 마이너 버전을 실행 중이면 Aurora에서 업그레이드하라는 메시지가 표시됩니다. 이 작업을 수행하려면 클러스터의 유지 관리 태스크에 권장 사항을 추가하면 됩니다.

패치는 업그레이드로 간주되지 않으며, 자동으로 적용되지 않습니다. Aurora PostgreSQL에서는 Aurora PostgreSQL DB 클러스터의 유지 관리 태스크에 권장 사항을 추가하여 패치를 적용하라는 메시지를 표시합니다. 자세한 내용은 마이너 버전 업그레이드 수행 및 패치 적용 방법을 참조하세요.

> **Note**
>
> 보안 또는 기타 중요한 문제를 해결하는 패치도 유지 관리 태스크로 추가됩니다. 그러나 이러한 패치는 필수입니다. 보류 중인 유지 관리 태스크에서 사용할 수 있게 되면 Aurora PostgreSQL DB 클러스터에 보안 패치를 적용해야 합니다.

업그레이드를 진행하는 과정에서 클러스터의 각 인스턴스가 새 버전으로 업그레이드될 때 일시적인 중단이 발생할 수 있습니다. 그러나 Aurora PostgreSQL 14.3.3, 13.7.3, 12.11.3, 11.16.3, 10.21.3 및 기타 마이너 버전의 상위 릴리스 및 그 이후의 메이저 버전에는 업그레이드 프로세스에 제로 가동 중지 패치 적용(ZDP) 기능이 도입되었습니다. 이 기능은 중단을 최소화하며, 대부분의 경우 중단이 발생하지 않습니다. 자세한 내용은 마이너 릴리스 업그레이드 및 제로 가동 중지 패치 적용 섹션을 참조하세요. ZDP의 지원되는 기능 및 제한 사항에 대한 자세한 내용은 제로 가동 중지 패치 적용의 제한 사항 섹션을 참조하세요.

메이저 버전 업그레이드

마이너 버전 업그레이드 및 패치와 달리 Aurora PostgreSQL에는 자동 메이저 버전 업그레이드 옵션이 없습니다. 최신 메이저 PostgreSQL 버전에는 기존 애플리케이션과 역호환되지 않는 데이터베이스 변경 사항이 포함될 수 있습니다. 새 기능을 사용하면 기존 애플리케이션이 올바르게 작동하지 않을 수 있습니다.

문제를 방지하려면 Aurora PostgreSQL DB 클러스터에서 DB 인스턴스를 업그레이드하기 전에 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트에 설명된 프로세스를 따르는 것이 좋습니다. 먼저 해당 절차에 따라 애플리케이션이 새 버전에서 실행될 수 있는지 확인합니다. 그런 다음 Aurora PostgreSQL DB 클러스터를 새 버전으로 직접 업그레이드하면 됩니다.

업그레이드 프로세스는 클러스터의 모든 인스턴스를 새 버전으로 업그레이드할 때 잠시 중단될 수 있습니다. 예비 계획 프로세스에도 시간이 걸립니다. 업그레이드 태스크는 항상 클러스터의 유지 관리 기간 동안 또는 작업이 거의 이루어지지 않는 기간에 수행하는 것이 좋습니다. 자세한 내용은 메이저 버전 업그레이드 수행 단원을 참조하세요.

> **Note**
>
> 마이너 버전과 메이저 버전을 업그레이드할 때는 중단이 짧게 발생할 수 있습니다. 따라서 유지 관리 기간 동안 또는 사용률이 많지 않은 기간에 업그레이드를 수행하거나 예약하는 것이 좋습니다.

Aurora PostgreSQL DB 클러스터는 때때로 운영 체제 업데이트가 필요합니다. 이러한 업데이트에는 glibc 라이브러리의 최신 버전도 포함될 수 있습니다. 이러한 업데이트를 진행하는 중에는 Aurora PostgreSQL에서 지원되는 데이터 정렬에 설명된 지침을 따르는 것이 좋습니다.

AWS 리전에서 사용 가능한 버전 목록 가져오기

다음과 같이 describe-db-engine-versions AWS CLI 명령을 통해 AWS 리전을 쿼리하여 Aurora PostgreSQL DB 클러스터의 업그레이드 대상으로 사용할 수 있는 모든 엔진 버전 목록을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Windows의 경우:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

예를 들어, Aurora PostgreSQL 버전 12.10 DB 클러스터의 유효한 업그레이드 대상을 식별하려면 다음 AWS CLI 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
```

```
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Windows의 경우:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

다음 표에서 서로 다른 Aurora PostgreSQL DB 버전의 메이저 버전 및 마이너 버전 업그레이드 대상을 모두 찾을 수 있습니다. 호환성을 유지하기 위해 일부 버전만 업그레이드 대상으로 제공됩니다. Aurora PostgreSQL은 분기별 마이너 버전 릴리스마다 새로운 기능을 도입하고 버그를 수정했습니다. Aurora PostgreSQL 마이너 릴리스에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

현재 소스 버전	업그레이드 대상
17.5	없음
17.4	17.5, 17.4
16.9	17.5
16.8	17.5, 17.4 16.9
16.6	17.5, 17.4 16.9, 16.8
16.4	17.5, 17.4 16.9, 16.8, 16.6
16.3	17.5, 17.4 16.9, 16.8, 16.6, 16.4
16.2	17.5, 17.4

현재 소스 버전	업그레이드 대상
	16.9, 16.8 16.6, 16.4, 16.3
16.1	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2
15.13	17.5 16.9
15.12	17.5, 17.4 16.9, 16.8 15.13
15.10	17.5, 17.4 16.9, 16.8, 16.6 15.13, 15.12
15.8	17.5, 17.4 16.9, 16.8, 16.6, 16.4 15.13, 15.12, 15.10
15.7	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3 15.13, 15.12, 15.10, 15.8
15.6	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2 15.13, 15.12, 15.10, 15.8, 15.7

현재 소스 버전	업그레이드 대상
15.5	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6
15.4	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5
15.3	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4
15.2	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3
14.18	17.5 16.9 15.13
14.17	17.5, 17.4 16.9, 16.8 15.13, 15.12 14.18

현재 소스 버전	업그레이드 대상
14.15	17.5, 17.4 16.9, 16.8, 16.6 15.13, 15.12, 15.10 14.18, 14.17
14.13	17.5, 17.4 16.9, 16.8, 16.6, 16.4 15.13, 15.12, 15.10, 15.8 14.18, 14.17, 14.15
14.12	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3 15.13, 15.12, 15.10, 15.8, 15.7 14.18, 14.17, 14.15, 14.13
14.11	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2 15.13, 15.12, 15.10, 15.8, 15.6 14.18, 14.17, 14.15, 14.13, 14.12
14.10	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5 14.18, 14.17, 14.15, 14.13, 14.12, 14.11

현재 소스 버전	업그레이드 대상
14.9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10
14.8	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9
14.7	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8
14.6	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7
14.5	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6

현재 소스 버전	업그레이드 대상
14.4	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5
14.3	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5, 14.4
13.21	17.5 16.9 15.13 14.18
13.20	17.5, 17.4 16.9, 16.8 15.13, 15.12 14.18, 14.17 13.21

현재 소스 버전	업그레이드 대상
13.18	17.5, 17.4 16.9, 16.8, 16.6 15.13, 15.12, 15.10 14.18, 14.17, 14.15 13.21, 13.20
13.16	17.5, 17.4 16.9, 16.8, 16.6, 16.4 15.13, 15.12, 15.10, 15.8 14.18, 14.17, 14.15, 14.13 13.21, 13.20, 13.18
13.15	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3 15.13, 15.12, 15.10, 15.8, 15.7 14.18, 14.17, 14.15, 14.13, 14.12 13.21, 13.20, 13.18, 13.16
13.14	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2 15.13, 15.12, 15.10, 15.8, 15.7, 15.6 14.18, 14.17, 14.15, 14.13, 14.11 13.21, 13.20, 13.18, 13.16, 13.15

현재 소스 버전	업그레이드 대상
13.13	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10 13.21, 13.20, 13.18, 13.16, 13.15, 13.14
13.12	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13
13.11	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12
13.10	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11

현재 소스 버전	업그레이드 대상
13.9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13,13.12, 13.11, 13.10
13.8	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9
13.7	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5, 14.4, 14.3 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8

현재 소스 버전	업그레이드 대상
12.22	17.5, 17.4 16.9, 16.8, 16.6 15.13, 15.12, 15.10 14.18, 14.17, 14.15 13.21, 13.20, 13.18
12.20	17.5, 17.4 16.9, 16.8, 16.6, 16.4 15.13, 15.12, 15.10, 15.8 14.15, 14.13 13.21, 13.20, 13.18, 13.16 12.22
12.19	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3 15.13, 15.12, 15.10, 15.8, 15.7 14.18, 14.17, 14.15, 14.13, 14.12 13.21, 13.20, 13.18, 13.16, 13.15 12.22, 12.20

현재 소스 버전	업그레이드 대상
12.18	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2 15.13, 15.12, 15.10, 15.8, 15.7, 15.6 14.18, 14.17, 14.15, 14.13, 14.12, 14.11 13.21, 13.20, 13.18, 13.16, 13.15, 13.14 12.22, 12.20, 12.19
12.17	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13 12.22, 12.20, 12.19, 12.18
12.16	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12 12.22, 12.20, 12.19, 12.18, 12.17

현재 소스 버전	업그레이드 대상
12.15	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11 12.22, 12.20, 12.19, 12.18, 12.17, 12.16
12.14	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15
12.13	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14

현재 소스 버전	업그레이드 대상
12.12	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13
12.11	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.5, 14.4, 14.3 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12

현재 소스 버전	업그레이드 대상
12.9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11
11.21	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12 12.22, 12.20, 12.19, 12.18, 12.17, 12.16

현재 소스 버전	업그레이드 대상
11.9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11, 12.09 11.21

고려 중인 버전에 관계없이 클러스터 DB 인스턴스 클래스의 가용성을 항상 확인해야 합니다. 예를 들어 `db.r4`는 Aurora PostgreSQL 13에서 지원되지 않습니다. Aurora PostgreSQL DB 클러스터에서 현재 db.r4 인스턴스 클래스를 사용한다면, 업그레이드하기 전에 db.r5로 전환해야 합니다. Graviton2 기반 클래스 및 인텔 기반 클래스 등 DB 인스턴스 클래스에 대한 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 섹션을 참조하세요.

메이저 버전 업그레이드 수행

메이저 버전 업그레이드에는 이전 버전의 데이터베이스와 역호환되지 않는 데이터베이스 변경 사항이 포함될 수 있습니다. 최신 버전의 새 기능을 사용하면 기존 애플리케이션이 올바르게 작동하지 않을 수 있습니다. 이러한 문제를 방지하고자 Amazon Aurora는 자동으로 메이저 버전 업그레이드를 적용하지 않습니다. 따라서 다음 단계에 따라 메이저 버전 업그레이드를 신중하게 계획하는 것이 좋습니다.

1. 테이블에 있는 버전에 대해 나열된 대상 중 사용 가능한 대상 목록에서 원하는 메이저 버전을 선택합니다. AWS CLI를 사용하여 현재 버전에 대해 AWS 리전에서 사용할 수 있는 정확한 버전 목록을 확인할 수 있습니다. 자세한 내용은 AWS 리전에서 사용 가능한 버전 목록 가져오기 단원을 참조하세요.
2. 새 버전의 평가판 배포에서 애플리케이션이 정상적으로 작동하는지 확인합니다. 전체 프로세스에 대한 내용은 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트 섹션을 참조하세요.

3. 평가판 배포에서 애플리케이션이 정상적으로 작동하는지 확인했다면 클러스터를 업그레이드할 수 있습니다. 자세한 내용은 Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드을 참조하세요.

> **Note**
>
> 13.6부터 시작하는 Babelfish for Aurora PostgreSQL 13 기반 버전에서 14.6부터 시작하는 Aurora PostgreSQL 14 기반 버전으로 메이저 버전 업그레이드를 수행할 수 있습니다. Babelfish for Aurora PostgreSQL 13.4 및 13.5는 메이저 버전 업그레이드를 지원하지 않습니다.

다음과 같이 describe-db-engine-versions AWS CLI 명령을 통해 AWS 리전을 쿼리하여 Aurora PostgreSQL DB 클러스터의 메이저 버전 업그레이드 대상으로 사용할 수 있는 엔진 버전 목록을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

Windows의 경우:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

업그레이드하려는 버전이 현재 버전의 대상이 아닌 경우가 있습니다. 이러한 경우 클러스터가 대상 행에 선택한 대상이 있는 버전이 될 때까지 versions table의 정보를 바탕으로 마이너 버전 업그레이드를 수행합니다.

새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트

최신 메이저 버전에는 각각 성능 개선을 위해 설계된 쿼리 옵티마이저의 개선 사항이 포함되어 있습니다. 그러나 워크로드에 새 버전에서 성능이 저하되는 쿼리가 포함되어 있을 수 있습니다. 따라서 프로덕션 환경에서 업그레이드하기 전에 성능을 테스트하고 검토하는 것이 좋습니다. 메이저 버전 업그레이드 후 계획 안정성 보장에 설명된 대로 QPM(쿼리 계획 관리) 확장을 사용하여 버전 간 쿼리 계획 안정성을 관리할 수 있습니다.

프로덕션 Aurora PostgreSQL DB 클러스터를 새로운 메이저 버전으로 업그레이드하기 전에 업그레이드를 테스트하여 모든 애플리케이션이 제대로 작동하는지 확인하는 것이 좋습니다.

1. 버전 호환 파라미터 그룹을 준비합니다.

   사용자 지정 DB 인스턴스 또는 DB 클러스터 파라미터 그룹을 사용하는 경우 다음 2가지 옵션이 있습니다.

   a. 새 DB 엔진 버전에 대한 기본 DB 인스턴스, DB 클러스터 파라미터 그룹 또는 둘 다 지정합니다.

   b. 새 DB 엔진 버전에 대한 사용자 지정 파라미터 그룹을 직접 생성합니다.

   업그레이드 요청의 일부로 새 DB 인스턴스 또는 DB 클러스터 파라미터 그룹을 연결하는 경우 파라미터를 적용하려면 업그레이드가 완료된 후 데이터베이스를 재부팅해야 합니다. 파라미터 그룹 변경 사항을 적용하기 위해 DB 인스턴스를 재부팅해야 하는 경우 인스턴스의 파라미터 그룹 상태가 `pending-reboot`로 표시됩니다. 인스턴스의 파라미터 그룹 상태는 콘솔에서 보거나 describe-db-instances 또는 describe-db-clusters 같은 CLI 명령을 사용하여 볼 수 있습니다.

2. 유효하지 않은 데이터베이스가 있는지 확인하고 유효하지 않은 데이터베이스를 삭제합니다.

   `pg_database` 카탈로그의 `datconnlimit` 열에는 `DROP DATABASE` 작업 중에 중단된 데이터베이스를 유효하지 않은 것으로 표시하는 `-2` 값이 포함되어 있습니다. 다음 쿼리를 사용하여 유효하지 않은 데이터베이스가 있는지 확인합니다.

   ```
   SELECT
       datname
   FROM
       pg_database
   WHERE
       datconnlimit = - 2;
   ```

   쿼리가 데이터베이스 이름을 반환하면 이러한 데이터베이스는 유효하지 않습니다. `DROP DATABASE invalid_db_name` 스테이트먼트를 사용하여 유효하지 않은 데이터베이스를 삭제합니다. 다음 동적 스테이트먼트를 사용하여 모든 유효하지 않은 데이터베이스를 삭제할 수 있습니다.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

3. 지원되지 않는 사용 확인:

   - 업그레이드를 시도하기 전에 열려 있는 준비된 트랜잭션을 모두 커밋하거나 롤백합니다. 다음 쿼리를 사용하여 인스턴스에 열려 있는 준비된 트랜잭션이 없음을 확인할 수 있습니다.

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - 업그레이드를 시도하기 전에 reg* 데이터 형식의 사용을 모두 제거하세요. `regtype` 및 `regclass` 이외에는 reg* 데이터 형식을 업그레이드할 수 없습니다. pg_upgrade 유틸리티(Amazon Aurora에서 업그레이드하는 데 사용)는 이 데이터 유형을 유지할 수 없습니다. 유틸리티에 대해 자세히 알아보려면 PostgreSQL 설명서의 [pg_upgrade]를 참조하세요.

     지원되지 않는 reg* 데이터 형식이 사용되지 않음을 확인하려면 각 데이터베이스에 다음 쿼리를 사용합니다.

     ```
     SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
      pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid
           AND NOT a.attisdropped
           AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                              'pg_catalog.regprocedure'::pg_catalog.regtype,
                              'pg_catalog.regoper'::pg_catalog.regtype,
                              'pg_catalog.regoperator'::pg_catalog.regtype,
                              'pg_catalog.regconfig'::pg_catalog.regtype,
                              'pg_catalog.regdictionary'::pg_catalog.regtype)
           AND c.relnamespace = n.oid
           AND n.nspname NOT IN ('pg_catalog', 'information_schema');
     ```

   - `pgRouting` 확장이 설치된 Aurora PostgreSQL 버전 10.18 이상의 DB 클러스터를 업그레이드하는 경우 버전 12.4 이상으로 업그레이드하기 전에 이 확장을 제거합니다.

     확장 `pg_repack` 버전 1.4.3이 설치된 Aurora PostgreSQL 10.x 버전을 업그레이드하는 경우, 더 높은 버전으로 업그레이드하기 전에 이 확장을 제거합니다.

4. template1 및 template0 데이터베이스를 확인합니다.

성공적인 업그레이드를 위해서는 template1 및 template0 데이터베이스가 존재해야 하며 템플릿으로 나열되어야 합니다. 이를 확인하려면 다음 명령을 사용합니다.

```
SELECT datname, datistemplate FROM pg_database;

datname     | datistemplate
-----------+---------------
template0   | t
rdsadmin    | f
template1   | t
postgres    | f
```

명령 출력에서 template1 및 template0 데이터베이스의 `datistemplate` 값은 `t`여야 합니다.

5. 논리적 복제 슬롯을 삭제합니다.

   Aurora PostgreSQL DB 클러스터가 논리적 복제 슬롯을 사용 중인 경우 업그레이드 프로세스를 진행할 수 없습니다. 논리적 복제 슬롯은 일반적으로 AWS DMS를 사용하여 데이터를 마이그레이션하거나 데이터베이스에서 데이터 레이크, BI 도구 또는 기타 대상으로 테이블을 복제하는 것과 같은 단기 데이터 마이그레이션 작업에 사용됩니다. 업그레이드하기 전에 존재하는 논리적 복제 슬롯의 용도를 알고 삭제해도 되는지 확인합니다. 다음 쿼리를 사용하여 논리적 복제 슬롯을 확인할 수 있습니다.

   ```
   SELECT * FROM pg_replication_slots;
   ```

   논리적 복제 슬롯이 계속 사용 중인 경우 삭제하면 안 되며 업그레이드를 진행할 수 없습니다. 그러나 논리적 복제 슬롯이 필요하지 않은 경우 다음 SQL을 사용하여 삭제할 수 있습니다.

   ```
   SELECT pg_drop_replication_slot(slot_name);
   ```

   `pglogical` 확장을 사용하는 논리적 복제 시나리오에서도, 게시자 노드에서 메이저 버전 업그레이드가 성공하려면 이 노드에서 슬롯을 삭제해야 합니다. 하지만 업그레이드가 끝나면 구독자 노드에서 복제 프로세스를 재시작할 수 있습니다. 자세한 내용은 [메이저 업그레이드 후 논리적 복제 재설정](#) 섹션을 참조하세요.

6. 백업을 수행합니다.

   업그레이드 프로세스는 업그레이드 중에 DB 클러스터의 DB 클러스터 스냅샷을 생성합니다. 업그레이드 프로세스 전에 수동 백업을 수행하려면 [DB 클러스터 스냅샷 생성](#) 단원을 참조하십시오.

7. 메이저 버전 업그레이드를 수행하기 전에 특정 확장을 사용 가능한 최신 버전으로 업그레이드합니다. 업데이트할 확장은 다음과 같습니다.

   - pgRouting
   - postgis_raster
   - postgis_tiger_geocoder
   - postgis_topology
   - address_standardizer
   - address_standardizer_data_us

   현재 설치된 각 확장에 대해 다음 명령을 실행합니다.

   ```
   ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
   ```

   자세한 내용은 [PostgreSQL 확장 버전 업그레이드]을 참조하세요. PostGIS 업그레이드에 대한 자세한 내용은 [6단계: PostGIS 확장 업그레이드]를 참조하세요.

8. 버전 11.x로 업그레이드하는 경우 메이저 버전 업그레이드를 수행하기 전에 지원되지 않는 확장을 삭제하세요. 삭제할 확장은 다음과 같습니다.

   - chkpass
   - tsearch2

9. 대상 버전에 따라 unknown 데이터 형식을 삭제합니다.

   PostgreSQL 버전 10은 unknown 데이터 형식을 지원하지 않습니다. 버전 9.6 데이터베이스가 unknown 데이터 형식을 사용하는 경우 버전 10으로 업그레이드하면 다음과 같은 오류 메시지가 표시됩니다.

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   이러한 열을 제거하거나 지원되는 데이터 형식으로 변경할 수 있도록 데이터베이스에서 unknown 데이터 형식을 찾으려면 각 데이터베이스에 대해 다음 SQL 코드를 사용합니다.

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
   ```

```
    pg_catalog.pg_attribute a
    WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
    a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
    c.relkind IN ('r','m','c') AND
    c.relnamespace = n.oid AND
    n.nspname !~ '^pg_temp_' AND
    n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
 'information_schema');
```

10.업그레이드 모의 실습을 수행합니다.

프로덕션 데이터베이스에서 업그레이드를 수행하기 전에 프로덕션 데이터베이스의 복제본에서 메이저 버전 업그레이드를 테스트하는 것이 좋습니다. 중복 테스트 인스턴스의 실행 계획을 모니터링하여 실행 계획 회귀가 발생할 수 있는지 확인하고 성능을 평가할 수 있습니다. 중복 테스트 인스턴스를 생성하려면 최근 스냅샷에서 데이터베이스를 복원하거나 데이터베이스를 복제합니다. 자세한 내용은 스냅샷에서 복원 또는 Aurora DB 클러스터에 대한 볼륨 복제 섹션을 참조하세요.

자세한 내용은 Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드 섹션을 참조하세요.

11.프로덕션 인스턴스를 업그레이드합니다.

메이저 버전 업그레이드의 모의 실습이 성공한 경우 확신을 가지고 프로덕션 데이터베이스를 업그레이드해도 됩니다. 자세한 내용은 Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드을 참조하세요.

> **Note**
>
> 업그레이드 프로세스 중에 클러스터의 백업 보존 기간이 0보다 긴 경우 Aurora PostgreSQL이 DB 클러스터 스냅샷을 생성합니다. 업그레이드 프로세스 중에 클러스터를 특정 시점으로 복원할 수 없습니다. 나중에 업그레이드가 시작되기 전과 인스턴스 자동 스냅샷이 완료된 후 시간으로 백업을 완료한 이후의 시간으로 특정 시점으로 복원을 수행할 수 있습니다. 그러나 이전 마이너 버전에는 특정 시점으로 복원을 수행할 수 없습니다.

진행 중인 업그레이드에 대한 자세한 내용은 Amazon RDS를 사용하여 pg_upgrade 유틸리티가 생성하는 두 개의 로그를 볼 수 있습니다. 이러한 로그는 `pg_upgrade_internal.log` 및 `pg_upgrade_server.log`입니다. Amazon Aurora는 이러한 로그의 파일 이름에 타임스탬프를 추가합니다. 다른 로그와 마찬가지로 이러한 로그를 볼 수 있습니다. 자세한 내용은 Amazon Aurora 로그 파일 모니터링 섹션을 참조하세요.

12PostgreSQL 확장을 업그레이드합니다. PostgreSQL 업그레이드 프로세스는 PostgreSQL 확장을 업그레이드하지 않습니다. 자세한 내용은 PostgreSQL 확장 버전 업그레이드 섹션을 참조하세요.

업그레이드 후 권장 사항

메이저 버전 업그레이드를 완료한 후에는 다음 작업을 수행하는 것이 좋습니다.

- ANALYZE 작업을 실행하여 pg_statistic 테이블을 새로 고칩니다. 모든 PostgreSQL DB 인스턴스의 모든 데이터베이스에 대해 이 작업을 수행해야 합니다. 옵티마이저 통계는 메이저 버전 업그레이드 중에 전송되지 않으므로 성능 문제를 방지하려면 모든 통계를 다시 생성해야 합니다. 파라미터 없이 명령을 실행하여 다음과 같이 현재 데이터베이스의 모든 일반 테이블에 대한 통계를 생성합니다.

```
ANALYZE VERBOSE;
```

  VERBOSE 플래그는 선택 사항이지만 이 플래그를 사용하면 진행 상황이 표시됩니다. 자세한 내용은 PostgreSQL 설명서의 ANALYZE를 참조하세요.

> **Note**
> 성능 문제를 방지하려면 업그레이드 후 시스템에서 ANALYZE를 실행합니다.

- PostgreSQL 버전 10으로 업그레이드한 경우 현재 있는 해시 인덱스에 대해 REINDEX를 실행합니다. 해시 인덱스는 버전 10에서 변경되었으며 다시 작성해야 합니다. 잘못된 해시 인덱스를 찾으려면 해시 인덱스가 포함된 각 데이터베이스에 대해 다음 SQL을 실행합니다.

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- 모든 것이 예상대로 작동하는지 확인하기 위해 유사한 워크로드로 업그레이드된 데이터베이스에서 애플리케이션을 테스트하는 것을 권장합니다. 업그레이드를 확인한 후 이 테스트 인스턴스를 삭제할 수 있습니다.

Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드

새 메이저 버전으로 업그레이드 프로세스를 시작하면 Aurora PostgreSQL은 클러스터를 변경하기 전에 Aurora DB 클러스터의 스냅샷을 생성합니다. 이 스냅샷은 마이너 버전 업그레이드가 아닌 메이저 버전 업그레이드에 대해서만 생성됩니다. 업그레이드 프로세스가 완료되면 RDS 콘솔의 스냅샷(Snapshots)에 나열된 수동 스냅샷 중에서 해당 스냅샷을 찾아볼 수 있습니다. 스냅샷 이름은 아래 예와 같이 접두사로 `preupgrade`, Aurora PostgreSQL DB 클러스터의 이름, 소스 버전, 대상 버전, 날짜 및 타임스탬프로 구성됩니다.

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

업그레이드가 완료된 후 필요한 경우 Aurora가 생성하여 수동 스냅샷 목록에 저장된 스냅샷을 통해 DB 클러스터를 이전 버전으로 복원할 수 있습니다.

> **Tip**
>
> 일반적으로 스냅샷은 Aurora DB 클러스터를 다양한 시점으로 복원하는 여러 방법을 제공합니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 및 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요. 하지만 Aurora PostgreSQL 스냅샷을 사용하여 이전 마이너 버전으로 복원하는 것을 지원하지 않습니다.

메이저 버전 업그레이드 프로세스 중에 Aurora는 볼륨을 할당하고 소스 Aurora PostgreSQL DB 클러스터를 복제합니다. 어떤 이유로든 업그레이드에 실패하면 Aurora PostgreSQL은 복제본을 사용하여 업그레이드를 롤백합니다. 소스 볼륨의 복제가 15개 이상 할당되면 후속 복제는 전체 복사본이 되고 시간이 오래 걸립니다. 이로 인해 업그레이드 프로세스가 더 오래 걸릴 수 있습니다. Aurora PostgreSQL에서 업그레이드를 롤백하는 경우 다음 사항에 유의하세요.

- 업그레이드 중에 할당된 원본 볼륨과 복제된 볼륨 모두에 대한 청구 항목 및 지표가 표시될 수 있습니다. Aurora PostgreSQL은 클러스터 백업 보존 기간이 업그레이드 시간을 초과하면 추가 볼륨을 정리합니다.
- 이 클러스터의 다음 교차 리전 스냅샷 복사본은 증분 복사본이 아닌 전체 복사본이 됩니다.

클러스터를 구성하는 DB 인스턴스를 안전하게 업그레이드할 수 있도록 Aurora PostgreSQL에서는 pg_upgrade 유틸리티를 사용합니다. 라이터 업그레이드가 완료되면 각 리더 인스턴스는 새 메이저 버전으로 업그레이드되는 동안 잠시 중단됩니다. PostgreSQL 유틸리티에 대해 자세히 알아보려면 PostgreSQL 설명서의 pg_upgrade를 참조하세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora PostgreSQL DB 클러스터를 새 버전으로 업그레이드할 수 있습니다.

콘솔

DB 클러스터의 엔진 버전을 업그레이드하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 업그레이드하려는 DB 클러스터를 선택합니다.
3. 수정을 선택합니다. DB 클러스터 수정 페이지가 나타납니다.
4. DB 엔진 버전에서 새 버전을 선택합니다.
5. 계속을 수정 사항을 요약한 내용을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다. 일부의 경우 이 옵션을 선택하면 중단이 발생할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.
7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 올바른 경우 클러스터 수정을 선택하여 변경 내용을 저장합니다.

   또는 뒤로를 선택하여 변경 내용을 편집하거나 취소를 선택하여 변경 내용을 취소합니다.

AWS CLI

DB 클러스터의 엔진 버전을 업그레이드하려면 modify-db-cluster AWS CLI 명령을 사용합니다. 다음 파라미터를 지정합니다.

- `--db-cluster-identifier` – DB 클러스터의 이름입니다.
- `--engine-version` – 업그레이드할 데이터베이스 엔진의 버전 번호입니다. 유효한 엔진 버전에 대한 정보를 보려면 AWS CLI describe-db-engine-versions 명령을 사용합니다.
- `--allow-major-version-upgrade` – `--engine-version` 파라미터가 DB 클러스터의 현재 메이저 버전과 다른 메이저 버전일 때 필수 플래그입니다.
- `--no-apply-immediately` – 변경 사항이 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 `--apply-immediately`를 사용합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

DB 클러스터의 엔진 버전을 업그레이드하려면 ModifyDBCluster 작업을 사용합니다. 다음 파라미터를 지정합니다.

- `DBClusterIdentifier` – DB 클러스터의 이름입니다(예:*mydbcluster*).
- `EngineVersion` – 업그레이드할 데이터베이스 엔진의 버전 번호입니다. 유효한 엔진 버전에 대한 정보를 보려면 DescribeDBEngineVersions 작업을 사용합니다.
- `AllowMajorVersionUpgrade` – `EngineVersion` 파라미터가 DB 클러스터의 현재 메이저 버전과 다른 메이저 버전일 때 필수 플래그입니다.
- `ApplyImmediately` – 변경 사항을 즉시 적용하거나 다음 유지 관리 기간에 적용합니다. 변경 사항을 바로 적용하려면 값을 `true`로 설정합니다. 변경 사항을 다음 유지 관리 기간에 적용하려면 값을 `false`로 설정합니다.

글로벌 데이터베이스에 대한 메이저 업그레이드

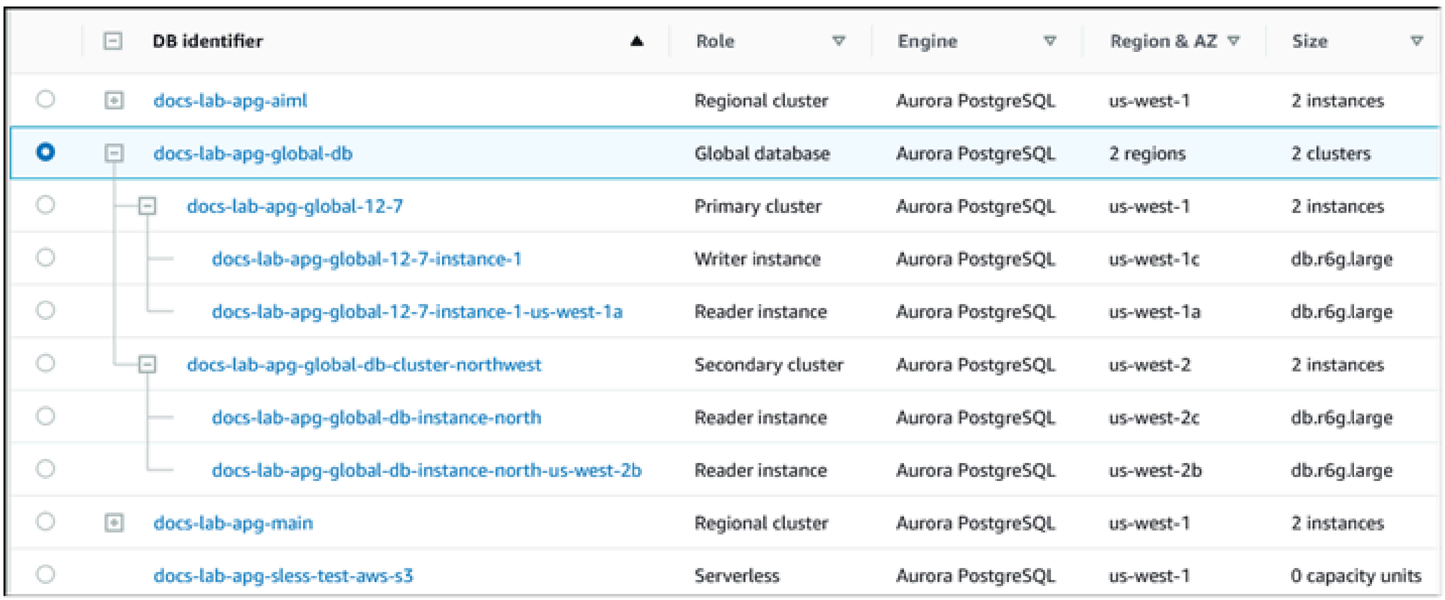
Aurora Global Database 클러스터의 경우 업그레이드 프로세스는 Aurora Global Database를 구성하는 모든 DB 클러스터를 동시에 업그레이드하여 모두 동일한 Aurora PostgreSQL 버전을 실행하도록 보장합니다. 또한, 시스템 테이블, 데이터 파일 형식 등에 대한 변경 사항이 모든 보조 클러스터에 자동으로 복제되도록 합니다.

글로벌 데이터베이스 클러스터를 최신 메이저 버전의 Aurora PostgreSQL로 업그레이드하려면 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트에 설명된 대로 업그레이드된 버전에서 애플리케이션을 테스트하는 것이 좋습니다. 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드

테스트의 step 1. 에 자세히 설명된 대로 업그레이드하기 전에 Aurora Global Database에서 각AWS 리전에 대한 DB 클러스터 파라미터 그룹 및 DB 파라미터 그룹 설정을 준비해야 합니다.

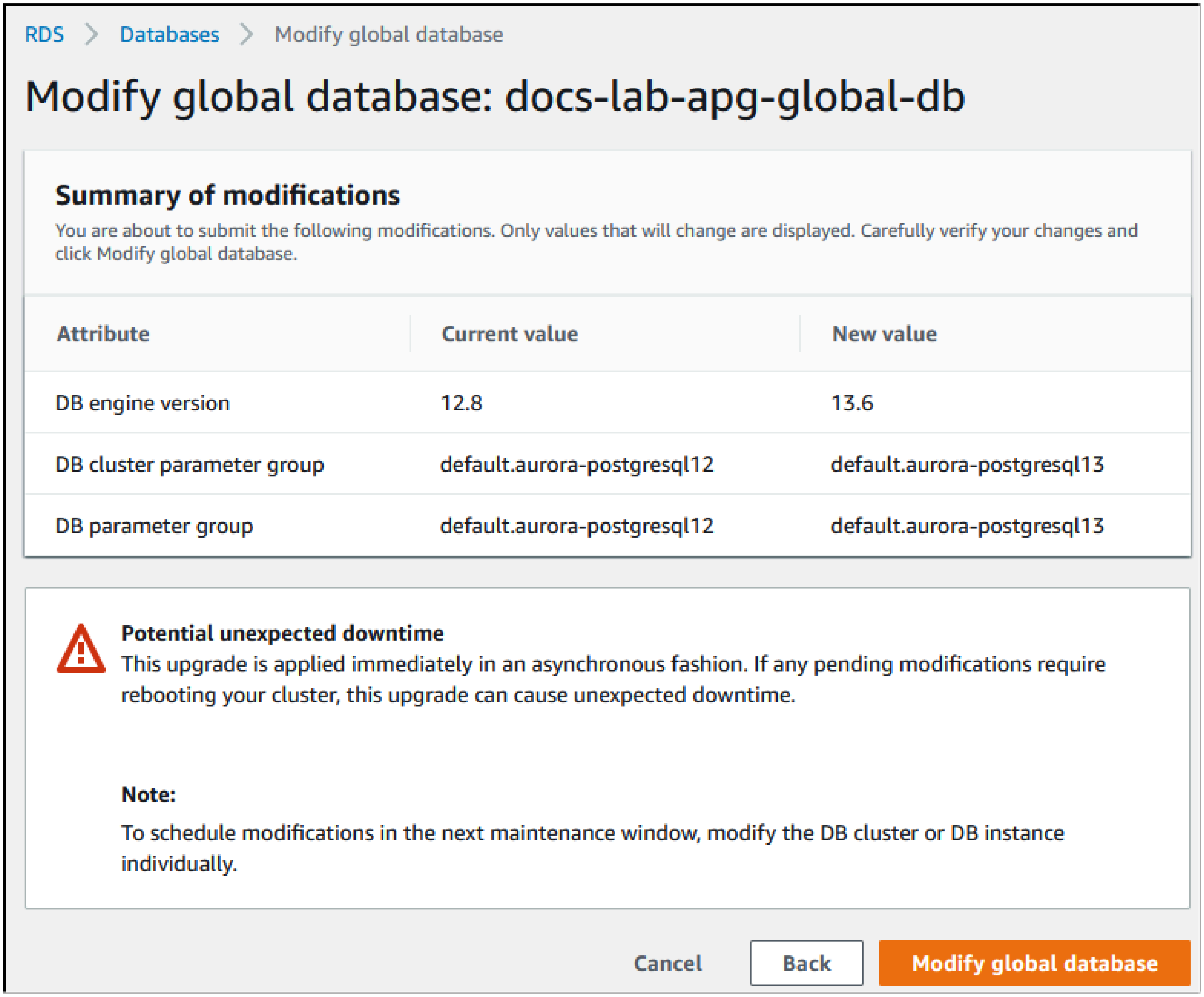
Aurora PostgreSQL 글로벌 데이터베이스 클러스터의 `rds.global_db_rpo` 파라미터에 Recovery Point Objective(RPO)가 설정된 경우 업그레이드하기 전에 해당 파라미터를 재설정해야 합니다. RPO가 켜져 있으면 메이저 버전 업그레이드 프로세스가 작동하지 않습니다. 기본적으로 이 파라미터는 비활성화됩니다. Aurora PostgreSQL 글로벌 데이터베이스 및 RPO에 대한 자세한 내용은 Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO 관리 섹션을 참조하세요.

새 버전의 평가판 배포에서 애플리케이션이 정상적으로 실행될 수 있는지 확인하면 업그레이드 프로세스를 시작할 수 있습니다. 그렇게 하려면 Aurora PostgreSQL 엔진을 새로운 메이저 버전으로 업그레이드 단원을 참조하세요. 다음 이미지와 같이 RDS 콘솔의 데이터베이스(Databases) 목록인 글로벌 데이터베이스(Global database)에서 최상위 항목을 선택해야 합니다.



DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

수정했을 때와 마찬가지로 메시지가 표시되어 프로세스를 진행할지 확인합니다.



콘솔을 사용하는 대신 AWS CLI 또는 RDS API를 사용하여 업그레이드 프로세스를 시작할 수 있습니다. 콘솔에서와 마찬가지로 다음과 같이 구성 요소가 아닌 Aurora Global Database 클러스터에서 작업합니다.

- modify-global-cluster AWS CLI 명령을 사용하여 AWS CLI로 Aurora Global Database 업그레이드를 시작합니다.
- ModifyGlobalCluster API를 사용하여 업그레이드를 시작합니다.

마이너 버전 업그레이드 수행

다음 방법을 사용하여 DB 클러스터의 마이너 버전을 업그레이드하거나 DB 클러스터에 패치를 적용할 수 있습니다.

주제

마이너 버전 업그레이드를 수행하기 전

마이너 버전 업그레이드 중에 가동 중지 시간을 줄이려면 다음 작업을 수행하는 것이 좋습니다.

- Aurora DB 클러스터 유지 관리는 트래픽이 적은 기간에 수행해야 합니다. 성능 개선 도우미를 사용하여 이러한 기간을 식별하여 유지 관리 기간을 올바르게 구성하세요. 성능 개선 도우미에 대한 자세한 내용은 Amazon RDS에서 성능 개선 도우미를 통한 DB 로드 모니터링을 참조하세요. DB 클러스터 유지 관리 기간에 대한 자세한 내용은 기본 DB 클러스터 유지 관리 기간 조정 섹션을 참조하세요.
- 지수 백오프와 지터를 지원하는 AWS SDK를 모범 사례로 사용하세요. 자세한 내용은 지수 백오프 및 지터를 참조하세요.

마이너 버전 업그레이드 수행 및 패치 적용 방법

마이너 버전 업그레이드 및 패치는 엄격한 테스트를 거친 후에만 AWS 리전에 공개됩니다. Aurora PostgreSQL은 업그레이드 및 패치를 출시하기 전에 마이너 커뮤니티 버전 출시 이후에 발생하는 알려진 보안 문제, 버그 및 기타 문제가 Aurora PostgreSQL 플릿 안정성을 저해하지 않도록 테스트합니다.

Aurora PostgreSQL에 마이너 버전 자동 업그레이드 사용이 켜져 있으면 지정된 유지 관리 기간 동안 DB 클러스터가 주기적으로 업그레이드됩니다. 마이너 버전 자동 업그레이드 사용 옵션이 Aurora PostgreSQL DB 클러스터의 모든 인스턴스에서 설정되어 있는지 확인합니다. 자동 마이너 버전 업그레이드를 설정하는 방법과 클러스터 및 인스턴스 수준에서 설정을 적용한 경우 작동 방식에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하세요.

다음과 같이 describe-db-instances AWS CLI 명령을 사용하여 모든 Aurora PostgreSQL DB 클러스터에 대한 마이너 버전 자동 업그레이드 사용 옵션 값을 확인합니다.

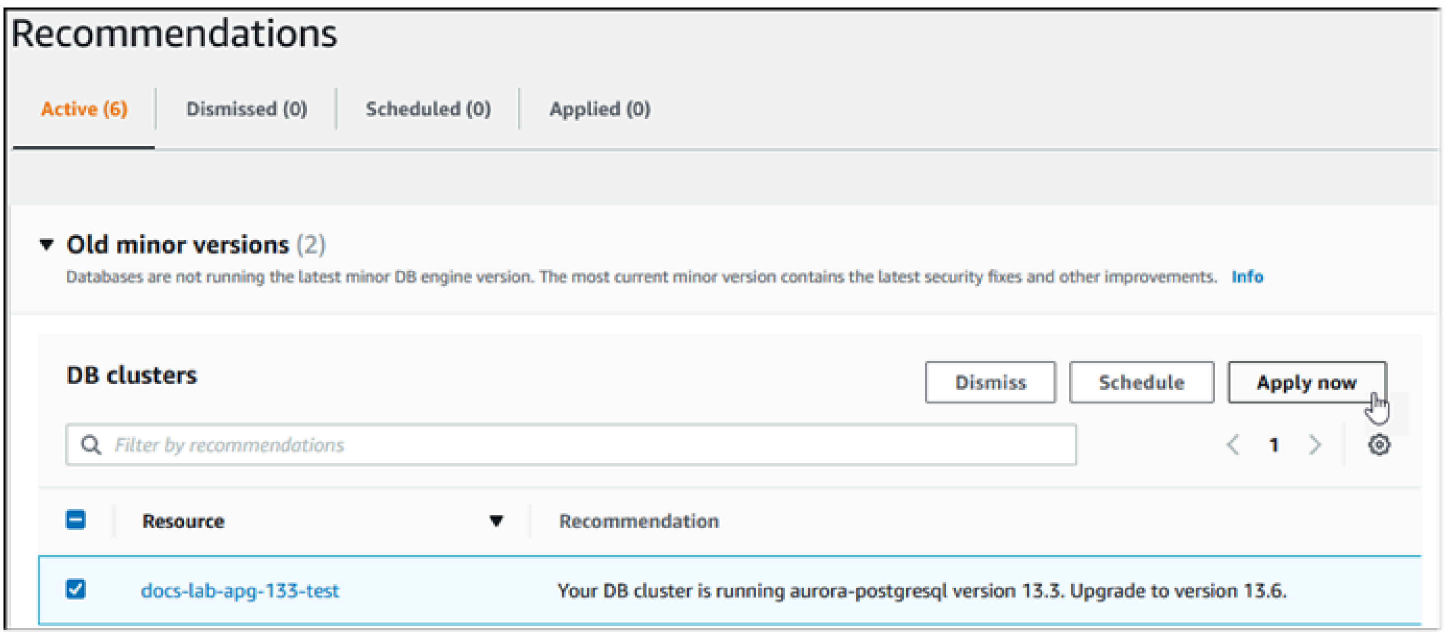
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

이 쿼리는 AutoMinorVersionUpgrade 설정 상태에 대한 true 또는 false 값과 함께 모든 Aurora DB 클러스터 및 해당 인스턴스 목록을 반환합니다. 명령에서는 기본 AWS 리전을 대상으로 삼도록 AWS CLI가 구성되어 있다고 가정합니다.

AmVU 옵션과 AmVU를 사용하도록 Aurora DB 클러스터를 수정하는 방법에 대한 자세한 내용은 Aurora DB 클러스터 마이너 버전 자동 업그레이드 섹션을 참조하세요.

유지 관리 태스크에 응답하거나 새 버전을 사용하도록 클러스터를 수정하여 Aurora PostgreSQL DB 클러스터를 새 마이너 버전으로 업그레이드할 수 있습니다.

RDS 콘솔을 사용하고 권장 사항(Recommendations) 메뉴를 열어 Aurora PostgreSQL DB 클러스터에 사용 가능한 업그레이드 또는 패치를 식별할 수 있습니다. 여기에는 이전 마이너 버전(Old minor versions)과 같은 다양한 유지 관리 문제 목록이 나와 있습니다. 프로덕션 환경에 따라 다음과 같이 업그레이드를 예약하거나 지금 적용(Apply now)을 선택하여 즉각적인 조치를 취할 수 있습니다.



패치 및 마이너 버전 업그레이드를 수동으로 적용하는 방법을 포함하여 Aurora DB 클러스터를 유지 관리하는 방법을 자세히 알아보려면 Amazon Aurora DB 클러스터 유지 관리 섹션을 참조하세요.

마이너 릴리스 업그레이드 및 제로 가동 중지 패치 적용

Aurora PostgreSQL DB 클러스터를 업그레이드하면 중단이 발생할 수 있습니다. 업그레이드 프로세스 중에 데이터베이스가 업그레이드될 때 데이터베이스가 종료됩니다. 데이터베이스가 사용 중인 동

안 업그레이드를 시작하면 DB 클러스터에서 처리하는 모든 연결 및 트랜잭션이 손실됩니다. 업그레이드를 수행하기 위해 데이터베이스가 유휴 상태가 될 때까지 기다리려면 오랜 시간을 기다려야 할 수 있습니다.

제로 가동 중지 패치 적용(ZDP) 기능은 업그레이드 프로세스를 개선합니다. ZDP를 사용하면 Aurora PostgreSQL DB 클러스터에 미치는 영향을 최소화하면서 마이너 버전 업그레이드와 패치를 모두 적용할 수 있습니다. Aurora PostgreSQL 버전 및 해당 마이너 버전의 더 높은 버전 릴리스와 더 최신의 메이저 버전 릴리스에 패치 또는 더 최신의 마이너 버전 업그레이드를 적용할 때 ZDP가 사용됩니다. 즉, 이러한 릴리스에서 새 마이너 버전으로 업그레이드하면 ZDP가 적용됩니다.

다음 테이블에는 ZDP를 사용할 수 있는 Aurora PostgreSQL 버전과 DB 인스턴스 클래스가 나와 있습니다.

버전	db.r* instance classes	db.t* instance classes	db.x* instance classes	db.serverless instance class
10.21 이상 버전	예	예	예	N/A
11.16 이상 버전	예	예	예	N/A
11.17 이상 버전	예	예	예	N/A
12.11 이상 버전	예	예	예	N/A
12.12 이상 버전	예	예	예	N/A
13.7 이상 버전	예	예	예	N/A
13.8 이상 버전	예	예	예	예
14.3 이상 버전	예	예	예	N/A
14.4 이상 버전	예	예	예	N/A
14.5 이상 버전	예	예	예	예
15.3 이상 버전	예	예	예	예
16.1 이상 버전	예	예	예	예

ZDP를 사용한 업그레이드 프로세스 중에 데이터베이스 엔진은 모든 새 트랜잭션을 일시 중지할 조용한 지점을 찾습니다. 이 작업은 패치 및 업그레이드 중에 데이터베이스를 보호합니다. 트랜잭션이 일시 중지된 상태에서 애플리케이션이 원활하게 실행되도록 하려면 재시도 로직을 코드에 통합하는 것이 좋습니다. 이 접근 방식을 통해 시스템은 짧은 가동 중지 시간을 장애 없이 관리할 수 있으며 업그레이드 후 새 트랜잭션을 재시도할 수 있습니다.

ZDP가 성공적으로 완료되면 애플리케이션 세션이 유지되며(연결이 끊긴 세션은 제외), 업그레이드가 계속 진행되는 동안 데이터베이스 엔진이 다시 시작됩니다. 데이터베이스 엔진을 다시 시작하면 처리량이 일시적으로 줄어들 수 있지만, 이러한 상황은 보통 몇 초나 최대 약 1분 정도만 지속됩니다.

경우에 따라 제로 가동 중지 패치 적용(ZDP)이 적용되지 않을 수 있습니다. 예를 들어, Aurora PostgreSQL DB 클러스터 또는 해당 인스턴스의 `pending` 상태에 있는 파라미터 변경으로 인해 ZDP가 제대로 적용되지 않을 수 있습니다.

ZDP 작업에 대한 지표와 이벤트는 콘솔의 이벤트(Events) 페이지에서 확인할 수 있습니다. 이벤트에는 ZDP 업그레이드 시작 및 업그레이드 완료가 포함됩니다. 여기에서 프로세스의 소요 시간 및 재시작 중에 발생한 보존 및 삭제된 연결 수를 확인할 수 있습니다. 데이터베이스 오류 로그에 자세한 내용이 나와 있습니다.

제로 가동 중지 패치 적용의 제한 사항

제로 가동 중지 패치 적용에는 다음 제한 사항이 있습니다.

- ZDP는 Aurora PostgreSQL 업그레이드 프로세스 전반에 걸쳐 Aurora PostgreSQL 라이터 인스턴스에 대한 현재 클라이언트 연결을 유지하려고 합니다. 그러나 다음과 같은 경우에는 ZDP 완료를 위해 연결이 끊어집니다.
  - 장기 실행 쿼리 또는 트랜잭션이 진행 중인 경우.
  - 데이터 정의 언어(DDL) 문이 실행 중인 경우.
  - 임시 테이블 또는 테이블 잠금이 사용 중인 경우.
  - 모든 세션이 알림 채널에서 수신 중인 경우.
  - 커서가 'WITH HOLD' 상태에서 사용 중인 경우.
  - TLSv1.1 연결이 사용 중인 경우. Aurora PostgreSQL 버전 16.1, 15.3, 14.8, 13.11, 12.15, 11.20 이상부터 ZDP에는 TLSv1.3 연결이 지원됩니다.
- 다음과 같은 경우에는 ZDP가 지원되지 않습니다.
  - Aurora PostgreSQL DB 클러스터가 Aurora Serverless v1로 구성된 경우.
  - Aurora 리더 인스턴스를 업그레이드하는 동안.

- 보조 리전의 Aurora Global Database 클러스터에 속하는 Aurora 리더 인스턴스를 업그레이드하는 동안.
- OS 패치 및 OS 업그레이드 중.

Aurora PostgreSQL 엔진을 새로운 마이너 버전으로 업그레이드

콘솔, AWS CLI 또는 RDS API를 사용하여 Aurora PostgreSQL DB 클러스터를 새 마이너 버전으로 업그레이드할 수 있습니다. 업그레이드를 수행하려면 메이저 버전 업그레이드에 권장되는 것과 동일한 모범 사례를 따르는 것이 좋습니다. 새 메이저 버전과 마찬가지로 새 마이너 버전에도 최적화 프로그램 개선 사항 (예: 수정 사항) 이 추가되어 쿼리 계획 회귀가 발생할 수 있습니다. 계획 안정성을 보장하려면 메이저 버전 업그레이드 후 계획 안정성 보장에 자세히 설명된 대로 쿼리 계획 관리(QPM) 확장을 사용하는 것이 좋습니다.

콘솔

Aurora PostgreSQL DB 클러스터의 엔진 버전을 업그레이드하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 업그레이드하려는 DB 클러스터를 선택합니다.
3. 수정을 선택합니다. DB 클러스터 수정 페이지가 나타납니다.
4. DB 엔진 버전에서 새 버전을 선택합니다.
5. 계속을 수정 사항을 요약한 내용을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다. 일부의 경우 이 옵션을 선택하면 중단이 발생할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.
7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 올바른 경우 클러스터 수정을 선택하여 변경 내용을 저장합니다.

   또는 뒤로를 선택하여 변경 내용을 편집하거나 취소를 선택하여 변경 내용을 취소합니다.

AWS CLI

DB 클러스터의 엔진 버전을 업그레이드하려면 다음 파라미터와 함께 modify-db-cluster AWS CLI 명령을 사용합니다.

- `--db-cluster-identifier` - Aurora PostgreSQL DB 클러스터의 이름입니다.

- `--engine-version` – 업그레이드할 데이터베이스 엔진의 버전 번호입니다. 유효한 엔진 버전에 대한 정보를 보려면 AWS CLI describe-db-engine-versions 명령을 사용합니다.
- `--no-apply-immediately` – 변경 사항이 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 대신 `--apply-immediately`를 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --no-apply-immediately
```

RDS API

DB 클러스터의 엔진 버전을 업그레이드하려면 ModifyDBCluster 작업을 사용합니다. 다음 파라미터를 지정합니다.

- `DBClusterIdentifier` – DB 클러스터의 이름입니다(예:*mydbcluster*).
- `EngineVersion` – 업그레이드할 데이터베이스 엔진의 버전 번호입니다. 유효한 엔진 버전에 대한 정보를 보려면 DescribeDBEngineVersions 작업을 사용합니다.
- `ApplyImmediately` – 변경 사항을 즉시 적용하거나 다음 유지 관리 기간에 적용합니다. 변경 사항을 바로 적용하려면 값을 `true`로 설정합니다. 변경 사항을 다음 유지 관리 기간에 적용하려면 값을 `false`로 설정합니다.

PostgreSQL 확장 버전 업그레이드

Aurora PostgreSQL DB 클러스터를 새로운 메이저 또는 마이너 버전으로 업그레이드해도 PostgreSQL 확장이 동시에 업그레이드되지 않습니다. 대부분의 확장은 메이저 또는 마이너 버전 업그레이드가 완료된 후에 따로 업그레이드해야 합니다. 그러나 경우에 따라 Aurora PostgreSQL DB 엔진을 업그레이드하기 전에 확장을 업그레이드해야 할 수 있습니다. 자세한 내용은 새로운 메이저 버전으로 프로덕션 DB 클러스터 업그레이드 테스트의 list of extensions to update 섹션을 참조하세요.

PostgreSQL 확장을 설치하려면 `rds_superuser` 권한이 필요합니다. 일반적으로 `rds_superuser`는 특정 확장에 대한 권한을 관련 사용자(역할)에게 위임하여 해당 확장을 용이하게 관리할 수 있도록 합니다. 즉, Aurora PostgreSQL DB 클러스터의 모든 확장을 업그레이드하는 태스크에는 다양한 사용자(역할)가 관여해야 할 수 있습니다. 특히 스크립트를 사용하여 업그레이드 프로세스를 자동화하려는 경우에는 이 점을 염두에 두세요. PostgreSQL 권한 및 역할에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 사용한 보안 섹션을 참조하세요.

> **Note**
>
> PostGIS 확장 프로그램 업데이트에 대한 자세한 내용은 PostGIS 확장을 사용하여 공간 데이터 관리(6단계: PostGIS 확장 업그레이드) 섹션을 참조하세요.
>
> `pg_repack` 확장을 업데이트하려면 확장을 삭제한 후 업그레이드된 DB 인스턴스에 새 버전을 생성합니다. 자세한 내용은 `pg_repack` 설명서의 pg_repack 설치를 참조하세요.

엔진 업그레이드 후 확장 버전을 업데이트하려면 `ALTER EXTENSION UPDATE` 명령을 사용합니다.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

현재 설치된 확장을 나열하려면 다음 명령에서 PostgreSQL pg_extension 카탈로그를 사용합니다.

```
SELECT * FROM pg_extension;
```

설치에 사용할 수 있는 특정 확장 버전의 목록을 보려면 다음 명령에서 PostgreSQL pg_available_extension_versions 보기를 사용하십시오.

```
SELECT * FROM pg_available_extension_versions;
```

대체 블루/그린 업그레이드 기법

경우에 따라서는 오래된 클러스터에서 업그레이드된 클러스터로 즉시 전환하는 것이 가장 중요한 우선순위일 수 있습니다. 또는 이전 클러스터와 새 클러스터를 나란히 실행하는 다단계 프로세스를 따를 수도 있습니다. 이 경우 새 클러스터가 인수할 준비가 될 때까지 이전 클러스터의 데이터를 새 클러스터로 복제합니다. 자세한 내용은 데이터베이스 업데이트에 블루/그린 배포 사용 섹션을 참조하십시오.

Aurora PostgreSQL 장기 지원(LTS) 릴리스 사용

DB 클러스터를 생성하거나 업그레이드할 때 각 새 Aurora PostgreSQL 버전을 일정 시간 동안 사용할 수 있습니다. 이 기간이 지나면 해당 버전을 사용하는 모든 클러스터를 업그레이드해야 합니다. 지원

기간이 끝나기 전에 클러스터를 수동으로 업그레이드하거나 Aurora PostgreSQL 버전이 더 이상 지원되지 않는 경우 Aurora에서 자동으로 클러스터를 업그레이드할 수 있습니다.

Aurora는 특정 Aurora PostgreSQL 버전을 LTS(장기 지원) 릴리스로 지정합니다. LTS 릴리스를 사용하는 데이터베이스 클러스터는 비 LTS 릴리스를 사용하는 클러스터보다 동일한 버전을 더 오래 유지하고 더 적은 업그레이드 주기를 거칠 수 있습니다. LTS 마이너 버전에는 패치 버전을 통한 버그 수정만 포함합니다. LTS 버전에는 도입 후 릴리스된 새로운 기능은 포함하지 않습니다.

일 년에 한 번 LTS 마이너 버전에서 실행되는 DB 클러스터는 LTS 릴리스의 최신 패치 버전으로 패치됩니다. 누적 보안 및 안정성 수정 프로그램의 혜택을 누릴 수 있도록 이 패치 작업을 수행합니다. 보안과 같이 적용해야 하는 중요한 수정 사항이 있는 경우 LTS 마이너 버전을 더 자주 패치할 수 있습니다.

> **Note**
>
> 수명 주기 동안 LTS 마이너 버전을 계속 사용하려면 해당 DB 인스턴스에 대해 자동 마이너 버전 업그레이드를 비활성화합니다. LTS 마이너 버전에서 DB 클러스터를 자동으로 업그레이드하지 않으려면 Aurora 클러스터의 모든 DB 인스턴스에서 자동 마이너 버전 업그레이드 활성화 확인란의 선택을 취소합니다.

대부분의 Aurora PostgreSQL 클러스터에서 LTS 릴리스를 사용하는 대신 최신 릴리스로 업그레이드하는 것이 좋습니다. 이렇게 하면 Aurora를 관리형 서비스로 활용하고 최신 기능 및 버그 수정에 액세스할 수 있습니다. LTS 릴리스는 다음과 같은 특성을 가진 클러스터 전용입니다.

- 중요 패치의 경우 드문 경우를 제외하고는 업그레이드에 대한 Aurora PostgreSQL 애플리케이션의 가동 중지를 감당할 수 없는 경우
- Aurora PostgreSQL 데이터베이스 엔진을 업데이트할 때마다 클러스터 및 관련 애플리케이션의 테스트 주기 시간이 오래 걸리는 경우
- Aurora PostgreSQL 클러스터의 데이터베이스 버전에 애플리케이션에 필요한 모든 DB 엔진 기능과 버그 수정이 있는 경우

Aurora PostgreSQL에 대한 기존 LTS 릴리스는 다음과 같습니다.

- PostgreSQL 15.10 2024년 12월 27일에 릴리스되었습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 PostgreSQL 15.10을 참조하세요.
- PostgreSQL 14.6. 2023년 1월 20일에 릴리스되었습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 PostgreSQL 14.6을 참조하세요.

- PostgreSQL 13.9. 2023년 1월 20일에 릴리스되었습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 PostgreSQL 13.9를 참조하세요.
- PostgreSQL 12.9. 2022년 2월 25일에 릴리스되었습니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 PostgreSQL 12.9를 참조하세요.
- PostgreSQL 11.9(Aurora PostgreSQL 릴리스 3.4. 2020년 12월 11일에 릴리스되었습니다. 이 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보의 PostgreSQL 11.9, Aurora PostgreSQL 릴리스 3.4를 참조하세요.

LTS 버전의 지원 일정 및 릴리스 주기에 대한 자세한 내용은 Aurora PostgreSQL의 릴리스 일정을 참조하세요.

Aurora 및 데이터베이스 엔진 버전을 식별하는 방법에 대한 자세한 내용은 Amazon Aurora PostgreSQL 버전 식별 섹션을 참조하세요.

Amazon Aurora PostgreSQL Limitless Database 사용

Amazon Aurora PostgreSQL Limitless Database는 자동화된 수평적 스케일링을 제공하여 초당 수백만 개의 쓰기 트랜잭션을 처리하고 페타바이트 데이터를 관리하면서 단일 데이터베이스 내에서 운영할 때의 단순성은 유지합니다. Aurora PostgreSQL Limitless Database를 사용하면 워크로드를 지원하기 위해 여러 DB 인스턴스에서 데이터를 확장하기 위한 복잡한 솔루션을 구축하고 유지 관리할 필요 없이 대규모 애플리케이션을 구축하는 데 집중할 수 있습니다.

주제

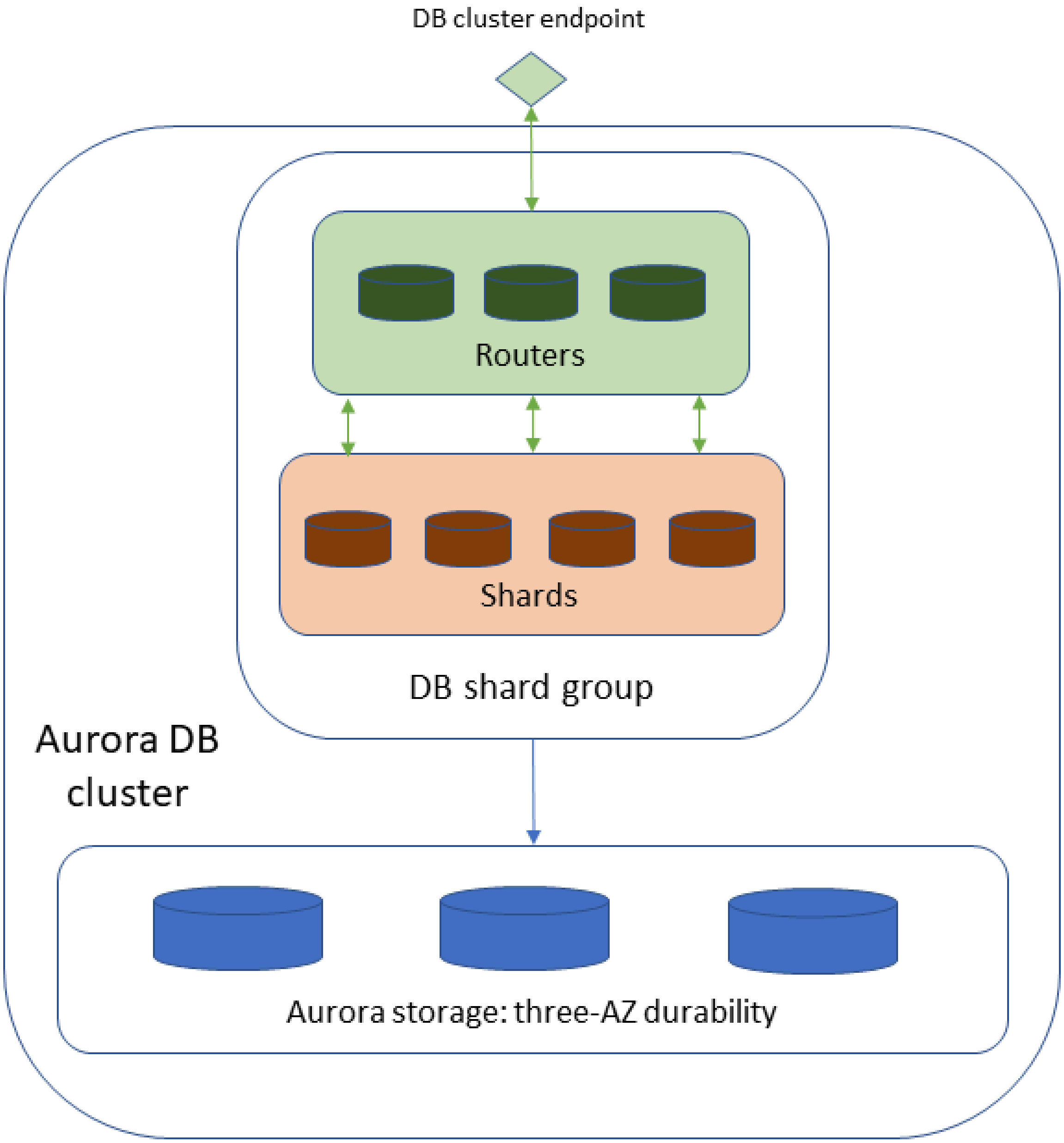
Aurora PostgreSQL Limitless Database 아키텍처

Limitless Database는 여러 데이터베이스 노드로 구성된 2계층 아키텍처로 확장할 수 있습니다. 노드는 라우터 또는 샤드입니다.

- 샤드는 각각 데이터베이스의 데이터 하위 집합을 저장하는 Aurora PostgreSQL DB 인스턴스로, 동시 처리를 통해 쓰기 처리량을 높일 수 있습니다.
- 라우터는 데이터베이스의 분산되어 있는 속성을 관리하고 데이터베이스 클라이언트에 단일 데이터베이스 이미지를 제공합니다. 라우터는 데이터가 저장되는 위치에 대한 메타데이터를 유지하고, 들어오는 SQL 명령을 구문 분석하고, 이러한 명령을 샤드로 전송합니다. 그런 다음 샤드의 데이터를 집계하여 단일 결과를 클라이언트에 반환하고 분산 트랜잭션을 관리하여 전체 분산 데이터베이스에서 일관성을 유지합니다.

Aurora PostgreSQL Limitless Database는 라이터 DB 인스턴스 및 리더 DB 인스턴스 대신 DB 샤드 그룹을 보유하므로 표준 Aurora DB 클러스터과 다릅니다. Limitless Database 아키텍처를 구성하는 모든 노드는 DB 샤드 그룹에 포함됩니다. DB 샤드 그룹의 개별 샤드 및 라우터는 AWS 계정에 표시되지 않습니다. DB 클러스터 엔드포인트를 사용하여 Limitless Database에 액세스합니다.

다음 그림은 Aurora PostgreSQL Limitless Database의 개괄적인 아키텍처를 보여줍니다.



Aurora PostgreSQL Limitless Database의 아키텍처와 이를 사용하는 방법에 대한 자세한 내용은 YouTube의 AWS Event 채널에서 Achieving scale with Amazon Aurora PostgreSQL Limitless Database를 참조하세요.

표준 Aurora DB 클러스터의 아키텍처에 대한 자세한 내용은 Amazon Aurora DB 클러스터 섹션을 참조하세요.

Aurora PostgreSQL Limitless Database의 주요 용어

DB 샤드 그룹

Limitless Database 노드(샤드 및 라우터)용 컨테이너입니다.

라우터

클라이언트에서 SQL 연결을 수락하고, SQL 명령을 샤드로 전송하고, 시스템 전반의 일관성을 유지하고, 결과를 클라이언트에 반환하는 노드입니다.

샤드

샤딩된 테이블의 하위 집합, 참조 테이블의 전체 사본 및 표준 테이블을 저장하는 노드입니다. 라우터의 쿼리를 수락하지만 클라이언트에 의해 직접 연결할 수는 없습니다.

샤딩된 테이블

데이터가 샤드에 파티셔닝되어 있는 테이블입니다.

샤드 키

샤드 간 파티셔닝을 결정하는 데 사용되는 샤딩된 테이블의 열 또는 열 집합입니다.

공동 배치된 테이블

동일한 샤드 키를 공유하고 명시적으로 공동 배치된 것으로 선언된 두 개의 샤딩된 테이블입니다. 동일한 샤드 키 값에 대한 모든 데이터는 동일한 샤드로 전송됩니다.

참조 테이블

모든 샤드에 전체 데이터가 복사된 테이블입니다.

표준 테이블

Limitless Database의 기본 테이블 유형입니다. 표준 테이블을 샤딩 및 참조 테이블로 변환할 수 있습니다.

모든 표준 테이블은 시스템에서 선택한 동일한 샤드에 저장되므로 단일 샤드 내에서 표준 테이블 간의 조인을 수행할 수 있습니다. 그러나 표준 테이블은 샤드의 최대 용량(128TiB)으로 제한됩니다. 또한 이 샤드는 샤딩된 테이블과 참조 테이블의 데이터를 저장하므로 표준 테이블의 유효 한도는 128TiB보다 낮습니다.

Aurora PostgreSQL Limitless Database의 테이블 유형

Aurora PostgreSQL Limitless Database는 샤딩된 테이블, 참조 테이블 및 표준 테이블의 세 가지 테이블 유형을 지원합니다.

샤딩된 테이블의 데이터는 DB 샤드 그룹의 모든 샤드에 분산되어 있습니다. Limitless Database는 테이블을 파티셔닝할 때 지정하는 열 또는 열 집합인 샤드 키를 사용하여 이를 자동으로 수행합니다. 샤드 키 값이 동일한 모든 데이터는 동일한 샤드로 전송됩니다. 샤딩은 범위 또는 목록 기반이 아닌 해시 기반입니다.

다음은 샤딩된 테이블의 좋은 사용 사례입니다.

- 애플리케이션이 고유한 데이터 하위 집합으로 작동하는 경우
- 테이블이 매우 큰 경우
- 테이블이 다른 테이블보다 더 빠르게 커질 수 있는 경우

샤딩된 테이블을 공동 배치될 수 있습니다. 즉, 동일한 샤드 키를 공유하며 두 테이블에서 샤드 키 값이 동일한 모든 데이터가 동일한 샤드로 전송됩니다. 테이블을 공동 배치하고 샤드 키를 사용하여 조인하는 경우 필요한 모든 데이터가 해당 샤드에 존재하므로 단일 샤드에서 조인을 수행할 수 있습니다.

참조 테이블은 모든 데이터의 전체 복사본이 DB 샤드 그룹의 모든 샤드에 있습니다. 참조 테이블은 일반적으로 쓰기 볼륨이 낮은 작은 테이블이지만 여전히 자주 조인해야 하고 샤딩에 적합하지 않은 경우에 자주 사용됩니다. 참조 테이블의 예로는 날짜 테이블과 주, 도시, 우편 번호 등의 지리적 데이터 테이블이 있습니다.

표준 테이블은 Aurora PostgreSQL Limitless Database의 기본 테이블 유형입니다. 분산 테이블이 아닙니다. Aurora PostgreSQL Limitless Database는 표준 테이블과 표준 테이블, 샤딩된 테이블, 참조 테이블 간의 조인을 지원합니다.

Aurora PostgreSQL Limitless Database의 요금 청구

Aurora PostgreSQL Limitless Database 요금이 청구되는 방법에 대한 자세한 내용은 Aurora에 대한 DB 인스턴스 결제 섹션을 참조하세요.

Aurora요금에 대한 자세한 내용은 Aurora 요금 페이지를 참조하십시오.

Aurora PostgreSQL Limitless Database 시작하기

Aurora PostgreSQL Limitless Database를 시작하려면 다음 작업을 수행합니다.

1. Aurora PostgreSQL DB 클러스터와 Limitless Database용 DB 샤드 그룹을 만듭니다. 자세한 내용은 Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 만들기 단원을 참조하십시오.
2. DB 샤드 그룹에서 샤딩된 테이블과 참조 테이블을 만듭니다. 자세한 내용은 Aurora PostgreSQL Limitless Database 테이블 만들기 단원을 참조하십시오.
3. DB 샤드 그룹의 용량을 변경하고, 샤드를 분할하고, 라우터를 추가합니다. 자세한 내용은 DB 샤드 그룹 사용 단원을 참조하십시오.
4. DB 샤드 그룹에 데이터를 로드합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database에 데이터 로드 단원을 참조하십시오.
5. DB 샤드 그룹에서 쿼리 및 기타 SQL 문을 실행합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database 쿼리 단원을 참조하십시오.
6. Limitless Database의 성능을 모니터링합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database 모니터링 단원을 참조하십시오.

Aurora PostgreSQL Limitless Database 요구 사항 및 고려 사항

Aurora PostgreSQL Limitless Database에는 다음과 같은 요구 사항과 고려 사항이 있습니다.

주제

Aurora PostgreSQL Limitless Database 요구 사항

Aurora PostgreSQL Limitless Database에 대한 다음 요구 사항을 따라야 합니다.

- 다음 AWS 리전을 사용할 수 있습니다.
  - 아시아 태평양(홍콩)
  - 아시아 태평양(싱가포르)
  - 아시아 태평양(시드니)
  - 아시아 태평양(도쿄)
  - 유럽(프랑크푸르트)
  - 유럽(아일랜드)
  - 유럽(스톡홀름)
  - 미국 동부(버지니아 북부)

    > Note
    >
    > 이 AWS 리전에서 Limitless Database DB 클러스터를 만드는 경우 DB 서브넷 그룹에 `us-east-1e` 가용 영역(AZ)을 포함하지 마세요. 리소스 제한으로 인해 `us-east-1e` AZ에서는 Aurora Serverless v2가 지원되지 않고, 따라서 Limitless Database도 지원되지 않습니다.

  - 미국 동부(오하이오)
  - 미국 서부(오리건)

이 안내서의 AWS CLI 프로시저에서는 사용 가능한 AWS 리전 중 하나를 사용한다고 가정합니다. AWS CLI의 기본 AWS 리전을 설정하는 방법에 대한 자세한 내용은 AWS Command Line Interface 버전 2 사용 설명서의 환경 변수를 설정하는 방법을 참조하세요.

- Aurora PostgreSQL Limitless Database는 Aurora I/O-Optimized DB 클러스터 스토리지 구성만 지원합니다. 자세한 내용은 Amazon Aurora DB 클러스터의 스토리지 구성 섹션을 참조하세요.
- Aurora PostgreSQL Limitless Database는 다음과 같은 Aurora PostgreSQL Limitless Database용 특수 Aurora PostgreSQL DB 엔진 버전을 사용합니다.
  - `16.4-limitless`
  - `16.6-limitless`
- DB 클러스터에는 라이터 또는 리더 DB 인스턴스가 있을 수 없습니다.
- 향상된 모니터링 및 성능 개선 도우미를 사용해야 합니다. 성능 개선 도우미 보존 기간은 최소 1개월 (31일)이어야 합니다.
- PostgreSQL 로그를 Amazon CloudWatch Logs로 내보내야 합니다.

> **Note**
>
> 향상된 모니터링, 성능 개선 도우미 및 CloudWatch Logs와 같은 일부 필수 기능에는 추가 요금이 부과됩니다. Aurora요금에 대한 자세한 내용은 Aurora 요금 페이지를 참조하십시오.

Aurora PostgreSQL Limitless Database 고려 사항

Aurora PostgreSQL Limitless Database의 DB 샤드 그룹에는 다음 고려 사항이 적용됩니다.

- DB 클러스터당 하나의 DB 샤드 그룹만 가질 수 있습니다.
- AWS 리전마다 최대 5개의 DB 샤드 그룹을 가질 수 있습니다.

  따라서 AWS 리전마다 최대 5개의 Aurora PostgreSQL Limitless Database DB 클러스터를 가질 수 있습니다. 자세한 내용은 Amazon Aurora의 할당량 섹션을 참조하세요.
- DB 샤드 그룹의 최대 용량을 16~6,144ACU로 설정할 수 있습니다. 6,144ACU보다 높은 용량 한도는 AWS에 문의하세요.

  초기 라우터 및 샤드 수는 DB 샤드 그룹을 만들 때 설정한 최대 용량에 따라 결정됩니다. 자세한 내용은 DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결 섹션을 참조하세요.
- DB 샤드 그룹의 최대 용량을 수정할 때 라우터 및 샤드 수는 변경되지 않습니다.

- DB 샤드 그룹을 만드는 DB 서브넷에 DB 샤드 그룹과 연결하기에 충분한 여유 IP 주소가 있는지 확인합니다. DB 샤드 그룹의 각 라우터에는 하나의 IP 주소가 필요하고 각 샤드에는 최대 3개의 IP 주소가 필요합니다.

  DB 샤드 그룹을 만들 때 만들어진 라우터 수에 대한 자세한 내용은 DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결 섹션을 참조하세요.
- DB 샤드 그룹에 공개적으로 액세스할 수 있도록 하면 VPC에 인터넷 게이트웨이를 설정해야 합니다.
- SQL 함수를 사용하여 샤드를 분할하고 라우터를 추가합니다.
- 샤드 병합은 지원되지 않습니다.
- 개별 샤드와 라우터는 삭제할 수 없습니다.
- 테이블 행에서 값을 변경하는 등 어떤 식으로든 샤드 키를 수정(UPDATE 작업 수행)할 수 없습니다.

  샤드 키를 변경하려면 삭제한 다음 다시 만듭니다.
- 반복 가능한 읽기, 읽기 커밋됨 및 읽기 커밋되지 않음의 격리 수준이 지원됩니다. 격리 수준을 직렬화 가능으로 설정할 수 없습니다.
- 일부 SQL 명령은 지원되지 않습니다. 자세한 내용은 Aurora PostgreSQL Limitless Database 참조 섹션을 참조하세요.
- 일부 PostgreSQL 확장은 지원되지 않습니다. 자세한 내용은 확장 프로그램 섹션을 참조하세요.
- 샤드 그룹을 만들거나 새 샤드 그룹 노드(샤드 또는 라우터)를 추가할 때 해당 노드는 DB 클러스터에서 사용할 수 있는 가용 영역(AZ) 중 하나에 만들어집니다. 개별 노드에 대해 특정 AZ를 선택할 수 없습니다.
- 2개의 컴퓨팅 중복성을 사용하는 경우(DB 샤드 그룹에 대해 2개의 컴퓨팅 대기) DB 서브넷 그룹에 AZ가 3개 이상 있는지 확인합니다.
- Aurora PostgreSQL Limitless Database는 샤딩된 테이블 이름에 최대 54자를 지원합니다.

Aurora PostgreSQL Limitless Database DB 클러스터에는 다음 고려 사항이 적용됩니다.

- AWS 관리형 정책을 사용하여 데이터베이스 및 애플리케이션에 대한 권한을 고객 사용 사례에 필요한 권한으로 제한하는 것이 좋습니다. 자세한 내용은 정책 모범 사례 섹션을 참조하세요.
- Aurora PostgreSQL Limitless Database DB 클러스터를 만들 때 DB 샤드 그룹에 대해서만 크기 조정 파라미터를 설정합니다.
- DB 샤드 그룹이 포함된 DB 클러스터는 중지하거나 시작할 수 없습니다.
- DB 클러스터를 삭제해야 하는 경우 먼저 DB 샤드 그룹을 삭제해야 합니다.

- Aurora PostgreSQL Limitless Database는 복제 소스가 될 수 없습니다.

Aurora PostgreSQL Limitless Database에서 지원되지 않는 기능

다음 Aurora PostgreSQL 기능은 Aurora PostgreSQL Limitless Database에서 지원되지 않습니다.

- Active Directory(Kerberos) 인증
- Amazon DevOps Guru
- Amazon ElastiCache
- Amazon RDS 블루/그린 배포
- Amazon RDS Proxy
- Aurora Auto Scaling(DB 클러스터에 리더 인스턴스 자동 추가)
- Aurora Global Database
- Aurora Machine Learning
- Aurora 권장 사항
- Aurora Serverless v1
- Aurora 제로 ETL 통합
- AWS Backup
- AWS Lambda 통합
- AWS Secrets Manager
- Babelfish for Aurora PostgreSQL
- DB 클러스터 복제본 생성
- 사용자 지정 엔드포인트
- 데이터베이스 활동 스트림
- 읽기 전용 복제본
- RDS Data API

Aurora PostgreSQL Limitless Database 사용을 위한 사전 조건

Aurora PostgreSQL Limitless Database를 사용하려면 먼저 다음 작업을 수행해야 합니다.

주제

- DB 샤드 그룹 작업 활성화

DB 샤드 그룹 작업 활성화

DB 샤드 그룹을 만들려면 먼저 DB 샤드 그룹 작업을 활성화해야 합니다.

- Aurora PostgreSQL Limitless Database에 액세스하는 사용자의 IAM 역할의 IAM 정책에 다음 섹션을 추가합니다.

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDBShardGroup",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBShardGroup",
                "rds:DescribeDBShardGroups",
                "rds:DeleteDBShardGroup",
                "rds:ModifyDBShardGroup",
                "rds:RebootDBShardGroup"
            ],
            "Resource": [
                "arn:aws:rds:*:*:shard-group:*",
                "arn:aws:rds:*:*:cluster:*"
            ]
        }
    ]
}
```

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 만들기

Aurora PostgreSQL의 Limitless Database 버전을 사용하여 새 Aurora DB 클러스터를 만들고 클러스터에 DB 샤드 그룹을 추가합니다. DB 샤드 그룹을 추가할 때 전체 DB 샤드 그룹의 최대 컴퓨팅 용량(모든 라우터 및 샤드의 용량 합계)을 Aurora 용량 단위(ACU)로 지정합니다. 각 ACU는 약 2기가바이트(GB)의 메모리로 해당 CPU 및 네트워킹의 합한 용량입니다. 크기 조정은 애플리케이션 워크로드에 따라 DB 샤드 그룹의 용량을 Aurora Serverless v2의 작동 방식과 유사하게 늘리거나 줄입니다.

주제

DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결

DB 샤드 그룹에 속한 초기 라우터 및 샤드 수는 DB 샤드 그룹을 만들 때 설정한 최대 용량에 따라 결정됩니다. 최대 용량이 높을수록 DB 샤드 그룹에서 만들어지는 라우터 및 샤드가 많아집니다.

각 노드(샤드 또는 라우터)에는 저마다 현재 용량 값이 있으며 단위는 ACU입니다.

- Limitless Database는 현재 용량이 너무 낮아 로드를 처리할 수 없는 경우 노드를 더 높은 용량으로 스케일 업합니다. 그러나 총 용량이 최대일 때는 노드가 스케일 업되지 않습니다.
- Limitless Database는 현재 용량이 필요 이상으로 높으면 노드를 더 낮은 용량으로 스케일 다운합니다. 그러나 총 용량이 최소일 때는 노드가 스케일 다운되지 않습니다.

다음 표는 Aurora 용량 단위(ACU)로 표시된 DB 샤드 그룹 최대 용량과 만들어진 노드(라우터 및 샤드) 수 간의 상관관계를 보여줍니다.

> **Note**
>
> 이러한 값은 변경될 수 있습니다.
> 컴퓨팅 중복을 0이 아닌 값으로 설정하면 총 샤드 수가 두 배 또는 세 배가 됩니다. 이로 인해 추가 비용이 발생합니다.
> 컴퓨팅 대기의 노드는 라이터와 동일한 용량으로 스케일 업 및 다운됩니다. 대기에 대해 용량 범위를 별도로 설정하지 않습니다.

총 노드 수	라우터	샤드	기본 최소 용량 (ACU)	최대 용량 범위 (ACU)
4	2	2	16	16~400
5	2	3	20	401~500
6	2	4	24	501~600
7	3	4	28	601~700
8	3	5	32	701~800
9	3	6	36	801~900
10	4	6	40	901~1,000
11	4	7	44	1,001~1,100
12	4	8	48	1,101~1,200
13	5	8	52	1,201~1,300
14	5	9	56	1,301~1,400
15	5	10	60	1,401~1,500
16	6	10	64	1,501~1,600
17	6	11	68	1,601~1,700
18	6	12	72	1,701~1,800
19	7	12	76	1,801~1,900
20	7	13	80	1,901~2,000
21	7	14	84	2,001~2,100
22	8	14	88	2,101~2,200
23	8	15	92	2,201~2,300

총 노드 수	라우터	샤드	기본 최소 용량 (ACU)	최대 용량 범위 (ACU)
24	8	16	96	2,301~6,144

DB 샤드 그룹의 최대 용량 기반 동적 구성은 만드는 중에만 사용할 수 있습니다. 최대 용량이 수정될 때 라우터와 샤드 수는 동일하게 유지됩니다. 자세한 내용은 DB 샤드 그룹의 용량 변경 단원을 참조하십시오.

SQL 명령을 사용하여 DB 샤드 그룹에 샤드와 라우터를 추가할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- DB 샤드 그룹에서 샤드 분할
- DB 샤드 그룹에 라우터 추가

> **Note**
> 샤드 또는 라우터는 삭제할 수 없습니다.

DB 클러스터 만들기

다음 프로시저에 따라 Aurora PostgreSQL Limitless Database를 사용하는 Aurora PostgreSQL DB 클러스터를 만듭니다.

AWS Management Console 또는 AWS CLI를 사용하여 Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터를 만들 수 있습니다. 기본 DB 클러스터와 DB 샤드 그룹을 만듭니다.

콘솔

AWS Management Console을 사용하여 기본 DB 클러스터를 만들 때 DB 샤드 그룹도 동일한 프로시저로 만들어집니다.

콘솔을 사용하여 DB 클러스터를 만드는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스 생성을 선택합니다.

   데이터베이스 생성 페이지가 표시됩니다.
3. 엔진 유형에서 Aurora(PostgreSQL 호환)를 선택합니다.
4. 버전에서 다음 중 하나를 선택합니다.
   - Limitless Database가 있는 Aurora PostgreSQL(PostgreSQL 16.4와 호환)
   - Limitless Database가 있는 Aurora PostgreSQL(PostgreSQL 16.6과 호환)
5. Aurora PostgreSQL Limitless Database의 경우:

a. DB 샤드 그룹 식별자를 입력합니다.

> **Important**
>
> DB 샤드 그룹을 만든 후에는 DB 클러스터 식별자 또는 DB 샤드 그룹 식별자를 변경할 수 없습니다.

b. DB 샤드 그룹 용량 범위의 경우:

   i. 최소 용량(ACU)을 입력합니다. 값을 최소 16ACU로 지정합니다.

      개발 환경의 경우 기본값은 16ACU입니다. 프로덕션 환경의 경우 기본값은 24ACU입니다.

   ii. 최대 용량(ACU)을 입력합니다. 최소 16ACU 또는 최대 6,144ACU의 값을 사용합니다.

      개발 환경의 경우 기본값은 64ACU입니다. 프로덕션 환경의 경우 기본값은 384ACU입니다.

   자세한 내용은 [DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결](#) 섹션을 참조하세요.

c. DB 샤드 그룹 배포의 경우 DB 샤드 그룹에 대한 대기를 만들지 선택합니다.

- 컴퓨팅 중복 없음 - 각 샤드에 대해 대기 없이 DB 샤드 그룹을 만듭니다. 이것이 기본값입니다.
- 단일 장애 조치 타겟을 사용한 컴퓨팅 중복 - 다른 가용 영역(AZ)에 하나의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.
- 2개의 장애 조치 타겟을 사용한 컴퓨팅 중복 - 2개의 서로 다른 AZ에서 2개의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.

> **Note**
>
> 컴퓨팅 중복을 0이 아닌 값으로 설정하면 DB 인스턴스의 총 샤드 수가 두 배 내지 세 배가 됩니다. 이러한 추가 DB 인스턴스는 컴퓨팅 대기 인스턴스로, 라이터 인스턴스와 동일한 용량으로 스케일 업 및 스케일 다운됩니다. 대기에 대해 용량 범위를 별도로 설정하지 않습니다. 그에 따라 ACU 사용량 및 청구 금액도 두 배, 세 배로 늘어납니다. 컴퓨팅 중복으로 인해 발생한 정확한 ACU 사용량을 알아보려면 DBShardGroup 지표의 `DBShardGroupComputeRedundancyCapacity` 지표를 참조하세요.

d. DB 샤드 그룹에 공개적으로 액세스할 수 있도록 할지 선택합니다.

> **Note**
>
> DB 샤드 그룹을 만든 후에는 이 설정을 수정할 수 없습니다.

6. 연결의 경우:

   a. (선택 사항) EC2 컴퓨팅 리소스에 연결을 선택한 다음 기존 EC2 인스턴스를 선택하거나 새 인스턴스를 만듭니다.

   > **Note**
   >
   > EC2 인스턴스에 연결하는 경우 DB 샤드 그룹에 공개적으로 액세스할 수 있도록 설정할 수 없습니다.

   b. 네트워크 유형에서 IPv4 또는 이중 스택 모드를 선택합니다.

   c. 가상 프라이빗 클라우드(VPC) 및 DB 서브넷 그룹을 선택하거나 기본 설정을 사용합니다.

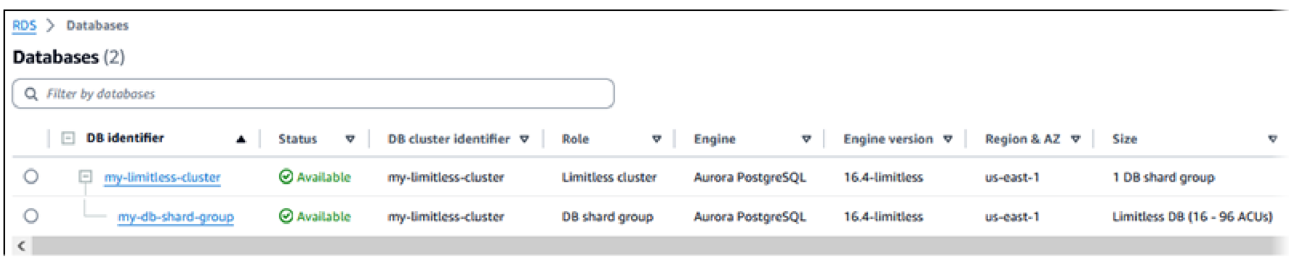
> **Note**
>
> 미국 동부(버지니아 북부) 리전에서 Limitless Database DB 클러스터를 만드는 경우 DB 서브넷 그룹에 `us-east-1e` 가용 영역(AZ)을 포함하지 마세요. 리소스 제한으로 인해 `us-east-1e` AZ에서는 Aurora Serverless v2가 지원되지 않고, 따라서 Limitless Database도 지원되지 않습니다.

   d. VPC 보안 그룹(방화벽)을 선택하거나 기본 설정을 사용합니다.

7. 데이터베이스 인증에서 암호 인증 또는 암호 및 IAM 데이터베이스 인증을 선택합니다.
8. 모니터링에서 성능 개선 도우미 켜기 및 향상된 모니터링 활성화 확인란이 선택되어 있는지 확인합니다.

   성능 개선 도우미에서 보존 기간을 최소 1개월로 선택합니다.
9. 페이지에서 마지막 추가 구성을 확장합니다.
10. 로그 내보내기의 경우 PostgreSQL 로그 확인란이 선택되어 있는지 확인합니다.
11. 필요에 따라 다른 설정을 지정합니다. 자세한 내용은 [Aurora DB 클러스터 설정](#) 섹션을 참조하세요.
12. 데이터베이스 생성를 선택합니다.

기본 DB 클러스터 및 DB 샤드 그룹이 만들어지면 데이터베이스 페이지에 표시됩니다.



CLI

AWS CLI를 사용하여 Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터를 만들 때 다음 작업을 수행합니다.

1. 기본 DB 클러스터 만들기.
2. DB 샤드 그룹 만들기.

기본 DB 클러스터 만들기

DB 클러스터를 만들려면 다음 파라미터가 필요합니다.

- `--db-cluster-identifier` - DB 클러스터의 이름입니다.
- `--engine` – DB 클러스터는 `aurora-postgresql` DB 엔진을 사용해야 합니다.
- `--engine-version` – DB 클러스터는 다음과 같은 DB 엔진 버전 중 하나를 사용해야 합니다.
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – DB 클러스터는 `aurora-iopt1` DB 클러스터 스토리지 구성을 사용해야 합니다.
- `--cluster-scalability-type` - Aurora DB 클러스터의 확장성 모드를 지정합니다. `limitless`로 설정하면 클러스터가 Aurora PostgreSQL Limitless Database로 작동합니다. `standard`(기본값)로 설정하면 클러스터가 일반 DB 인스턴스 만들기를 사용합니다.

> **Note**
>
> DB 클러스터를 만든 후에는 이 설정을 수정할 수 없습니다.

- `--master-username` - DB 클러스터의 마스터 사용자 이름입니다.
- `--master-user-password` - 마스터 사용자의 암호입니다.
- `--enable-performance-insights` – 성능 개선 도우미를 활성화해야 합니다.
- `--performance-insights-retention-period` – 성능 개선 도우미 보존 기간은 최소 31일이어야 합니다.
- `--monitoring-interval` - DB 클러스터에 대한 확장 모니터링 지표를 수집하는 시점 사이의 간격(초)입니다. 이 값은 `0`일 수 없습니다.
- `--monitoring-role-arn` - RDS에서 Amazon CloudWatch Logs로 확장 모니터링 지표를 전송할 수 있도록 하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- `--enable-cloudwatch-logs-exports` - `postgresql` 로그를 CloudWatch Logs로 내보내야 합니다.

다음 파라미터는 선택적입니다.

- `--db-subnet-group-name` - DB 클러스터와 연결할 DB 서브넷 그룹입니다. 이것은 DB 클러스터와 연결된 VPC도 결정합니다.

> **Note**
>
> 미국 동부(버지니아 북부) 리전에서 Limitless Database DB 클러스터를 만드는 경우 DB 서브넷 그룹에 `us-east-1e` 가용 영역(AZ)을 포함하지 마세요. 리소스 제한으로 인해 `us-east-1e` AZ에서는 Aurora Serverless v2가 지원되지 않고, 따라서 Limitless Database도 지원되지 않습니다.

- `--vpc-security-group-ids` - DB 클러스터와 연결할 VPC 보안 그룹의 목록입니다.
- `--performance-insights-kms-key-id` - 성능 개선 도우미 데이터의 암호화를 위한 AWS KMS key 식별자입니다. KMS 키를 지정하지 않으면 AWS 계정의 기본 키가 사용됩니다.
- `--region` - DB 클러스터를 만드는 AWS 리전입니다. Aurora PostgreSQL Limitless Database를 지원해야 합니다.

기본 VPC 및 VPC 보안 그룹을 사용하려면 `--db-subnet-group-name` 및 `--vpc-security-group-ids` 옵션을 생략합니다.

기본 DB 클러스터를 만드는 방법

- ```
  aws rds create-db-cluster \
      --db-cluster-identifier my-limitless-cluster \
      --engine aurora-postgresql \
      --engine-version 16.6-limitless \
      --storage-type aurora-iopt1 \
      --cluster-scalability-type limitless \
      --master-username myuser \
      --master-user-password mypassword \
      --db-subnet-group-name mysubnetgroup \
      --vpc-security-group-ids sg-c7e5b0d2 \
      --enable-performance-insights \
      --performance-insights-retention-period 31 \
      --monitoring-interval 5 \
  ```
  ```

```
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
    --enable-cloudwatch-logs-exports postgresql
```

자세한 내용은 create-db-cluster를 참조하세요.

DB 샤드 그룹 만들기

다음으로, 방금 만든 DB 클러스터에서 DB 샤드 그룹을 만듭니다. 다음 파라미터는 필수 파라미터입니다.

- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.

  DB 샤드 그룹 식별자에는 다음과 같은 제약이 있습니다.

  - DB 샤드 그룹을 만드는 AWS 계정과 AWS 리전에서 고유해야 합니다.
  - 1~63자의 문자, 숫자 또는 하이픈을 포함해야 합니다.
  - 첫 번째 자리는 문자여야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
  - **Important**

    DB 샤드 그룹을 만든 후에는 DB 클러스터 식별자 또는 DB 샤드 그룹 식별자를 변경할 수 없습니다.
- `--db-cluster-identifier` - DB 샤드 그룹을 만드는 DB 클러스터의 이름입니다.
- `--max-acu` - DB 샤드 그룹의 최대 용량입니다. ACU는 16~6,144개여야 합니다. 6,144ACU보다 높은 용량 한도는 AWS에 문의하세요.

  초기 라우터 및 샤드 수는 DB 샤드 그룹을 만들 때 설정한 최대 용량에 따라 결정됩니다. 최대 용량이 높을수록 DB 샤드 그룹에서 만들어지는 라우터 및 샤드가 많아집니다. 자세한 내용은 DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결 섹션을 참조하세요.

다음 파라미터는 선택적입니다.

- `--compute-redundancy` - DB 샤드 그룹에 대한 대기를 만들지를 나타냅니다. 이 파라미터의 값은 다음과 같을 수 있습니다.
  - `0` - 각 샤드에 대해 대기 없이 DB 샤드 그룹을 만듭니다. 이것이 기본값입니다.
  - `1` - 다른 가용 영역(AZ)에 하나의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.
  - `2` - 서로 다른 2개의 AZ에 2개의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.

  Note

  컴퓨팅 중복을 0이 아닌 값으로 설정하면 총 샤드 수가 두 배 또는 세 배가 됩니다. 이로 인해 추가 비용이 발생합니다.

컴퓨팅 대기의 노드는 라이터와 동일한 용량으로 스케일 업 및 다운됩니다. 대기에 대해 용량 범위를 별도로 설정하지 않습니다.

- `--min-acu` – DB 샤드 그룹의 최소 용량입니다. ACU는 기본값인 16개 이상이어야 합니다.
- `--publicly-accessible|--no-publicly-accessible` - 공개적으로 액세스할 수 있는 IP 주소를 DB 샤드 그룹에 할당할지를 나타냅니다. DB 샤드 그룹에 대한 액세스는 클러스터에서 사용하는 보안 그룹에 의해 제어됩니다.

  기본값은 `--no-publicly-accessible`입니다.

  > **Note**
  >
  > DB 샤드 그룹을 만든 후에는 이 설정을 수정할 수 없습니다.

DB 샤드 그룹을 만드는 방법

- ```
  aws rds create-db-shard-group \
      --db-shard-group-identifier my-db-shard-group \
      --db-cluster-identifier my-limitless-cluster \
      --max-acu 1000
  ```
  ```

DB 샤드 그룹 사용

다음 작업을 수행하여 Aurora PostgreSQL Limitless Database에 대한 DB 샤드 그룹을 추가하고 관리합니다.

주제

Aurora PostgreSQL Limitless Database DB 클러스터에 연결

Aurora PostgreSQL Limitless Database를 사용하려면 클러스터 라이터 또는 리더 엔드포인트에 연결합니다. PostgreSQL에서 작동하는 `psql` 또는 다른 연결 유틸리티를 사용할 수 있습니다.

```
$ psql -h DB_cluster_endpoint -p port_number -U database_username -d postgres_limitless
```

다음 예시에서는 CLI에서 만든 DB 클러스터의 엔드포인트를 사용합니다.

```
$ psql -h my-limitless-cluster.cluster-ckifpdyyyxxx.us-east-1.rds.amazonaws.com -p 5432
 -U postgres -d postgres_limitless
```

> **Note**
>
> Aurora PostgreSQL Limitless Database에서 DB 샤드 그룹의 기본 데이터베이스는 `postgres_limitless`입니다.

무한 연결 플러그인 사용

Aurora PostgreSQL Limitless Database에 연결할 때 클라이언트는 클러스터 엔드포인트를 사용하여 연결하고 Amazon Route 53에 의해 트랜잭션 라우터로 라우팅됩니다. 그러나 Route 53는 로드 밸런싱 기능이 제한되어 있으며 트랜잭션 라우터의 워크로드를 불균형하게 만들 수 있습니다. AWS JDBC 드라이버용 무한 연결 플러그인은 로드 인식으로 클라이언트 측 로드 밸런싱을 수행하여 이 문제를 해결합니다. AWS JDBC 드라이버에 대한 자세한 내용은 Amazon Web Services(AWS) JDBC 드라이버를 사용하여 Aurora PostgreSQL에 연결 섹션을 참조하세요.

DB 샤드 그룹에서 라우터 및 샤드 수 찾기

다음 쿼리를 사용하여 라우터 및 샤드 수를 찾을 수 있습니다.

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
 6             | shard
```

DB 샤드 그룹 설명

`describe-db-shard-groups` AWS CLI 명령을 사용하여 DB 샤드 그룹을 설명합니다. 다음 파라미터는 선택 사항입니다.

- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.

다음 예시에서는 특정 DB 샤드 그룹을 설명합니다.

```
aws rds describe-db-shard-groups --db-shard-group-identifier my-db-shard-group
```

출력은 다음 예시와 유사합니다.

```
{
    "DBShardGroups": [
        {
            "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
            "DBShardGroupIdentifier": "my-shard-group",
            "DBClusterIdentifier": "my-limitless-cluster",
            "MaxACU": 1000.0,
            "ComputeRedundancy": 0,
            "Status": "available",
            "PubliclyAccessible": false,
            "Endpoint": "my-limitless-cluster.limitless-ccetp2abcdef.us-
east-1.rds.amazonaws.com"
        }
```

```
    ]
}
```

DB 샤드 그룹 재부팅

최대 용량 변경으로 인해 `max_connections` 파라미터가 변경되는 경우와 같이 DB 샤드 그룹을 재부팅해야 하는 경우가 있습니다.

AWS Management Console 또는 AWS CLI를 사용하여 DB 샤드 그룹의 용량을 변경할 수 있습니다.

콘솔

다음 절차를 따르세요.

AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

1. 데이터베이스 페이지로 이동합니다.
2. 재부팅할 DB 샤드 그룹을 선택합니다.
3. 작업에서 재부팅을 선택합니다.
4. 확인을 선택합니다.

CLI

DB 파라미터 그룹을 재부팅하려면 `reboot-db-shard-group` AWS CLI 명령을 다음 파라미터와 함께 사용합니다.

- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.

다음 예시에서는 DB 샤드 그룹을 재부팅합니다.

```
aws rds reboot-db-shard-group --db-shard-group-identifier my-db-shard-group
```

DB 샤드 그룹의 용량 변경

AWS Management Console 또는 AWS CLI를 사용하여 DB 샤드 그룹의 용량을 변경할 수 있습니다.

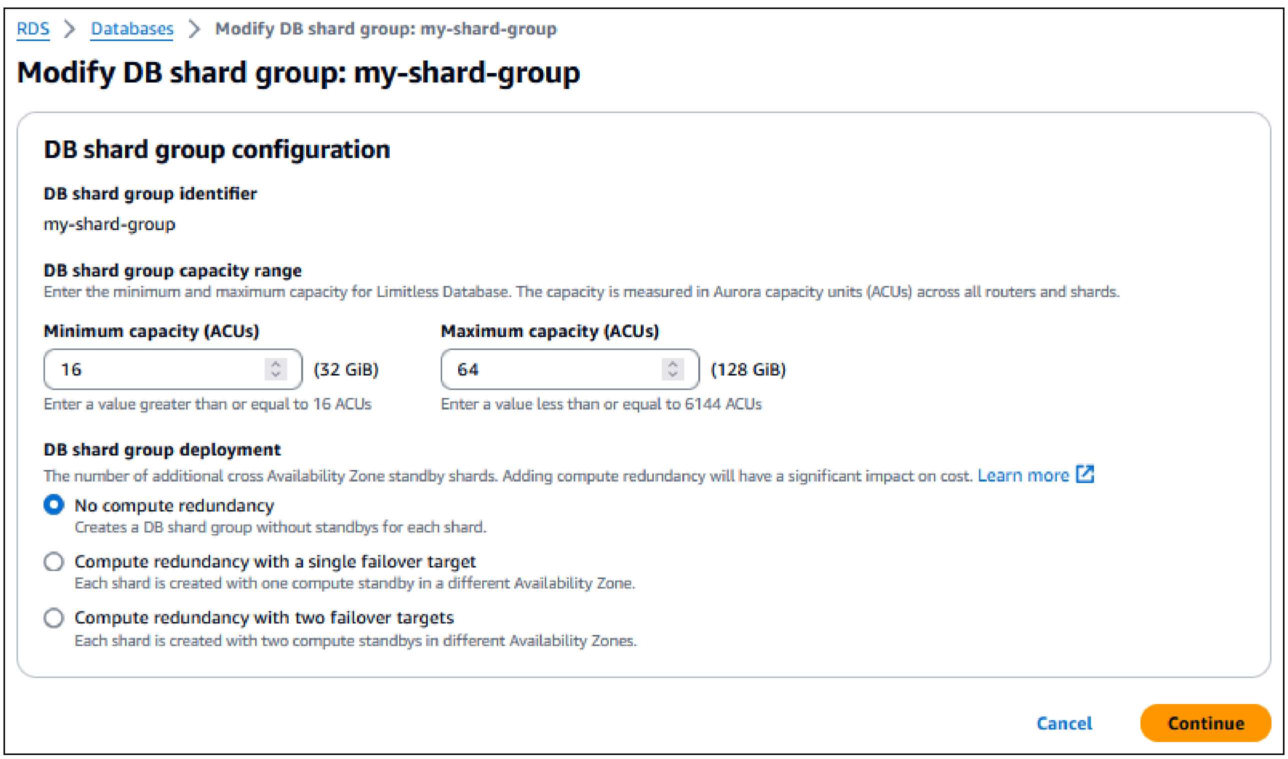
콘솔

다음 절차를 따르세요.

https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.

1. 데이터베이스 페이지로 이동합니다.
2. 수정할 DB 샤드 그룹을 선택합니다.
3. 작업에서 수정을 선택합니다.

   DB 샤드 그룹 수정 페이지가 표시됩니다.



4. 새 최소 용량(ACU) 값을 입력합니다(예: **100**).
5. 새 최대 용량(ACU) 값을 입력합니다(예: **1000**).
6. 계속을 선택합니다.

   변경 사항 요약과 함께 확인 페이지가 표시됩니다.

7. 변경 사항을 검토한 다음 DB 샤드 그룹 수정을 선택합니다.

CLI

`modify-db-shard-group` AWS CLI 명령을 다음 파라미터와 함께 사용합니다.

- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.
- `--max-acu` - DB 샤드 그룹의 새로운 최대 용량입니다. DB 샤드 그룹의 최대 용량을 16~6,144ACU로 설정할 수 있습니다. 6,144ACU보다 높은 용량 한도는 AWS에 문의하세요.

  라우터 및 샤드 수는 변경되지 않습니다.
- `--min-acu` - DB 샤드 그룹의 새로운 최소 용량입니다. ACU는 기본값인 16개 이상이어야 합니다.

다음 CLI 예시에서는 DB 샤드 그룹의 용량 범위를 100~1,000ACU로 변경합니다.

```
aws rds modify-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --min-acu 100 \
    --max-acu 1000
```

DB 샤드 그룹에서 샤드 분할

DB 샤드 그룹의 샤드를 두 개의 더 작은 샤드로 수동으로 분할할 수 있습니다. 이를 사용자 시작 샤드 분할이라고 합니다.

Aurora PostgreSQL Limitless Database는 데이터의 양이 매우 많거나 사용량이 매우 많을 때도 샤드를 분할할 수 있습니다. 이를 시스템 시작 샤드 분할이라고 합니다.

주제

사전 조건

사용자 시작 샤드 분할에는 다음과 같은 사전 조건이 있습니다.

- DB 샤드 그룹이 있어야 합니다.
- DB 샤드 그룹은 비워둘 수 없습니다. 샤딩된 테이블을 하나 이상 포함해야 합니다.
- 사용자에게 `rds_aurora_limitless_cluster_admin` 권한이 있어야 합니다. `rds_superuser`에는 이 권한이 있으므로 마스터 사용자에게도 권한이 있습니다. `rds_superuser`는 다른 사용자에게 권한을 부여할 수 있습니다.

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

- 분할하려는 샤드의 하위 클러스터(노드) ID를 알아야 합니다. 다음 쿼리를 사용하여 ID를 찾을 수 있습니다.

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
```

```
3                | shard
4                | shard
5                | shard
6                | shard
```

시스템 시작 샤드 분할을 활성화하려면 DB 클러스터와 연결된 사용자 지정 DB 클러스터 파라미터 그룹에서 다음 DB 클러스터 파라미터를 설정합니다.

파라미터	값
`rds_aurora.limitless_enable_auto_scale`	`on`
`rds_aurora.limitless_auto_scale_options`	`split_shard` 또는 `add_router,split_shard` 중 하나
`rds_aurora.limitless_finalize_split_shard_mode`	이 파라미터는 시스템 시작 샤드 분할을 마무리하는 방법을 결정합니다. 값은 다음 중 하나일 수 있습니다. • `user_initiated` - 샤드 분할을 마무리할 시기를 사용자가 결정합니다. 이것이 기본값입니다. • `immediate` – 샤드 분할이 즉시 마무리됩니다. 자세한 내용은 샤드 분할 마무리 섹션을 참조하세요. **Note** 이 파라미터는 시스템 시작 샤드 분할에만 적용됩니다.

자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 섹션을 참조하세요.

샤드 분할

DB 샤드 그룹에서 샤드를 분할하려면 rds_aurora.limitless_split_shard 함수를 사용합니다. 이 함수는 비동기식으로 실행되는 샤드 분할 작업을 시작합니다.

```
SELECT rds_aurora.limitless_split_shard('subcluster_id');
```

작업을 성공적으로 제출하면 작업 ID가 반환될 때까지 기다립니다. 예를 들면 다음과 같습니다.

```
SELECT rds_aurora.limitless_split_shard('3');

    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
> 동시 샤드 분할 작업은 지원되지 않습니다. 각 작업을 순차적으로 실행하고 다른 추가 작업을 시작하기 전에 각 작업을 완료합니다.

샤드 분할 추적

작업 ID를 사용하여 샤드 분할 작업을 추적할 수 있습니다. 특정 작업을 설명하고 이에 대한 세부 정보를 얻으려면 다음 쿼리를 실행합니다.

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(job_id);
```

예시:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(1691300000000);

    job_id     |    action   |      job_details       | status  |    submission_time
 |                   message
---------------+-------------+------------------------+---------
+------------------------+-----------------------------------------
 1691300000000 | SPLIT_SHARD | Split Shard 3 by User | SUCCESS | 2023-08-06 05:33:20+00
 | Scaling job succeeded.                 +
```

```
                |             |                        |          |
 | New shard instance with ID 7 was created.
(1 row)
```

존재하지 않는 작업을 입력으로 전달하면 쿼리가 오류를 반환합니다.

```
SELECT * from rds_aurora.limitless_list_shard_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

작업 ID 없이 동일한 쿼리를 사용하여 모든 샤드 분할 작업의 상태를 추적할 수 있습니다. 예를 들면 다음과 같습니다.

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs();

    job_id     |   action    | job_details           |   status    |    submission_time
     |                  message
---------------+-------------+-----------------------+-------------
+------------------------
+---------------------------------------------------------------
 1691200000000 | SPLIT_SHARD | Split Shard 3 by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | SPLIT_SHARD | Split Shard 4 by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded. +
                |             |                        |             |
     | New shard instance with ID 7 was created.
 1691400000000 | SPLIT_SHARD | Split Shard 5 by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add shard job 1691400000000.
                |             |                        |             |
     | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | SPLIT_SHARD | Split Shard 5 by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

작업 상태는 다음 값 중 하나일 수 있습니다.

- IN_PROGRESS - 샤드 분할 작업이 제출되어 진행 중입니다. 한 번에 하나의 작업만 진행할 수 있습니다.
- PENDING - 샤드 분할 작업을 사용자가 마무리하기를 기다리고 있습니다. 자세한 내용은 [샤드 분할 마무리](#) 섹션을 참조하세요.
- CANCELLATION_IN_PROGRESS - 사용자가 샤드 분할 작업을 취소하는 중입니다.

- CANCELED - 사용자 또는 시스템이 샤드 분할 작업을 성공적으로 취소했습니다.
- SUCCESS - 샤드 분할 작업이 성공적으로 완료되었습니다. message 필드에는 새로운 샤드의 인스턴스 ID가 포함됩니다.
- FAILED - 샤드 분할 작업이 실패했습니다. message 필드에는 실패에 대한 세부 정보와 실패한 작업에 대한 후속 조치로 수행할 수 있는 작업이 포함되어 있습니다.

샤드 분할 마무리

마무리는 샤드 분할 프로세스의 마지막 단계입니다. 이로 인해 어느 정도 가동 중지 시간이 발생합니다. 샤드 분할 작업을 시작하면 작업이 성공적으로 완료된 직후에 마무리가 이루어집니다.

rds_aurora.limitless_enable_auto_scale 파라미터를 사용하여 시스템 시작 샤드 분할을 활성화한 경우 시스템에서 워크로드를 기반으로 샤드를 분할하는 경우가 있습니다.

이 경우 마무리가 즉시 수행될지 아니면 사용자가 선택한 시간에 수행될지를 사용자가 선택할 수 있습니다. rds_aurora.limitless_finalize_split_shard_mode DB 클러스터 파라미터를 사용하여 마무리 시기를 선택합니다.

- 값을 immediate로 설정하면 즉시 마무리됩니다.
- 값을 user_initiated로 설정하면 사용자가 샤드 분할 작업을 수동으로 완료해야 합니다.

  RDS 이벤트가 사용자에게 전송되고 샤드 분할 작업의 상태가 PENDING으로 설정됩니다.

user_initiated로 설정하면 rds_aurora.limitless_finalize_split_shard 함수를 사용하여 샤드 분할 작업을 마무리합니다.

```
SELECT * FROM rds_aurora.limitless_finalize_split_shard(job_id);
```

> **Note**
>
> 이 함수는 사용자가 아닌 시스템에서 시작한 샤드 분할에만 적용됩니다.

샤드 분할 취소

IN_PROGRESS 또는 PENDING인 사용자 시작 또는 시스템 시작 샤드 분할을 취소할 수 있습니다. 취소하려면 작업 ID가 필요합니다.

```
SELECT * from rds_aurora.limitless_cancel_shard_scale_jobs(job_id);
```

오류가 없으면 출력이 반환되지 않습니다. 작업 추적 쿼리를 사용하여 취소를 추적할 수 있습니다.

DB 샤드 그룹에 라우터 추가

DB 샤드 그룹에 라우터를 추가할 수 있습니다.

주제

사전 조건

라우터를 추가하는 데는 다음과 같은 사전 조건이 있습니다.

- DB 샤드 그룹이 있어야 합니다.
- 사용자에게 `rds_aurora_limitless_cluster_admin` 권한이 있어야 합니다. `rds_superuser`에는 이 권한이 있으므로 마스터 사용자에게도 권한이 있습니다. `rds_superuser`는 다른 사용자에게 권한을 부여할 수 있습니다.

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

> **Note**
>
> DB 샤드 그룹이 만들어진 후 AWS 계정의 기본 CA 인증서를 변경하면 새 라우터는 기존 라우터의 CA 인증서와 다른 새 CA 인증서를 사용합니다. 트러스트 스토어에 따라 일부 연결이 실패할 수 있습니다.

- 시스템 시작 라우터 추가를 활성화하려면 DB 클러스터와 연결된 사용자 지정 DB 클러스터 파라미터 그룹에서 다음 DB 클러스터 파라미터를 설정합니다.

파라미터	값
`rds_aurora.limitless_enable_auto_scale`	`on`

파라미터	값
rds_aurora.limitless_auto_scale_options	add_router 또는 add_router,split_shard

자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 섹션을 참조하세요.

라우터 추가

라우터를 추가하려면 rds_aurora.limitless_add_router 함수를 사용합니다. 이 함수는 비동기식으로 실행되는 라우터 추가 작업을 시작합니다.

```
SELECT rds_aurora.limitless_add_router();
```

작업을 성공적으로 제출하면 작업 ID가 반환될 때까지 기다립니다. 예를 들면 다음과 같습니다.

```
    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
> 동시 라우터 추가 작업은 지원되지 않습니다. 작업을 순차적으로 실행하고 다른 추가 작업을 시작하기 전에 각 작업을 완료합니다.

라우터 추가 추적

작업 ID를 사용하여 라우터 추가 작업을 추적할 수 있습니다. 특정 작업을 설명하고 이에 대한 세부 정보를 얻으려면 다음 쿼리를 실행합니다.

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(job_id);
```

예시:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(1691300000000);
```

```
    job_id      |   action   |        job_details        | status  |    submission_time
   |                   message
---------------+------------+---------------------------+---------
+------------------------+-----------------------------------------
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                   +
               |            |                           |         |
   | New router instance with ID 7 was created.
(1 row)
```

존재하지 않는 작업을 입력으로 전달하면 쿼리가 오류를 반환합니다.

```
SELECT * from rds_aurora.limitless_list_router_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

작업 ID 없이 동일한 쿼리를 사용하여 모든 라우터 추가 작업의 상태를 추적할 수 있습니다. 예를 들면 다음과 같습니다.

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs();

    job_id      |   action   |        job_details        |   status    |
 submission_time      |                  message
---------------+------------+---------------------------+-------------
+------------------------+-----------------------------------------
 1691200000000 | ADD_ROUTER | Add 1 new Router by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                 +
               |            |                           |             |
       | New router instance with ID 7 was created.
 1691400000000 | ADD_ROUTER | Add 1 new Router by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add router job 1691400000000.
               |            |                           |             |
       | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | ADD_ROUTER | Add 1 new Router by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

작업 상태는 다음 값 중 하나일 수 있습니다.

- IN_PROGRESS - 라우터 추가 작업이 제출되어 진행 중입니다. 한 번에 하나의 작업만 진행할 수 있습니다.
- CANCELLATION_IN_PROGRESS - 사용자가 라우터 추가 작업을 취소하는 중입니다.
- CANCELED - 사용자 또는 시스템이 라우터 추가 작업을 성공적으로 취소했습니다.
- SUCCESS - 라우터 추가 작업이 성공적으로 완료되었습니다. message 필드에는 새로운 라우터의 인스턴스 ID가 포함됩니다.
- FAILED - 라우터 추가 작업이 실패했습니다. message 필드에는 실패에 대한 세부 정보와 실패한 작업에 대한 후속 조치로 수행할 수 있는 작업이 포함되어 있습니다.

> **Note**
>
> 라우터 추가를 마무리할 필요가 없으므로 PENDING 상태가 없습니다. 가동 중지 시간이 발생하지 않습니다.

라우터 추가 취소

IN_PROGRESS인 라우터 추가를 취소할 수 있습니다. 취소하려면 작업 ID가 필요합니다.

```
SELECT * from rds_aurora.limitless_cancel_router_scale_jobs(job_id);
```

오류가 없으면 출력이 반환되지 않습니다. 작업 추적 쿼리를 사용하여 취소를 추적할 수 있습니다.

DB 샤드 그룹 삭제

필요한 경우 DB 샤드 그룹을 삭제할 수 있습니다. DB 샤드 그룹을 삭제하면 컴퓨팅 노드(샤드 및 라우터)가 삭제되지만 스토리지는 삭제되지 않습니다.

콘솔

https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.

1. 데이터베이스 페이지로 이동합니다.
2. 삭제할 DB 샤드 그룹을 선택합니다.
3. 작업에 대해 삭제를 선택합니다.
4. 입력란에 **`delete me`**를 입력한 다음, 삭제를 선택합니다.

DB 샤드 그룹이 삭제됩니다.

AWS CLI

DB 파라미터 그룹을 삭제하려면 `delete-db-shard-group` AWS CLI 명령을 다음 파라미터와 함께 사용합니다.

- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.

다음 예시에서는 앞서 만든 Aurora PostgreSQL DB 클러스터에서 DB 샤드 그룹을 삭제합니다.

```
aws rds delete-db-shard-group --db-shard-group-identifier my-db-shard-group
```

기존 Aurora PostgreSQL Limitless Database DB 클러스터에 DB 샤드 그룹 추가

예를 들어 DB 클러스터를 복원 중이거나 DB 샤드 그룹을 삭제한 경우 기존 DB 클러스터에서 DB 샤드 그룹을 만들 수 있습니다.

기본 DB 클러스터 및 DB 샤드 그룹 요구 사항에 대한 자세한 내용은 Aurora PostgreSQL Limitless Database 요구 사항 및 고려 사항 섹션을 참조하세요.

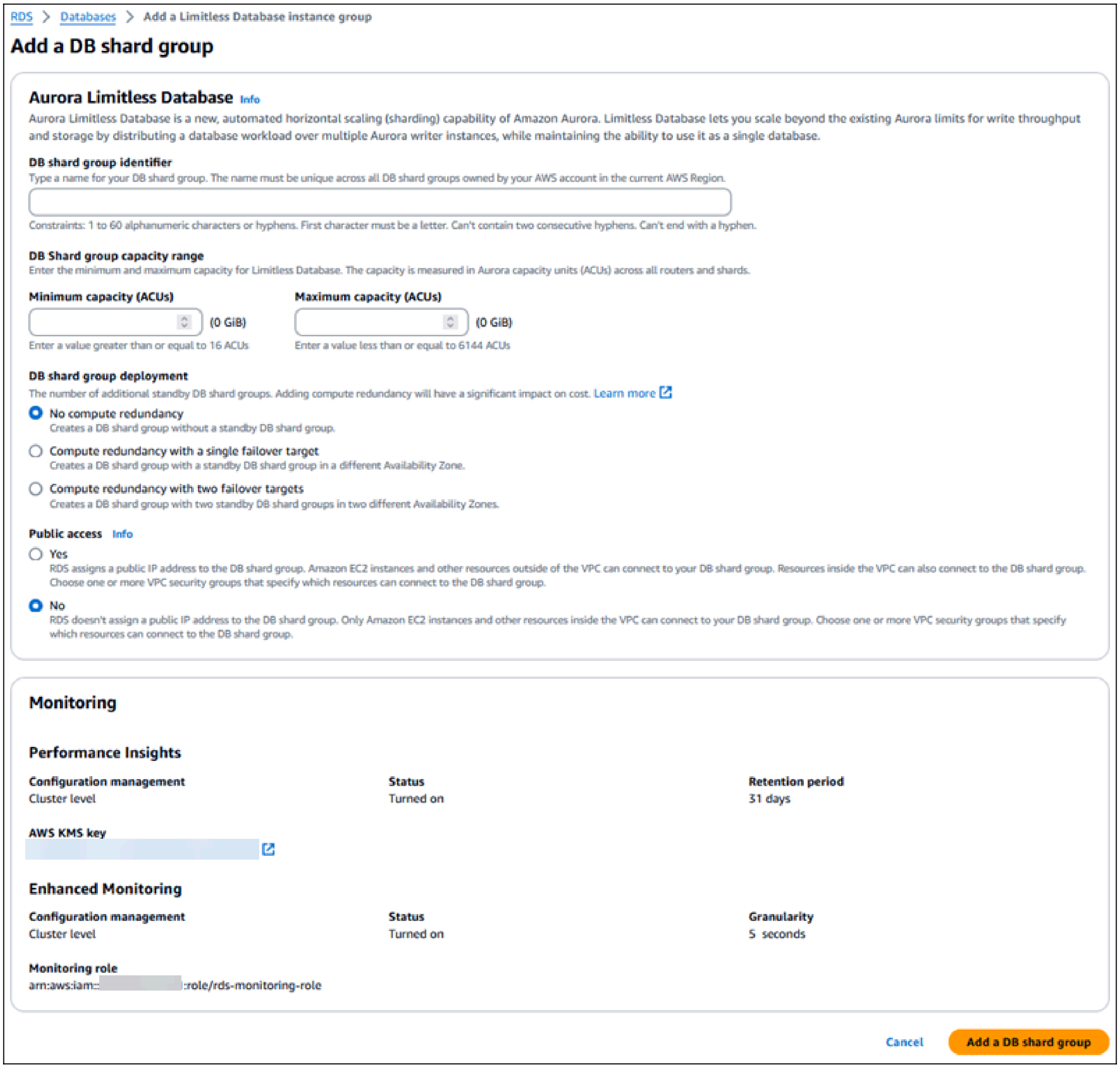
> **Note**
>
> 클러스터당 하나의 샤드 그룹만 가질 수 있습니다.
> DB 샤드 그룹을 만들려면 Limitless Database DB 클러스터가 `available` 상태여야 합니다.

콘솔

AWS Management Console을 사용하여 기존 DB 클러스터에 DB 샤드 그룹을 추가할 수 있습니다.

DB 샤드 그룹을 추가하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스 페이지로 이동합니다.
3. DB 샤드 그룹을 추가할 Limitless Database DB 클러스터를 선택합니다.
4. 작업에서 DB 샤드 그룹 추가를 선택합니다.



5. DB 샤드 그룹 식별자를 입력합니다.

> ⚠ Important
>
> DB 샤드 그룹을 만든 후에는 DB 클러스터 식별자 또는 DB 샤드 그룹 식별자를 변경할 수 없습니다.

6. 최소 용량(ACU)을 입력합니다. 값을 최소 16ACU로 지정합니다.
7. 최대 용량(ACU)을 입력합니다. 16~6,144ACU의 값을 사용합니다.

   자세한 내용은 DB 샤드 그룹 최대 용량과 만들어진 라우터 및 샤드 수 연결 단원을 참조하십시오.

8. DB 샤드 그룹 배포의 경우 DB 샤드 그룹에 대한 대기를 만들지 선택합니다.

   - 컴퓨팅 중복 없음 - 각 샤드에 대해 대기 없이 DB 샤드 그룹을 만듭니다. 이것이 기본값입니다.
   - 단일 장애 조치 타겟을 사용한 컴퓨팅 중복 - 다른 가용 영역(AZ)에 하나의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.
   - 2개의 장애 조치 타겟을 사용한 컴퓨팅 중복 - 2개의 서로 다른 AZ에서 2개의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.

9. DB 샤드 그룹에 공개적으로 액세스할 수 있도록 할지 선택합니다.

   > **Note**
   >
   > DB 샤드 그룹을 만든 후에는 이 설정을 수정할 수 없습니다.

10. DB 샤드 그룹 추가를 선택합니다.

AWS CLI

`create-db-shard-group` AWS CLI 명령을 사용하여 DB 샤드 그룹을 만듭니다.

다음 파라미터는 필수 파라미터입니다.

- `--db-cluster-identifier` - DB 샤드 그룹이 속한 DB 클러스터입니다.
- `--db-shard-group-identifier` - DB 샤드 그룹의 이름입니다.

  DB 샤드 그룹 식별자에는 다음과 같은 제약이 있습니다.

  - DB 샤드 그룹을 만드는 AWS 계정과 AWS 리전에서 고유해야 합니다.
  - 1~63자의 문자, 숫자 또는 하이픈을 포함해야 합니다.
  - 첫 번째 자리는 문자여야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.

> **Important**
>
> DB 샤드 그룹을 만든 후에는 DB 클러스터 식별자 또는 DB 샤드 그룹 식별자를 변경할 수 없습니다.

- `--max-acu` – DB 샤드 그룹의 최대 용량입니다. 16~6,144ACU의 값을 사용합니다.

다음 파라미터는 선택적입니다.

- `--compute-redundancy` - DB 샤드 그룹에 대한 대기를 만들지를 나타냅니다. 이 파라미터의 값은 다음과 같을 수 있습니다.
  - `0` - 각 샤드에 대해 대기 없이 DB 샤드 그룹을 만듭니다. 이것이 기본값입니다.
  - `1` - 다른 가용 영역(AZ)에 하나의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.
  - `2` - 서로 다른 2개의 AZ에 2개의 컴퓨팅 대기가 있는 DB 샤드 그룹을 만듭니다.

> **Note**
>
> 컴퓨팅 중복을 0이 아닌 값으로 설정하면 총 노드 수가 두 배 또는 세 배가 됩니다. 이로 인해 추가 비용이 발생합니다.

- `--min-acu` – DB 샤드 그룹의 최소 용량입니다. ACU는 기본값인 16개 이상이어야 합니다.

- `--publicly-accessible|--no-publicly-accessible` - 공개적으로 액세스할 수 있는 IP 주소를 DB 샤드 그룹에 할당할지를 나타냅니다. DB 샤드 그룹에 대한 액세스는 클러스터에서 사용하는 보안 그룹에 의해 제어됩니다.

  기본값은 `--no-publicly-accessible`입니다.

> **Note**
>
> DB 샤드 그룹을 만든 후에는 이 설정을 수정할 수 없습니다.

다음 예시에서는 Aurora PostgreSQL DB 클러스터에 DB 샤드 그룹을 만듭니다.

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-db-cluster \
    --db-shard-group-identifier my-new-shard-group \
    --max-acu 1000
```

출력은 다음 예시와 유사합니다.

```
{
    "Status": "CREATING",
    "Endpoint": "my-db-cluster.limitless-ckifpdyyyxxx.us-east-1.rds.amazonaws.com",
    "PubliclyAccessible": false,
    "DBClusterIdentifier": "my-db-cluster",
    "MaxACU": 1000.0,
    "DBShardGroupIdentifier": "my-new-shard-group",
    "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
    "ComputeRedundancy": 0
}
```

Aurora PostgreSQL Limitless Database 테이블 만들기

Aurora PostgreSQL Limitless Database에 데이터를 포함하는 세 가지 유형의 테이블이 있습니다.

- 표준 - Aurora PostgreSQL Limitless Database의 기본 테이블 유형입니다. CREATE TABLE 명령을 사용하여 표준 테이블을 만들고 해당 테이블에서 데이터 설명 언어(DDL) 및 데이터 조작 언어(DML) 작업을 실행할 수 있습니다.

  표준 테이블은 분산 테이블이 아닙니다. 시스템에서 내부적으로 선택한 샤드 중 하나에 저장됩니다.
- 샤딩됨 - 이러한 테이블은 여러 샤드에 분산됩니다. 데이터는 테이블의 지정된 열 값에 따라 샤드 간에 분할됩니다. 이 열 세트를 샤드 키라고 합니다.
- 참조 - 이러한 테이블은 모든 샤드에 복제됩니다. 제품 카탈로그 및 우편번호와 같이 자주 수정되지 않는 참조 데이터에 사용됩니다.

  참조 테이블과 샤딩된 테이블 간의 조인 쿼리는 샤드에서 실행할 수 있으므로 샤드와 라우터 간의 불필요한 데이터 이동이 없습니다.

무한 테이블을 만드는 방법에는 두 가지가 있습니다.

- 변수를 사용하여 무한 테이블 만들기 - 샤딩된 테이블과 참조 테이블을 새로 만들려는 경우 이 방법을 사용합니다.
- 표준 테이블을 무한 테이블로 변환 - 기존 표준 테이블을 샤딩된 테이블 및 참조 테이블로 변환하려는 경우 이 방법을 사용합니다.

또한 Aurora PostgreSQL Limitless Database에 대한 샘플 스키마도 제공합니다.

변수를 사용하여 무한 테이블 만들기

변수를 사용하여 테이블 만들기 모드를 설정하여 샤딩된 테이블과 참조 테이블을 만들 수 있습니다. 그러면 만든 테이블은 다른 모드를 설정할 때까지 이 모드를 사용합니다.

다음 변수를 사용하여 샤딩된 테이블과 참조 테이블을 만듭니다.

- `rds_aurora.limitless_create_table_mode` - 이 세션 변수를 `sharded` 또는 `reference`로 설정합니다. 이 변수의 기본값은 `standard`입니다.
- `rds_aurora.limitless_create_table_shard_key` - 이 세션 변수를 샤드 키로 사용할 열 이름 배열로 설정합니다. 이 변수는 `rds_aurora.limitless_create_table_mode`가 `sharded`가 아니면 무시됩니다.

  배열 열에 리터럴을 삽입할 때와 마찬가지로 값의 형식을 `untyped array literal`로 지정합니다. 자세한 내용은 PostgreSQL 설명서의 Arrays를 참조하세요.
- `rds_aurora.limitless_create_table_collocate_with` - 이 세션 변수를 특정 테이블 이름으로 설정하여 새로 만든 테이블을 해당 테이블과 공동 배치합니다.

  두 개 이상의 테이블이 동일한 샤드 키를 사용하여 샤딩되는 경우 해당 테이블을 명시적으로 정렬(공동 배치)할 수 있습니다. 두 개 이상의 테이블이 공동 배치되면 동일한 샤드 키 값을 가진 테이블의 행이 동일한 샤드에 배치됩니다. 공동 배치는 일부 작업을 단일 샤드로 제한하여 성능을 개선하는 데 도움이 됩니다.

> **Note**
>
> 모든 프라이머리 키 및 고유 키에는 샤드 키가 포함되어야 합니다. 즉, 샤드 키는 프라이머리 키 또는 고유 키의 하위 집합입니다.
> 무한 테이블에는 몇 가지 제한 사항이 있습니다. 자세한 내용은 Aurora PostgreSQL Limitless Database에 대한 DDL 제한 사항 및 기타 정보 단원을 참조하십시오.

주제

변수를 사용하여 무한 테이블을 만드는 예시

다음 예시에서는 이러한 변수를 사용하여 샤딩된 테이블과 참조 테이블을 만드는 방법을 보여줍니다.

id 샤드 키를 사용하여 items이라는 샤딩된 테이블을 만듭니다.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items(id int, val int, item text);
COMMIT;
```

item_id 및 item_cat 열로 구성된 샤드 키를 사용하여 items이라는 샤딩된 테이블을 만듭니다.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
CREATE TABLE items(item_id int, item_cat varchar, val int, item text);
COMMIT;
```

item_id 및 item_cat 열로 구성된 샤드 키를 사용하여 item_description이라는 샤딩된 테이블을 만들고 이전 예시의 items 테이블과 공동 배치합니다.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
SET LOCAL rds_aurora.limitless_create_table_collocate_with='items';
CREATE TABLE item_description(item_id int, item_cat varchar, color_id int);
COMMIT;
```

colors라는 참조 테이블을 만듭니다.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE colors(color_id int primary key, color varchar);
COMMIT;
```

rds_aurora.limitless_create_table_mode 세션 변수를 standard로 재설정하려면 다음 문을 사용합니다.

```
RESET rds_aurora.limitless_create_table_mode;
```

이 변수를 재설정하면 테이블이 기본값인 표준 테이블로 만들어집니다. 표준 테이블에 대한 자세한 내용은 표준 테이블을 무한 테이블로 변환 섹션을 참조하세요.

Aurora PostgreSQL Limitless Database 테이블 뷰

다음 뷰를 사용하여 Limitless Database 테이블에 대한 정보를 찾을 수 있습니다.

rds_aurora.limitless_tables

`rds_aurora.limitless_tables` 뷰에는 무한 테이블 및 해당 유형에 대한 정보가 포함되어 있습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | standard    | active       | standard    |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     |
 HASH (order_id)
         2 |     18579 | public      | customer    | active       | sharded     |
 HASH (cust_id)
(4 rows)
```

rds_aurora.limitless_table_collocations

`rds_aurora.limitless_table_collocations` 뷰에는 공동 배치된 샤딩된 테이블에 대한 정보가 포함되어 있습니다. 예를 들어, `orders` 및 `customers` 테이블은 공동 배치되며 `collocation_id`가 동일합니다. `users` 및 `followers` 테이블은 공동 배치되며 `collocation_id`가 동일합니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
          16002 | public      | orders
          16002 | public      | customers
          16005 | public      | users
          16005 | public      | followers
(4 rows)
```

rds_aurora.limitless_table_collocation_distributions

rds_aurora.limitless_table_collocation_distributions은 각 공동 배치의 키 분배를 보여줍니다.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
          16002 |             6 | -9223372036854775808 | -4611686018427387904
          16002 |             5 | -4611686018427387904 |                    0
          16002 |             4 |                    0 |  4611686018427387904
          16002 |             3 |  4611686018427387904 |  9223372036854775807
          16005 |             6 | -9223372036854775808 | -4611686018427387904
          16005 |             5 | -4611686018427387904 |                    0
          16005 |             4 |                    0 |  4611686018427387904
          16005 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

표준 테이블을 무한 테이블로 변환

표준 테이블을 샤딩된 테이블 또는 참조 테이블로 변환할 수 있습니다. 변환 중에 데이터가 표준 테이블에서 분산 테이블로 이동한 다음 소스 표준 테이블이 삭제됩니다. `INSERT INTO SELECT FROM` 명령으로 데이터가 이동합니다.

목차

샤딩된 테이블 만들기

표준 테이블에서 `rds_aurora.limitless_alter_table_type_sharded` 프로시저를 실행하여 샤딩된 테이블을 만듭니다. 이 프로시저에서는 표준 테이블과 열 목록을 가져온 다음 열 목록을 샤드 키로 사용하여 지정된 테이블을 배포합니다. 프로시저는 동기식으로 실행되고 테이블에서 `ACCESS EXCLUSIVE` 잠금을 획득합니다.

프로시저가 성공적으로 완료되면 소스 표준 테이블이 삭제되고 이름이 같은 샤딩된 테이블을 사용할 수 있게 됩니다.

`rds_aurora.limitless_alter_table_type_sharded` 프로시저는 다음과 같은 구문을 사용합니다.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('schema.table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn']);
```

프로시저에는 다음과 같은 파라미터가 필요합니다.

- `schema` - 샤딩할 테이블이 포함된 데이터베이스 스키마입니다. 스키마가 지정되지 않은 경우 프로시저는 `search_path`를 사용합니다.
- `table` - 샤딩할 테이블입니다.
- `shard_keyn` - 샤드 키로 사용할 테이블 열의 배열입니다.

  샤드 키 값은 문자열 리터럴이므로 대/소문자를 구분합니다. 샤드 키에 작은따옴표(')가 포함된 경우 다른 작은따옴표를 사용하여 이스케이프 처리합니다. 예를 들어 테이블 열의 이름이 `customer's id`인 경우 `customer''s id`를 샤드 키로 사용합니다. 백슬래시(\)와 큰따옴표(")는 이스케이프 처리할 필요가 없습니다.

> **Note**
>
> 모든 프라이머리 키 및 고유 키에는 샤드 키가 포함되어야 합니다. 즉, 샤드 키는 프라이머리 키 또는 고유 키의 하위 집합입니다.
> 샤딩된 테이블에서 CHECK 제약은 표현식을 지원하지 않습니다.
> 자세한 내용은 제약 조건 단원을 참조하십시오.

샤딩된 테이블을 만드는 방법

다음 예시에서는 customer_id 샤드 키를 사용하여 customer 샤딩된 테이블을 만드는 방법을 보여 줍니다.

1. 표준 테이블을 만듭니다.

```
CREATE TABLE customer (customer_id INT PRIMARY KEY NOT NULL, zipcode INT, email
 VARCHAR);
```

2. 표준 테이블을 샤딩된 테이블로 변환합니다.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.customer',
 ARRAY['customer_id']);

postgres=> \d

                    List of relations
 Schema |     Name      |        Type        |       Owner
--------+---------------+--------------------+--------------------
 public | customer      | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table      | postgres_limitless
 public | customer_fs2 | foreign table      | postgres_limitless
 public | customer_fs3 | foreign table      | postgres_limitless
 public | customer_fs4 | foreign table      | postgres_limitless
 public | customer_fs5 | foreign table      | postgres_limitless
(6 rows)
```

공동 배치된 테이블 만들기

두 개 이상의 테이블이 동일한 샤드 키를 사용하여 샤딩되는 경우 해당 테이블을 명시적으로 정렬(공동 배치)할 수 있습니다. 두 개 이상의 테이블이 공동 배치되면 동일한 샤드 키 값을 가진 테이블의 행

이 동일한 샤드에 배치됩니다. 공동 배치는 일부 작업을 단일 샤드로 제한하여 성능을 개선하는 데 도움이 됩니다.

다음 구문과 함께 `rds_aurora.limitless_alter_table_type_sharded` 프로시저를 사용합니다.

```
postgres=> CALL
 rds_aurora.limitless_alter_table_type_sharded('schema.collocated_table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn'], 'schema.sharded_table');
```

프로시저에는 다음과 같은 파라미터가 필요합니다.

- `schema` - 공동 배치할 테이블이 포함된 데이터베이스 스키마입니다. 스키마가 지정되지 않은 경우 프로시저는 `search_path`를 사용합니다.
- `collocated_table` - 공동 배치할 테이블입니다.
- `shard_keyn` - 샤드 키로 사용할 테이블 열의 배열입니다.

  동일한 열 이름 및 열 유형을 포함하여 원래 샤딩된 테이블과 동일한 샤드 키를 사용해야 합니다.
- `sharded_table` - `collocated_table`을 공동 배치할 때 사용하는 샤딩된 테이블입니다.

공동 배치된 테이블을 만드는 방법

1. 샤딩된 테이블 만들기의 프로시저에 따라 첫 번째 샤딩된 테이블을 만듭니다.
2. 공동 배치된 테이블의 표준 테이블을 만듭니다.

   ```
   CREATE TABLE mytable2 (customer_id INT PRIMARY KEY NOT NULL, column1 INT, column2
    VARCHAR);
   ```
3. 표준 테이블을 공동 배치된 테이블로 변환합니다.

   ```
   postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.mytable2',
   ARRAY['customer_id'], 'public.customer');

   postgres=> \d

                       List of relations
    Schema |     Name      |        Type        |        Owner
   --------+---------------+--------------------+--------------------
    public | customer      | partitioned table | postgres_limitless
    public | customer_fs1  | foreign table     | postgres_limitless
   ```

```
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | mytable2     | partitioned table | postgres_limitless
 public | mytable2_fs1 | foreign table     | postgres_limitless
 public | mytable2_fs2 | foreign table     | postgres_limitless
 public | mytable2_fs3 | foreign table     | postgres_limitless
 public | mytable2_fs4 | foreign table     | postgres_limitless
 public | mytable2_fs5 | foreign table     | postgres_limitless
(12 rows)
```

참조 테이블 만들기

표준 테이블에서 `rds_aurora.limitless_alter_table_type_reference` 프로시저를 실행하여 참조 테이블을 만듭니다. 이 프로시저는 지정된 테이블을 DB 샤드 그룹의 모든 샤드에 복제하고 테이블 유형을 참조로 변경합니다. 프로시저는 동기식으로 실행되고 테이블에서 ACCESS EXCLUSIVE 잠금을 획득합니다.

프로시저가 성공적으로 완료되면 소스 표준 테이블이 삭제되고 이름이 같은 참조 테이블을 사용할 수 있게 됩니다.

`rds_aurora.limitless_alter_table_type_reference` 프로시저는 다음과 같은 구문을 사용합니다.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_reference('schema.table');
```

저장 프로시저에는 다음 파라미터가 필요합니다.

- `schema` - 복제할 테이블이 포함된 데이터베이스 스키마입니다. 스키마가 지정되지 않은 경우 프로시저는 `search_path`를 사용합니다.
- `table` - 복제할 테이블입니다.

> **Note**
>
> 참조 테이블을 만들 때 사용하는 표준 테이블에는 프라이머리 키가 있어야 합니다.
> 참조 테이블에서 CHECK 제약은 표현식을 지원하지 않습니다.
> 이전 함수인 `limitless_table_alter_type_reference`는 더 이상 사용되지 않습니다.

참조 테이블을 만드는 방법

다음 예시에서는 zipcodes 참조 테이블을 만드는 방법을 보여줍니다.

1. 표준 테이블을 만듭니다.

```
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);
```

2. 표준 테이블을 참조 테이블로 변환합니다.

```
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcodes');

postgres=> \d

                    List of relations
 Schema |     Name      |       Type        |       Owner
--------+---------------+-------------------+--------------------
 public | customer      | partitioned table | postgres_limitless
 public | customer_fs1  | foreign table     | postgres_limitless
 public | customer_fs2  | foreign table     | postgres_limitless
 public | customer_fs3  | foreign table     | postgres_limitless
 public | customer_fs4  | foreign table     | postgres_limitless
 public | customer_fs5  | foreign table     | postgres_limitless
 public | zipcodes      | foreign table     | postgres_limitless
(7 rows)
```

출력에는 customer 샤딩된 테이블과 zipcodes 참조 테이블이 표시됩니다.

Aurora PostgreSQL Limitless Database 샘플 스키마

Aurora PostgreSQL Limitless Database에 대한 다음 샘플 스키마를 제공합니다.

- Limitless E-Commerce 샘플 스키마
- Limitless pgbench

이러한 스키마를 사용하여 샘플 데이터베이스를 빠르게 만들고 Aurora PostgreSQL Limitless Database 테이블에 데이터를 로드할 수 있습니다. 자세한 내용은 GitHub 리포지토리를 참조하세요.

Aurora PostgreSQL Limitless Database에 데이터 로드

COPY 명령을 사용하거나 데이터 로드 유틸리티를 사용하여 Aurora PostgreSQL Limitless Database 테이블에 데이터를 로드할 수 있습니다.

> **Note**
>
> 표준 테이블, 샤딩된 테이블 및 참조 테이블에 데이터를 로드할 수 있습니다.

목차

Aurora PostgreSQL Limitless Database에서 COPY 명령 사용

psql 유틸리티의 \copy 기능을 사용하여 Aurora PostgreSQL Limitless Database로 데이터를 가져오고 내보낼 수 있습니다.

COPY 명령을 사용하여 Aurora PostgreSQL Limitless Database로 데이터 로드

Aurora PostgreSQL Limitless Database는 데이터 가져오기에서 psql 유틸리티의 \copy 기능과 호환됩니다.

Aurora PostgreSQL과 같은 Limitless Database에서는 다음이 지원되지 않습니다.

- DB 인스턴스에 대한 직접 SSH 액세스 - 데이터 파일(예: .csv 형식)을 DB 인스턴스 호스트에 복사하고 파일에서 COPY를 실행할 수 없습니다.
- DB 인스턴스에서 로컬 파일 사용 - COPY ... FROM STDIN 및 COPY ... TO STDOUT을 사용합니다.

PostgreSQL의 COPY 명령에는 로컬 파일(FROM/TO)로 작업하고 클라이언트와 서버(STDIN/STDOUT) 간의 연결을 사용하여 데이터를 전송하는 옵션이 있습니다. 자세한 내용은 PostgreSQL 설명서의 COPY를 참조하세요.

PostgreSQL psql 유틸리티의 \copy 명령은 psql 클라이언트를 실행하는 컴퓨터의 로컬 파일과 호환됩니다. 연결된 원격(예: Limitless Database) 서버에서 해당 COPY ... FROM STDIN 또는 COPY ... FROM STDOUT 명령을 간접 호출합니다. 로컬 파일에서 STDIN로 데이터를 읽거나 STDOUT에서 여기에 데이터를 씁니다.

데이터를 여러 파일로 분할

데이터는 Aurora PostgreSQL Limitless Database의 여러 샤드에 저장됩니다. \copy를 사용하여 데이터 로드 속도를 높이려면 데이터를 여러 파일로 분할할 수 있습니다. 그런 다음 별도의 \copy 명령을 병렬로 실행하여 각 데이터 파일에 대해 독립적으로 가져옵니다.

예를 들어 가져올 행이 300만 개인 CSV 형식의 입력 데이터 파일이 있습니다. 파일을 각각 20만 개의 행을 포함하는 청크(15개의 청크)로 분할할 수 있습니다.

```
split -l200000 data.csv data_ --additional-suffix=.csv -d
```

그러면 data_00.csv부터 data_14.csv까지의 파일이 생성됩니다. 그런 다음, 다음과 같은 15개의 병렬 \copy 명령을 사용하여 데이터를 가져올 수 있습니다.

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table from '/tmp/data_00.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_01.csv';" postgres_limitless &
...
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_13.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_14.csv';" postgres_limitless
```

이 기법을 사용하면 단일 \copy 명령을 사용하는 것보다 약 10배 더 빠르게 동일한 양의 데이터를 가져옵니다.

COPY 명령을 사용하여 Limitless Database 데이터를 파일로 복사

다음 예시와 같이 \copy 명령을 사용하여 무한 테이블의 데이터를 파일로 복사할 수 있습니다.

```
postgres_limitless=> \copy test_table TO '/tmp/test_table.csv' DELIMITER ',' CSV
 HEADER;
```

Aurora PostgreSQL Limitless Database 데이터 로드 유틸리티 사용

Aurora는 Aurora PostgreSQL DB 클러스터 또는 RDS for PostgreSQL DB 인스턴스에서 Limitless Database로 직접 데이터를 로드하는 유틸리티를 제공합니다.

데이터 로드 유틸리티를 사용하려면 다음 단계를 수행합니다.

1. 사전 조건
2. 소스 데이터베이스 준비
3. 대상 데이터베이스 준비
4. 데이터베이스 자격 증명 만들기
5. 다음 중 하나입니다.
   - 스크립트를 사용하여 데이터베이스 인증 및 리소스 액세스 설정(권장)
   - 데이터베이스 인증 및 리소스 액세스 수동 설정
6. Aurora PostgreSQL DB 클러스터 또는 RDS for PostgreSQL DB 인스턴스에서 데이터 로드

제한 사항

데이터 로드 유틸리티에는 다음과 같은 제한 사항이 있습니다.

- enum, `ARRAY`, `BOX`, `CIRCLE`, `LINE`, `LSEG`, `PATH`, `PG_LSN`, `PG_SNAPSHOT`, `POLYGON`, `TSQUERY`, `TSVECTOR`, `TXID_SNAPSHOT` 데이터 유형은 지원되지 않습니다.
- 로드 중에 `VARBIT` 데이터 유형에서 선행 0(`0`)이 제거됩니다.
- 소스 테이블에 복합 프라이머리 키가 있는 경우 데이터 마이그레이션이 실패합니다.
- 대상 테이블에 외래 키가 있는 경우 데이터 마이그레이션이 실패합니다.
- RDS for PostgreSQL 다중 AZ DB 클러스터에서 데이터 로드는 지원되지 않습니다.

사전 조건

데이터 로드 유틸리티에는 다음과 같은 사전 조건이 있습니다.

- 소스 데이터베이스는 Aurora PostgreSQL 또는 RDS for PostgreSQL 버전 11.x 이상을 사용해야 합니다.
- 소스 데이터베이스는 대상 DB 샤드 그룹과 동일한 AWS 계정 및 AWS 리전에 있어야 합니다.

- 소스 DB 클러스터 또는 DB 인스턴스가 `available` 상태여야 합니다.
- 소스 데이터베이스와 무한 데이터베이스의 테이블은 테이블 이름, 열 이름 및 열 데이터 유형이 동일해야 합니다.
- 소스 및 대상 테이블에는 열과 열 순서가 동일한 프라이머리 키가 있어야 합니다.
- 데이터 로드 명령을 실행하려면 무한 데이터베이스에 연결할 수 있는 환경이 있어야 합니다. 사용 가능한 명령은 다음과 같습니다.
  - `rds_aurora.limitless_data_load_start`
  - `rds_aurora.limitless_data_load_cancel`
- CDC의 경우:
  - 소스 데이터베이스와 대상 DB 샤드 그룹 모두 동일한 DB 서브넷 그룹, VPC 보안 그룹 및 데이터베이스 포트를 사용해야 합니다. 이러한 설정은 소스 데이터베이스와 DB 샤드 그룹의 라우터 모두에 대한 네트워크 연결을 위한 것입니다.
  - 소스 데이터베이스에서 논리적 복제를 활성화해야 합니다. 소스 데이터베이스 사용자에게 논리적 복제를 읽을 권한이 있어야 합니다.

소스 데이터베이스 준비

데이터 로드를 위해 소스 데이터베이스에 액세스하려면 수신 네트워크 트래픽을 허용해야 합니다. 다음 단계를 수행합니다.

소스 데이터베이스로의 네트워크 트래픽을 허용하는 방법

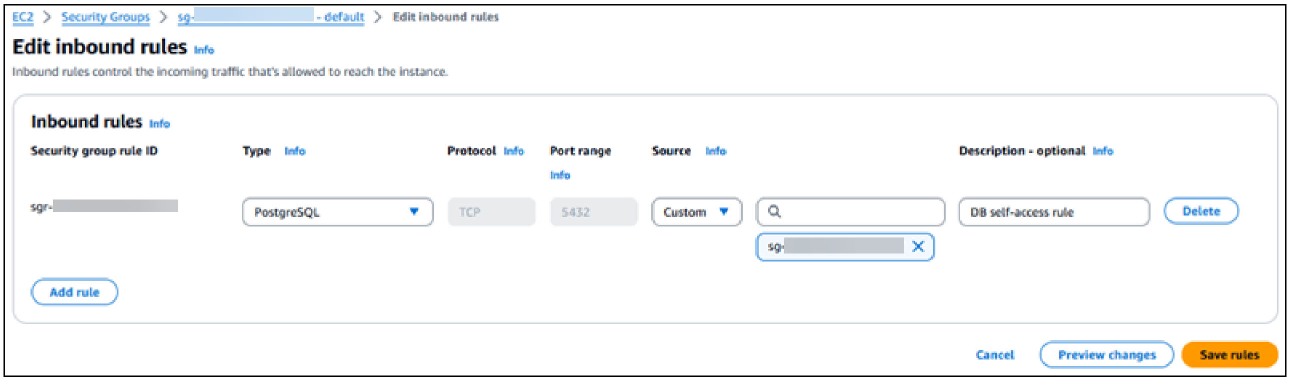
1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 보안 그룹 페이지로 이동합니다.
3. 소스 DB 클러스터 또는 인스턴스에서 사용하는 보안 그룹의 보안 그룹 ID를 선택합니다.

   예를 들어 보안 그룹 ID는 `sg-056a84f1712b77926`입니다.
4. 인바운드 규칙 탭에서 다음을 수행합니다.

   a. 인바운드 규칙 편집을 선택합니다.

   b. 소스 DB 클러스터 또는 인스턴스에 대한 새 인바운드 규칙을 추가합니다.

      - 포트 범위 - 소스 데이터베이스의 데이터베이스 포트로, 일반적으로 `5432`입니다.
      - 보안 그룹 ID - 이 예시에서는 `sg-056a84f1712b77926`입니다.



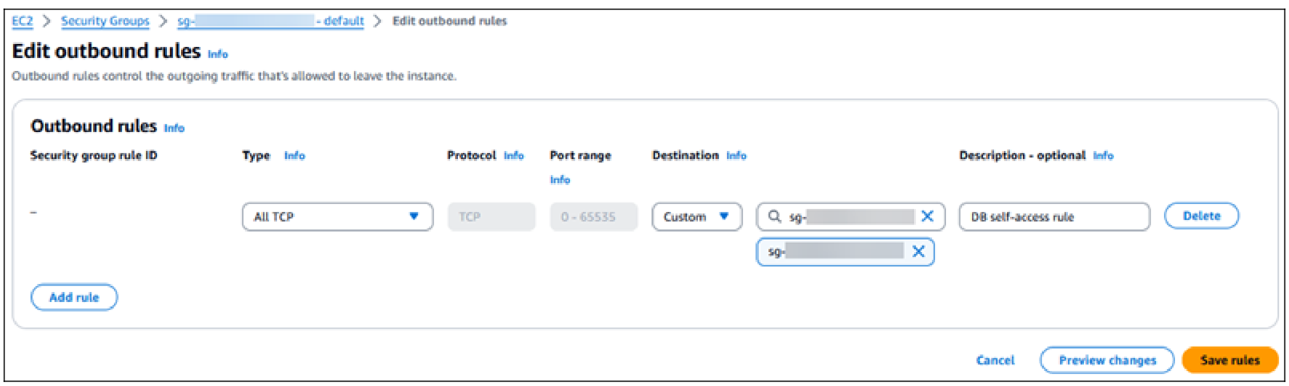
5. 아웃바운드 규칙 탭에서 다음을 수행합니다.

   a. 아웃바운드 규칙 편집을 선택합니다.

   b. 소스 DB 클러스터 또는 인스턴스에 대한 새 아웃바운드 규칙을 추가합니다.

      • 데이터베이스 포트 - `All traffic`(`0-65535` 포트 포함)

      • 보안 그룹 ID - 이 예시에서는 `sg-056a84f1712b77926`입니다.



6. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.

7. 네트워크 ACL 페이지로 이동합니다.

8. [기본 네트워크 ACL]에 설명된 대로 기본 네트워크 ACL 구성을 추가합니다.

대상 데이터베이스 준비

[Aurora PostgreSQL Limitless Database 테이블 만들기]의 프로시저에 따라 DB 샤드 그룹에 대상 테이블을 만듭니다.

대상 테이블에는 소스 테이블과 동일한 스키마, 테이블 이름 및 프라이머리 키가 있어야 합니다.

데이터베이스 자격 증명 만들기

소스 및 대상 데이터베이스에서 데이터베이스 사용자를 만들고 사용자에게 필요한 권한을 부여해야 합니다. 자세한 내용은 PostgreSQL 설명서에서 [CREATE USER](#) 및 [GRANT](#)를 참조하세요.

소스 데이터베이스 자격 증명 만들기

소스 데이터베이스 사용자가 명령에서 전달되어 로드를 시작합니다. 이 사용자에게는 소스 데이터베이스에서 복제를 수행할 권한이 있어야 합니다.

1. 데이터베이스 마스터 사용자(또는 `rds_superuser` 역할을 가진 다른 사용자)를 사용하여 `LOGIN` 권한이 있는 소스 데이터베이스 사용자를 만듭니다.

```
CREATE USER source_db_username WITH PASSWORD 'source_db_user_password';
```

2. 소스 데이터베이스 사용자에게 `rds_superuser` 역할을 부여합니다.

```
GRANT rds_superuser to source_db_username;
```

3. `full_load_and_cdc` 모드를 사용하는 경우 소스 데이터베이스 사용자에게 `rds_replication` 역할을 부여합니다. `rds_replication` 역할은 논리적 슬롯을 관리하고 논리적 슬롯을 사용하여 데이터를 스트리밍할 수 있는 권한을 부여합니다.

```
GRANT rds_replication to source_db_username;
```

대상 데이터베이스 자격 증명 만들기

대상 데이터베이스 사용자는 DB 샤드 그룹의 대상 테이블에 쓸 수 있는 권한이 있어야 합니다.

1. 데이터베이스 마스터 사용자(또는 `rds_superuser` 역할을 가진 다른 사용자)를 사용하여 `LOGIN` 권한이 있는 대상 데이터베이스 사용자를 만듭니다.

```
CREATE USER destination_db_username WITH PASSWORD 'destination_db_user_password';
```

2. 대상 데이터베이스 사용자에게 `rds_superuser` 역할을 부여합니다.

```
GRANT rds_superuser to destination_db_username;
```

스크립트를 사용하여 데이터베이스 인증 및 리소스 액세스 설정

설정 스크립트는 고객 관리형 AWS KMS key 1개, AWS Identity and Access Management(IAM) 역할 1개 및 AWS Secrets Manager 보안 암호 2개를 만듭니다.

설정 스크립트를 사용하려면 다음 단계를 수행합니다.

1. AWS CLI가 설치되고 AWS 계정 자격 증명으로 구성되어 있어야 합니다.
2. jq 명령줄 JSON 프로세서를 설치합니다. 자세한 내용은 jqlang/jq를 참조하세요.
3. data_loading_script.zip 파일을 컴퓨터에 복사하고 data_load_aws_setup_script.sh 파일을 추출합니다.
4. 스크립트를 편집하여 자리 표시자 변수를 다음에 적합한 값으로 바꿉니다.
   - 사용자의 AWS 계정
   - AWS 리전은
   - 소스 데이터베이스 자격 증명
   - 대상 데이터베이스 자격 증명
5. 컴퓨터에서 새 터미널을 열고 다음 명령을 실행합니다.

```
bash ./data_load_aws_setup_script.sh
```

데이터 로드 유틸리티용 설정 스크립트

여기에 참조용으로 data_load_aws_setup_script.sh 파일의 텍스트를 제공합니다.

```
#!/bin/bash
# Aurora Limitless data loading - AWS resources setup script #
# Set up the account credentials in advance. #
# Update the following script variables. #

####################################
#### Start of variable section ####

ACCOUNT_ID="12-digit_AWS_account_ID"
REGION="AWS_Region"
DATE=$(date +'%m%d%H%M%S')
RANDOM_SUFFIX="${DATE}"
SOURCE_SECRET_NAME="secret-source-${DATE}"
SOURCE_USERNAME="source_db_username"
```

```
SOURCE_PASSWORD="source_db_password"
DESTINATION_SECRET_NAME="secret-destination-${DATE}"
DESTINATION_USERNAME="destination_db_username"
DESTINATION_PASSWORD="destination_db_password"
DATA_LOAD_IAM_ROLE_NAME="aurora-data-loader-${RANDOM_SUFFIX}"
TMP_WORK_DIR="./tmp_data_load_aws_resource_setup/"

#### End of variable section ####
#################################

# Main logic start
echo "DATE - [${DATE}]"
echo "RANDOM_SUFFIX - [${RANDOM_SUFFIX}]"
echo 'START!'

mkdir -p $TMP_WORK_DIR

# Create the symmetric KMS key for encryption and decryption.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_key_response.txt"
aws kms create-key --region $REGION | tee $TMP_FILE_PATH
KMS_KEY_ARN=$(cat $TMP_FILE_PATH | jq -r '.KeyMetadata.Arn')
aws kms create-alias \
    --alias-name alias/"${DATA_LOAD_IAM_ROLE_NAME}-key" \
    --target-key-id $KMS_KEY_ARN \
    --region $REGION

# Create the source secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_source_secret_response.txt"
aws secretsmanager create-secret \
    --name $SOURCE_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$SOURCE_USERNAME\",\"password\":\"$SOURCE_PASSWORD
\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
SOURCE_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the destination secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_destination_secret_response.txt"
aws secretsmanager create-secret \
    --name $DESTINATION_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$DESTINATION_USERNAME\",\"password\":
\"$DESTINATION_PASSWORD\"}" \
```

```
    --region $REGION \
    | tee $TMP_FILE_PATH
DESTINATION_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the RDS trust policy JSON file.
# Use only rds.amazonaws.com for RDS PROD use cases.
TRUST_POLICY_PATH="${TMP_WORK_DIR}rds_trust_policy.json"
echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}' > $TRUST_POLICY_PATH

# Create the IAM role.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_iam_role_response.txt"
aws iam create-role \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --assume-role-policy-document "file://${TRUST_POLICY_PATH}" \
    --tags Key=assumer,Value=aurora_limitless_table_data_load \
    --region $REGION \
    | tee $TMP_FILE_PATH
IAM_ROLE_ARN=$(cat $TMP_FILE_PATH | jq -r '.Role.Arn')

# Create the permission policy JSON file.
PERMISSION_POLICY_PATH="${TMP_WORK_DIR}data_load_permission_policy.json"
permission_json_policy=$(cat &lt;&lt;EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "$SOURCE_SECRET_ARN",
                "$DESTINATION_SECRET_ARN"
            ]
        },
        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "$KMS_KEY_ARN"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
```

```
            "Resource": "*"
        }
    ]
}
EOF
)
echo $permission_json_policy > $PERMISSION_POLICY_PATH

# Add the inline policy.
aws iam put-role-policy \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --policy-name aurora-limitless-data-load-policy \
    --policy-document "file://${PERMISSION_POLICY_PATH}" \
    --region $REGION

# Create the key policy JSON file.
KEY_POLICY_PATH="${TMP_WORK_DIR}data_load_key_policy.json"
key_json_policy=$(cat &lt;&lt;EOF
{
    "Id": "key-aurora-limitless-data-load-$RANDOM_SUFFIX",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "$IAM_ROLE_ARN"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*"
        }
```

```
    ]
}
EOF
)
echo $key_json_policy > $KEY_POLICY_PATH

# Add the key policy.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_put_key_policy_response.txt"
sleep 10 # sleep 10 sec for IAM role ready
aws kms put-key-policy \
    --key-id $KMS_KEY_ARN \
    --policy-name default \
    --policy "file://${KEY_POLICY_PATH}" \
    --region $REGION \
    | tee $TMP_FILE_PATH

echo 'DONE!'

echo "ACCOUNT_ID : [${ACCOUNT_ID}]"
echo "REGION : [${REGION}]"
echo "RANDOM_SUFFIX : [${RANDOM_SUFFIX}]"
echo "IAM_ROLE_ARN : [${IAM_ROLE_ARN}]"
echo "SOURCE_SECRET_ARN : [${SOURCE_SECRET_ARN}]"
echo "DESTINATION_SECRET_ARN : [${DESTINATION_SECRET_ARN}]"

# Example of a successful run:
# ACCOUNT_ID : [012345678912]
# REGION : [ap-northeast-1]
# RANDOM_SUFFIX : [0305000703]
# IAM_ROLE_ARN : [arn:aws:iam::012345678912:role/aurora-data-loader-0305000703]
# SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-source-0305000703-yQDtow]
# DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-destination-0305000703-5d5Jy8]

# If you want to manually clean up failed resource,
# please remove them in the following order:
# 1. IAM role.
 # aws iam delete-role-policy --role-name Test-Role --policy-name ExamplePolicy --
region us-east-1
# aws iam delete-role --role-name Test-Role --region us-east-1
# 2. Source and destination secrets.
# aws secretsmanager delete-secret --secret-id MyTestSecret --force-delete-without-
recovery --region us-east-1
```

```
# 3. KDM key.
# aws kms schedule-key-deletion --key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab --pending-window-in-days 7
 --region us-east-1
```

데이터 로드 유틸리티 설정 스크립트의 출력

다음 예시는 스크립트의 성공적인 실행에서 얻은 출력을 보여줍니다.

```
% bash ./data_load_aws_setup_script.sh
DATE - [0305000703]
RANDOM_SUFFIX - [0305000703]
START!
{
    "KeyMetadata": {
        "AWSAccountId": "123456789012",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:ap-
northeast-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-03-05T00:07:49.852000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow",
    "Name": "secret-source-0305000703",
    "VersionId": "a017bebe-a71b-4220-b923-6850c2599c26"
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
destination-0305000703-5d5Jy8",
    "Name": "secret-destination-0305000703",
```

```
    "VersionId": "32a1f989-6391-46b1-9182-f65d242f5eb6"
}
{
    "Role": {
        "Path": "/",
        "RoleName": "aurora-data-loader-0305000703",
        "RoleId": "AROAYPX63ITQOYORQSC6U",
        "Arn": "arn:aws:iam::123456789012:role/aurora-data-loader-0305000703",
        "CreateDate": "2024-03-05T00:07:54+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "rds.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "assumer",
                "Value": "aurora_limitless_table_data_load"
            }
        ]
    }
}
DONE!
ACCOUNT_ID : [123456789012]
REGION : [ap-northeast-1]
RANDOM_SUFFIX : [0305000703]
IAM_ROLE_ARN : [arn:aws:iam::123456789012:role/aurora-data-loader-0305000703]
SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow]
DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:123456789012:secret:secret-destination-0305000703-5d5Jy8]
```

실패한 리소스 정리

실패한 리소스를 수동으로 정리하려면 다음 순서로 제거합니다.

1. IAM 역할. 예:

```
aws iam delete-role-policy \
--role-name Test-Role \
--policy-name ExamplePolicy

aws iam delete-role \
--role-name Test-Role
```

2. 소스 및 대상 보안 암호. 예:

```
aws secretsmanager delete-secret \
--secret-id MyTestSecret \
--force-delete-without-recovery
```

3. KMS 키. 예:

```
aws kms schedule-key-deletion \
--key-id arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab
 \
--pending-window-in-days 7
```

그런 다음 스크립트를 다시 시도할 수 있습니다.

데이터베이스 인증 및 리소스 액세스 수동 설정

데이터베이스 인증 및 리소스 액세스를 설정하는 수동 프로세스에는 다음 단계가 있습니다.

1. 고객 관리형 AWS KMS key 만들기
2. IAM 역할 권한 정책 추가
3. 데이터베이스 보안 암호 만들기
4. IAM 역할 만들기
5. 고객 관리형 AWS KMS key 업데이트

이 프로세스는 선택 사항이며 스크립트를 사용하여 데이터베이스 인증 및 리소스 액세스 설정에서와 동일한 작업을 수행합니다. 스크립트를 사용하는 것이 좋습니다.

고객 관리형 AWS KMS key 만들기

Creating symmetric encryption keys의 프로시저에 따라 고객 관리형 KMS 키를 만듭니다. 다음 요구 사항을 충족하는 경우 기존 키를 사용할 수도 있습니다.

고객 관리형 KMS 키를 만드는 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/kms에서 AWS KMS 콘솔을 엽니다.
2. 고객 관리형 키 페이지로 이동합니다.
3. 키 생성을 선택합니다.
4. 키 구성 페이지에서 다음 단계를 따릅니다.
   a. 키 유형에서 대칭을 선택합니다.
   b. 키 사용에서 암호화 및 해독을 선택합니다.
   c. 다음을 선택합니다.
5. 레이블 추가 페이지에서 **`limitless`**와 같은 별칭을 입력한 후 다음을 선택합니다.
6. 키 관리 권한 정의 페이지에서 키 관리자가 이 키를 삭제하도록 허용 확인란이 선택되어 있는지 확인한 후 다음을 선택합니다.
7. 키 사용 권한 정의 페이지에서 다음을 선택합니다.
8. 검토 페이지에서 완료를 선택합니다.

   나중에 키 정책을 업데이트합니다.

IAM 역할 권한 정책 추가에서 사용할 KMS 키의 Amazon 리소스 이름(ARN)을 기록합니다.

AWS CLI를 사용하여 고객 관리형 KMS 키를 만드는 방법에 대한 자세한 내용은 create-key 및 create-alias를 참조하세요.

데이터베이스 보안 암호 만들기

데이터 로드 유틸리티가 소스 및 대상 데이터베이스 테이블에 액세스할 수 있도록 하려면 AWS Secrets Manager에서 두 개의 보안 암호를 만듭니다. 하나는 소스 데이터베이스용이고 다른 하나는 대상 데이터베이스용입니다. 이러한 보안 암호는 소스 및 대상 데이터베이스에 액세스하기 위한 사용자 이름과 암호를 저장합니다.

Create an AWS Secrets Manager secret의 프로시저에 따라 키-값 페어 보안 암호를 만듭니다.

데이터베이스 보안 암호를 만드는 방법

1. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
2. Store a new secret(새 보안 암호 저장)을 선택합니다.
3. 보안 암호 유형 선택 페이지에서 다음을 수행합니다.
   a. 보안 암호 유형에서 다른 유형의 보안 암호를 선택합니다.
   b. 키/값 페어에서 일반 텍스트 탭을 선택합니다.
   c. 다음 JSON 코드를 입력합니다. 여기서 *sourcedbreader* 및 *sourcedbpassword*는 소스 데이터베이스 자격 증명 만들기의 소스 데이터베이스 사용자의 자격 증명입니다.

      ```
      {
          "username":"sourcedbreader",
          "password":"sourcedbpassword"
      }
      ```
   d. 암호화 키에서 고객 관리형 AWS KMS key 만들기에서 만든 KMS 키(예: `limitless`)를 선택합니다.
   e. 다음을 선택합니다.
4. 보안 암호 구성 페이지에서 보안 암호 이름(예: **`source_DB_secret`**)을 입력한 후 다음을 선택합니다.
5. 교체 구성 - 선택 사항 페이지에서 다음을 선택합니다.
6. 검토(Review) 페이지에서 시작(Store)을 선택합니다.
7. 대상 데이터베이스 보안 암호에 대해 같은 프로시저를 반복합니다.

a. 다음 JSON 코드를 입력합니다. 여기서 *destinationdbwriter* 및 *destinationdbpassword*는 대상 데이터베이스 자격 증명 만들기의 대상 데이터베이스 사용자의 자격 증명입니다.

```
{
    "username":"destinationdbwriter",
    "password":"destinationdbpassword"
}
```

b. 보안 암호 이름(예: **destination_DB_secret**)을 입력합니다.

IAM 역할 권한 정책 추가에서 사용할 보안 암호의 ARN을 기록합니다.

IAM 역할 만들기

데이터를 로드하려면 AWS 리소스에 대한 액세스를 제공해야 합니다. 액세스를 제공하려면 IAM 사용자에게 권한을 위임할 역할 생성의 프로시저를 따라 `aurora-data-loader` IAM 역할을 만듭니다.

IAM 역할을 만들려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 역할 페이지로 이동합니다.
3. 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.

   a. 신뢰할 수 있는 엔터티 유형에서 사용자 지정 신뢰 정책을 선택합니다.

   b. 사용자 지정 신뢰 정책에 대해 다음 JSON 코드를 입력합니다.

      JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
```

```
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   c. 다음을 선택합니다.

5. 권한 추가 페이지에서 다음을 선택합니다.
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.

   a. 역할 이름에 **aurora-data-loader** 또는 원하는 다른 이름을 입력합니다.
   b. 태그 추가를 선택하고 다음 태그를 입력합니다.

      - 키: **assumer**
      - 값: **aurora_limitless_table_data_load**

> Important
>
> Aurora PostgreSQL Limitless Database는 이 태그가 있는 IAM 역할만 수임할 수 있습니다.

   c. 역할 생성을 선택합니다.

고객 관리형 AWS KMS key 업데이트

[키 정책 변경](#)의 프로시저에 따라 `aurora-data-loader` IAM 역할을 기본 키 정책에 추가합니다.

키 정책에 IAM 역할을 추가하는 방법

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/kms에서 AWS KMS 콘솔을 엽니다.
2. 고객 관리형 키 페이지로 이동합니다.
3. 고객 관리형 AWS KMS key 만들기에서 만든 KMS 키(예: `limitless`)를 선택합니다.
4. 키 정책 탭의 키 사용자에서 추가를 선택합니다.
5. 키 사용자 추가 창에서 IAM 역할 만들기에서 만든 IAM 역할의 이름(예: aurora-data-loader)을 선택합니다.

6. 추가를 선택합니다.

IAM 역할 권한 정책 추가

만든 IAM 역할에 권한 정책을 추가해야 합니다. 이를 통해 Aurora PostgreSQL Limitless Database 데이터 로드 유틸리티는 네트워크 연결을 구축하고 소스 및 대상 DB 자격 증명 보안 암호를 검색하기 위해 관련 AWS 리소스에 액세스할 수 있습니다.

자세한 내용은 역할 수정을 참조하세요.

권한 정책을 추가하는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 역할 페이지로 이동합니다.
3. aurora-data-loader와 같이 IAM 역할 만들기에서 만든 IAM 역할을 선택합니다.
4. 권한 탭의 권한 정책에서 권한 추가를 선택한 다음 인라인 정책 생성을 선택합니다.
5. 권한 지정 페이지에서 JSON 편집기를 선택합니다.
6. 다음 템플릿을 복사하여 JSON 편집기에 붙여넣고 자리 표시자를 데이터베이스 보안 암호 및 KMS 키의 ARN으로 바꿉니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
```

```
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:source_DB_secret-ABC123",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:destination_DB_secret-456DEF"
            ]
        },          {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/aa11bb22-
####-####-####-fedcba123456"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

7. 오류를 확인하고 수정합니다.
8. 다음을 선택합니다.
9. 검토 및 생성 페이지에 정책 이름(예: **`data_loading_policy`**)을 입력한 후 정책 생성을 선택합니다.

Aurora PostgreSQL DB 클러스터 또는 RDS for PostgreSQL DB 인스턴스에서 데이터 로드

리소스 및 인증 설정을 완료한 후 클러스터 엔드포인트에 연결하고 `postgres_limitless`와 같은 무한 데이터베이스에서 `rds_aurora.limitless_data_load_start` 저장 프로시저를 직접 호출합니다. 무한 데이터베이스는 DB 샤드 그룹의 데이터베이스로, 데이터를 마이그레이션하려는 곳입니다.

이 함수는 백그라운드에서 명령에 지정된 소스 데이터베이스에 비동기식으로 연결하고, 소스에서 데이터를 읽고, 데이터를 샤드에 로드합니다. 성능을 높이기 위해 병렬 스레드를 사용하여 데이터를 로드합니다. 함수는 `SELECT` 명령을 실행하여 명령에 제공된 테이블의 데이터를 읽어 특정 시점 테이블 스냅샷을 검색합니다.

샤딩된 테이블, 참조 테이블 및 표준 테이블에 데이터를 로드할 수 있습니다.

`rds_aurora.limitless_data_load_start` 직접 호출에서 데이터베이스, 스키마 또는 테이블 수준에서 데이터를 로드할 수 있습니다.

- 데이터베이스 - 각 직접 호출에서 한 번에 하나의 데이터베이스를 로드할 수 있으며, 데이터베이스 내의 스키마 또는 테이블 수에는 제한이 없습니다.
- 스키마 - 각 직접 호출에서 최대 15개의 스키마를 로드할 수 있으며, 각 스키마 내의 테이블 수에는 제한이 없습니다.
- 테이블 - 각 직접 호출에서 최대 15개의 테이블을 로드할 수 있습니다.

> **Note**
>
> 이 기능은 Amazon RDS 스냅샷 또는 데이터베이스의 특정 시점 격리를 사용하지 않습니다. 테이블 간의 일관성을 위해 소스 데이터베이스를 복제하고 복제된 데이터베이스를 소스로 가리키는 것이 좋습니다.

저장 프로시저는 다음과 같은 구문을 사용합니다.

```
CALL rds_aurora.limitless_data_load_start('source_type',
    'source_DB_cluster_or_instance_ID',
    'source_database_name',
    'streaming_mode',
    'data_loading_IAM_role_arn',
    'source_DB_secret_arn',
    'destination_DB_secret_arn',
    'ignore_primary_key_conflict_boolean_flag',
    'is_dry_run',
    (optional parameter) schemas/tables => ARRAY['name1', 'name2', ...]);
```

입력 파라미터는 다음과 같습니다.

- `source_type` – 소스 유형: `aurora_postgresql` 또는 `rds_postgresql`
- `source_DB_cluster_or_instance_ID` – 소스 Aurora PostgreSQL DB 클러스터 식별자 또는 RDS for PostgreSQL DB 인스턴스 식별자
- `source_database_name` - 소스 데이터베이스 이름. 예: *postgres*
- `streaming_mode` – 변경 데이터 캡처(CDC)를 포함할지 여부: `full_load` 또는 `full_load_and_cdc`
- `data_loading_IAM_role_arn` - `aurora-data-loader`에 대한 IAM 역할 Amazon 리소스 이름(ARN)
- `source_DB_secret_arn` - 소스 DB 보안 암호 ARN
- `destination_DB_secret_arn` - 대상 DB 보안 암호 ARN
- `ignore_primary_key_conflict_boolean_flag` - 프라이머리 키 충돌이 발생하는 경우 계속할지 여부:
  - `true`로 설정하면 데이터 로드는 프라이머리 키 충돌이 있는 행의 새 변경 사항을 무시합니다.
  - `false`로 설정하면 프라이머리 키 충돌이 발생할 때 데이터 로드가 대상 테이블의 기존 행을 덮어씁니다.
- `is_dry_run` - 데이터 로드 작업이 소스 및 대상 데이터베이스에 연결할 수 있는지 테스트할지 여부:
  - `true`로 설정하면 데이터를 로드하지 않고 연결을 테스트합니다.
  - `false`로 설정하면 데이터를 로드합니다.
- (선택 사항) `schemas` 또는 `tables` - 로드할 스키마 또는 테이블의 배열. 다음 중 하나를 지정할 수 있습니다.

- tables => ARRAY['*schema1.table1*', '*schema1.table2*', '*schema2.table1*', ...] 형식의 테이블 목록
- schemas => ARRAY['*schema1*', '*schema2*', ...] 형식의 스키마 목록

이 파라미터를 포함하지 않으면 지정된 전체 소스 데이터베이스가 마이그레이션됩니다.

출력 파라미터는 메시지가 있는 작업 ID입니다.

다음 예시에서는 `rds_aurora.limitless_data_load_start` 저장 프로시저를 사용하여 Aurora PostgreSQL DB 클러스터에서 데이터를 로드하는 방법을 보여줍니다.

```
CALL rds_aurora.limitless_data_load_start('aurora_postgresql',
    'my-db-cluster',
    'postgres',
    'full_load_and_cdc',
    'arn:aws:iam::123456789012:role/aurora-data-loader-8f2c66',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-source-8f2c66-EWrr0V',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-destination-8f2c66-
d04fbD',
    'true',
    'false',
    tables => ARRAY['public.customer', 'public.order', 'public.orderdetails']);

INFO: limitless data load job id 1688761223647 is starting.
```

데이터 로드 모니터링

Aurora PostgreSQL Limitless Database는 데이터 로드 작업을 모니터링하는 몇 가지 방법을 제공합니다.

- 데이터 로드 작업 나열
- 작업 ID를 사용하여 데이터 로드 작업의 세부 정보 보기
- Amazon CloudWatch 로그 그룹 모니터링
- RDS 이벤트 모니터링

데이터 로드 작업 나열

클러스터 엔드포인트에 연결하고 `rds_aurora.limitless_data_load_jobs` 뷰를 사용하여 데이터 로드 작업을 나열할 수 있습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_jobs LIMIT 6;

    job_id      |  status    | message |     source_db_identifier      | source_db_name |
 full_load_complete_time |
   progress_details                                                                    |
     start_time        |   last_updated_time     |  streaming_mode    | source_engine_type
 | ignore_primary_key_conflict | is_dryrun
---------------+-----------+---------+--------------------------------+----------------
+-------------------------
+----------------------------------------------------------------------------------------
+-------------------------+------------------------+--------------------
+--------------------+-----------------------------+-----------
 1725697520693 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:48:15+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "9
 of 9 tables loaded", "COMPLETED_AT": "2024/09/07 08:48:15+00", "RECORDS_MIGRATED":
 600003}} | 2024-09-07 08:47:13+00 | 2024-09-07 08:48:15+00 | full_load          |
 aurora_postgresql   | t                           | f
 1725696114225 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:24:20+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "3
 of 3 tables loaded", "COMPLETED_AT": "2024/09/07 08:24:20+00", "RECORDS_MIGRATED":
 200001}} | 2024-09-07 08:23:56+00 | 2024-09-07 08:24:20+00 | full_load          |
 aurora_postgresql   | t                           | f
 1725696067630 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:23:45+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "6
 of 6 tables loaded", "COMPLETED_AT": "2024/09/07 08:23:45+00", "RECORDS_MIGRATED":
```

```
 400002}} | 2024-09-07 08:23:10+00 | 2024-09-07 08:23:45+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725694221753 | CANCELED  |          | persistent-kdm-auto-source-01 | postgres
    |                          | {}

           | 2024-09-07 07:31:18+00 | 2024-09-07 07:51:49+00 | full_load_and_cdc |
 aurora_postgresql  | t                           | f
 1725691698210 | COMPLETED |          | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:51+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:51+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:42+00 | 2024-09-07 07:10:52+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725691695049 | COMPLETED |          | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:48+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:48+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:41+00 | 2024-09-07 07:10:48+00 | full_load         |
 aurora_postgresql  | t                           | f
(6 rows)
```

작업 레코드는 90일 후에 삭제됩니다.

작업 ID를 사용하여 데이터 로드 작업의 세부 정보 보기

작업 ID를 알고 있는 경우 클러스터 엔드포인트에 연결하고 `rds_aurora.limitless_data_load_job_details` 뷰를 사용하여 테이블 이름, 작업 상태 및 로드된 행 수를 포함하여 해당 데이터 로드 작업의 세부 정보를 볼 수 있습니다. 데이터 로드 시작 함수에 대한 응답 또는 `rds_aurora.limitless_data_load_jobs` 뷰에서 작업 ID를 가져올 수 있습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_job_details WHERE
 job_id='1725696114225';

job_id        | destination_table_name | destination_schema_name | start_time
      | status    | full_load_rows | full_load_total_rows | full_load_complete_time |
 cdc_insert | cdc_update | cdc_delete
--------------+------------------------+-------------------------
+------------------------+-----------+----------------+---------------------
+------------------------+------------+-----------+------------
1725696114225 | standard_1             | public                  | 2024-09-07
 08:23:57+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:08+00  | 0          | 0          | 0
1725696114225 | standard_2             | public                  | 2024-09-07
 08:24:08+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:17+00  | 0          | 0          | 0
```

```
1725696114225 | standard_3             | public                     | 2024-09-07
 08:24:18+00 | COMPLETED | 1             | 1                          | 2024-09-07
 08:24:20+00  | 0           | 0           | 0
1725696114225 | standard_4             | public                     | 2024-09-07
 08:23:58+00 | PENDING   | 0             | 0                          |
    | 0           | 0           | 0
(4 rows)
```

작업 레코드는 90일 후에 삭제됩니다.

Amazon CloudWatch 로그 그룹 모니터링

데이터 로드 작업 상태가 RUNNING으로 변경되면 Amazon CloudWatch Logs를 사용하여 런타임 진행 상황을 확인할 수 있습니다.

CloudWatch 로그 스트림을 모니터링하는 방법

AWS Management Console에 로그인하고 https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

1. 로그로 이동한 다음 로그 그룹으로 이동합니다.
2. /aws/rds/aurora-limitless-database 로그 그룹을 선택합니다.
3. job_id별로 데이터 로드 작업의 로그 스트림을 검색합니다.

   로그 스트림의 패턴은 Data-Load-Job-***job_id***입니다.
4. 로그 스트림을 선택하여 로그 이벤트를 확인합니다.

각 로그 스트림에는 작업 상태와 Aurora PostgreSQL Limitless Database 대상 테이블에 로드된 행 수가 포함된 이벤트가 표시됩니다. 데이터 로드 작업이 실패하면 실패 상태와 이유를 보여주는 오류 로그도 만들어집니다.

작업 레코드는 90일 후에 삭제됩니다.

RDS 이벤트 모니터링

또한 데이터 로드 작업은 작업이 성공, 실패 또는 취소되는 경우와 같은 RDS 이벤트를 게시합니다. 대상 데이터베이스에서 이벤트를 볼 수 있습니다.

자세한 내용은 ??? 단원을 참조하십시오.

데이터 로드 취소

데이터 로드 작업을 취소하려면 작업 ID를 입력 파라미터로 사용하여 `rds_aurora.limitless_data_load_cancel` 저장 프로시저를 직접 호출합니다. 특정 데이터 로드 작업이 시작된 DB 샤드 그룹의 동일한 데이터베이스에서 이 저장 프로시저를 직접 호출합니다. 예시:

```
CALL rds_aurora.limitless_data_load_cancel(12345);

INFO: limitless data load job with id 12345 is canceling without rollback.
```

존재하지 않거나 동일한 DB 샤드 그룹에서 실행되지 않는 데이터 로드 작업은 취소할 수 없습니다.

Aurora PostgreSQL Limitless Database 데이터 로드 유틸리티는 응답에 표시된 대로 로드된 데이터를 롤백 없이 대상 테이블에 남깁니다. 로드된 데이터를 유지하지 않으려면 대상 테이블을 잘라내는 것이 좋습니다.

Aurora PostgreSQL Limitless Database 쿼리

Aurora PostgreSQL Limitless Database는 쿼리에 대한 PostgreSQL 구문과 호환됩니다. `psql` 또는 PostgreSQL에서 작동하는 다른 연결 유틸리티를 사용하여 Limitless Database를 쿼리할 수 있습니다. 쿼리를 실행하려면 Aurora PostgreSQL Limitless Database DB 클러스터에 연결에서와 같이 무한 엔드포인트에 연결합니다.

모든 PostgreSQL `SELECT` 쿼리는 Aurora PostgreSQL Limitless Database에서 지원됩니다. 그러나 쿼리는 두 계층에서 수행됩니다.

1. 클라이언트가 쿼리를 전송하는 라우터
2. 데이터가 있는 샤드

성능은 서로 다른 샤드에서 서로 다른 쿼리에 대해 높은 수준의 동시 처리를 허용하는 방식으로 데이터베이스를 쿼리하는 데 달려 있습니다. 쿼리는 먼저 분산 트랜잭션 계층(라우터)에서 구문 분석됩니다. 쿼리 실행을 계획하기 전에 쿼리에 참여하는 모든 관계의 위치를 식별하는 분석 단계가 있습니다. 모든 관계가 동일한 샤드 또는 참조 테이블에 필터링된 샤드 키가 있는 샤드 테이블인 경우 쿼리 계획은 라우터 계층에서 건너뛰고 계획 및 실행을 위해 샤드까지 완전히 푸시다운됩니다. 이 프로세스는 서로 다른 노드(라우터 및 샤드) 간의 왕복 횟수를 줄이고 대부분의 경우 더 나은 성능을 제공합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database의 단일 샤드 쿼리 단원을 참조하십시오.

> **Note**
> 샤드와 별도로 데이터를 검색하여 쿼리가 더 잘 수행되는 카테시안 곱(교차 조인)과 같은 특정 경우가 있을 수 있습니다.

쿼리 실행 계획에 대한 자세한 내용은 Aurora PostgreSQL Limitless Database 참조에서 EXPLAIN 섹션을 참조하세요. 쿼리에 대한 일반적인 내용은 PostgreSQL 설명서에서 Queries를 참조하세요.

주제

- Aurora PostgreSQL Limitless Database의 단일 샤드 쿼리
- Aurora PostgreSQL Limitless Database의 분산 쿼리
- Aurora PostgreSQL Limitless Database의 PostgreSQL 로그에서 분산 쿼리 추적
- Aurora PostgreSQL Limitless Database의 분산 교착

Aurora PostgreSQL Limitless Database의 단일 샤드 쿼리

단일 샤드 쿼리는 SQL ACID 의미 체계를 유지하면서 샤드에서 직접 실행할 수 있는 쿼리입니다. 라우터의 쿼리 플래너가 이러한 쿼리를 발견하면 플래너는 이를 감지하고 전체 SQL 쿼리를 해당 샤드로 푸시다운합니다.

이 최적화를 통해 라우터와 샤드 간의 네트워크 왕복 횟수를 줄여 성능을 높일 수 있습니다. 현재 이 최적화는 `INSERT`, `SELECT`, `UPDATE` 및 `DELETE` 쿼리에 대해 수행됩니다.

주제

- 단일 샤드 쿼리 예시
- 단일 샤드 쿼리에 대한 제한 사항
- 완전한 자격이 있는(명시적인) 조인
- 활성 샤드 키 설정

단일 샤드 쿼리 예시

다음 예시에서는 샤드 키 `customer_id`와 참조 테이블 `zipcodes`가 있는 샤딩된 테이블 `customers`가 있습니다.

SELECT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE
 customer_id = 100;

                       QUERY PLAN
---------------------------------------------------------
 Foreign Scan
   Output: customer_id, other_id, customer_name, balance
   Remote SQL:  SELECT customer_id,
     other_id,
     customer_name,
     balance
    FROM public.customers
   WHERE (customer_id = 100)
 Single Shard Optimized
(9 rows)
```

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM orders
```

```
    LEFT JOIN zipcodes ON orders.zipcode_id = zipcodes.zipcode_id
    WHERE customer_id = 11;

                                            QUERY PLAN

------------------------------------------------------------------------------------------------
 Foreign Scan
   Output: customer_id, order_id, zipcode_id, customer_name, balance,
 zipcodes.zipcode_id, zipcodes.city
   Remote SQL:  SELECT orders.customer_id,
     orders.order_id,
     orders.zipcode_id,
     orders.customer_name,
     orders.balance,
     zipcodes.zipcode_id,
     zipcodes.city
    FROM (public.orders
      LEFT JOIN public.zipcodes ON ((orders.zipcode_id = zipcodes.zipcode_id)))
   WHERE (orders.customer_id = 11)
 Single Shard Optimized
(13 rows)
```

INSERT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) INSERT INTO customers
    (customer_id, other_id, customer_name, balance)
    VALUES (1, 10, 'saikiran', 1000);

                      QUERY PLAN
-------------------------------------------------------
 Insert on public.customers
   ->  Result
         Output: 1, 10, 'saikiran'::text, '1000'::real
 Single Shard Optimized
(4 rows)
```

UPDATE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) UPDATE orders SET balance = balance +
 100
    WHERE customer_id = 100;
```

```
                                          QUERY PLAN

-------------------------------------------------------------------------------------------------
 Update on public.orders
   Foreign Update on public.orders_fs00002 orders_1
   ->  Foreign Update
         Remote SQL:  UPDATE public.orders SET balance = (balance + (100)::double
 precision)
   WHERE (customer_id = 100)
 Single Shard Optimized
(6 rows)
```

DELETE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) DELETE FROM orders
    WHERE customer_id = 100 and balance = 0;

                             QUERY PLAN
---------------------------------------------------------------------
 Delete on public.orders
   Foreign Delete on public.orders_fs00002 orders_1
   ->  Foreign Delete
         Remote SQL:  DELETE FROM public.orders
   WHERE ((customer_id = 100) AND (balance = (0)::double precision))
 Single Shard Optimized
(6 rows)
```

단일 샤드 쿼리에 대한 제한 사항

단일 샤드 쿼리에는 다음과 같은 제한 사항이 있습니다.

함수

단일 샤드 쿼리에 함수가 포함된 경우 쿼리는 다음 조건 중 하나가 적용되는 경우에만 단일 샤드 최적화 자격을 얻습니다.

- 함수를 변경할 수 없는 경우 자세한 내용은 함수 휘발성 섹션을 참조하세요.
- 함수를 변경할 수 있지만 rds_aurora.limitless_distributed_functions 뷰에 등록되어 있는 경우. 자세한 내용은 함수 배포 섹션을 참조하세요.

보기

쿼리에 하나 이상의 뷰가 포함된 경우 쿼리에 대해 다음 조건 중 하나가 있는 경우 단일 샤드 최적화가 비활성화됩니다.

- 뷰에 `security_barrier` 속성이 있는 경우
- 쿼리에 사용되는 객체에 여러 사용자 권한이 필요한 경우. 예를 들어 쿼리에 두 개의 뷰가 포함되어 있으며, 뷰가 두 명의 다른 사용자로 실행되는 경우입니다.

```
CREATE VIEW v1 AS SELECT customer_name FROM customers c WHERE c.customer_id =  1;
CREATE VIEW v2 WITH (security_barrier) AS SELECT customer_name FROM customers c
 WHERE c.customer_id =  1;

postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v1;
                                     QUERY PLAN
-----------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Output: customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 c  (cost=0.00..24.12 rows=6 width=32)
           Output: c.customer_name
           Filter: (c.customer_id = 1)
         Query Identifier: -6005737533846718506
   Remote SQL:  SELECT customer_name
    FROM ( SELECT c.customer_name
            FROM public.customers c
           WHERE (c.customer_id = 1)) v1
 Query Identifier: -5754424854414896228
(12 rows)


postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v2;
                                         QUERY PLAN
---------------------------------------------------------------------------------------------
 Foreign Scan on public.customers_fs00001 c  (cost=100.00..128.41 rows=7 width=32)
   Output: c.customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 customers  (cost=0.00..24.12 rows=6
 width=32)
           Output: customers.customer_name
           Filter: (customers.customer_id = 1)
         Query Identifier: 4136563775490008117
   Remote SQL: SELECT customer_name FROM public.customers WHERE ((customer_id = 1))
```

```
 Query Identifier: 5056054318010163757
(9 rows)
```

PREPARE 및 EXECUTE 문

Aurora PostgreSQL Limitless Database는 준비된 SELECT, UPDATE 및 DELETE 문에 대한 단일 샤드 최적화를 지원합니다.

그러나 plan_cache_mode가 'force_generic_plan'으로 설정된 상태로 PREPARE 및 EXECUTE에 준비된 문을 사용하는 경우 쿼리 플래너는 해당 쿼리에 대한 단일 샤드 최적화를 거부합니다.

PL/pgSQL

PL/pgSQL 변수가 있는 쿼리는 암시적으로 준비된 문으로 실행됩니다. 쿼리에 PL/pgSQL 변수가 포함된 경우 쿼리 플래너는 단일 샤드 최적화를 거부합니다.

문에 PL/pgSQL 변수가 포함되지 않은 경우 PL/pgSQL 블록에서 최적화가 지원됩니다.

완전한 자격이 있는(명시적인) 조인

단일 샤드 최적화는 파티션 제거를 기반으로 합니다. PostgreSQL 옵티마이저는 일정한 조건에 따라 파티션을 제거합니다. Aurora PostgreSQL Limitless Database에서 나머지 파티션과 테이블이 모두 동일한 샤드에 있는 것으로 확인되면 쿼리가 단일 샤드 최적화에 적합한 것으로 표시됩니다. 파티션 제거가 작동하려면 모든 필터 조건이 명시적이어야 합니다. Aurora PostgreSQL Limitless Database는 문에 있는 모든 샤딩된 테이블의 샤드 키에 조인 조건자 또는 필터 조건자가 하나 이상 있지 않은 경우 파티션을 제거할 수 없습니다.

customer_id 열을 기반으로 customers, orders 및 order_details 테이블을 파티셔닝했다고 가정합니다. 이 스키마에서 애플리케이션은 고객의 모든 데이터를 단일 샤드에 유지하려고 시도합니다.

다음과 같은 쿼리를 가정합니다.

```
SELECT * FROM
    customers c, orders o, order_details od
WHERE c.customer_id = o.customer_id
    AND od.order_id = o.order_id
    AND c.customer_id = 1;
```

이 쿼리는 고객(`c.customer_id = 1`)의 모든 데이터를 검색합니다. 이 고객의 데이터는 단일 샤드에 있지만 Aurora PostgreSQL Limitless Database는이 쿼리에 단일 샤드 쿼리 자격을 부여하지 않습니다. 쿼리에 대한 옵티마이저 프로세스는 다음과 같습니다.

1. 옵티마이저는 다음 조건에 따라 `customers` 및 `orders`에 대한 파티션을 제거할 수 있습니다.

```
c.customer_id = 1
c.customer_id = o.customer_id
o.customer_id =  1 (transitive implicit condition)
```

2. 테이블에 상수 조건이 없으므로 옵티마이저는 `order_details`에 대한 파티션을 제거할 수 없습니다.
3. 옵티마이저는 `order_details`에서 모든 파티션을 읽었다고 결론을 내립니다. 따라서 쿼리는 단일 샤드 최적화 자격을 얻지 못합니다.

이를 단일 샤드 쿼리로 만들기 위해 다음과 같은 명시적 조인 조건을 추가합니다.

```
o.customer_id = od.customer_id
```

변경된 쿼리는 다음과 같습니다.

```
SELECT * FROM
    customers c, orders o,  order_details od
WHERE c.customer_id = o.customer_id
     AND o.customer_id = od.customer_id
     AND od. order_id = o. order_id
 AND c.customer_id =  1;
```

이제 옵티마이저가 `order_details`의 파티션을 제거할 수 있습니다. 새 쿼리는 단일 샤드 쿼리가 되고 최적화 자격을 얻습니다.

활성 샤드 키 설정

이 기능을 사용하면 데이터베이스를 쿼리하는 동안 단일 샤드 키를 설정할 수 있으므로 모든 SELECT 및 DML 쿼리가 샤드 키와 함께 상수 조건자로 추가됩니다. 이 기능은 Aurora PostgreSQL Limitless Database로 마이그레이션하고 테이블에 샤드 키를 추가하여 스키마를 비정규화한 경우에 유용합니다.

쿼리의 의미 체계를 변경하지 않고 기존 SQL 로직에 샤드 키 조건자를 자동으로 추가할 수 있습니다. 활성 샤드 키 조건자 추가는 [호환되는 테이블](#)에 대해서만 수행됩니다.

활성 샤드 키 기능은 다음 구문을 가진 rds_aurora.limitless_active_shard_key 변수를 사용합니다.

```
SET [session | local] rds_aurora.limitless_active_shard_key = '{"col1_value",
 "col2_value", ...}';
```

활성 샤드 키 및 외래 키에 대한 몇 가지 고려 사항:

- 상위 테이블과 하위 테이블이 공동 배치되고 외래 키가 샤드 키의 슈퍼 집합인 경우 샤딩된 테이블에 외래 키 제약이 있을 수 있습니다.
- 샤딩된 테이블에는 참조 테이블에 대한 외래 키 제약이 있을 수 있습니다.
- 참조 테이블에는 다른 참조 테이블에 대한 외래 키 제약이 있을 수 있습니다.

customer_id 열에 샤딩된 customers 테이블이 있다고 가정합니다.

```
BEGIN;
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE customers(customer_id int PRIMARY KEY, name text , email text);
COMMIT;
```

활성 샤드 키 세트의 경우 쿼리의 변환은 다음과 같습니다.

SELECT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
SELECT * FROM customers;

-- This statement is changed to:
SELECT * FROM customers WHERE customer_id = '123'::int;
```

INSERT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
INSERT INTO customers(name, email) VALUES('Alex', 'alex@example.com');

-- This statement is changed to:
INSERT INTO customers(customer_id, name, email) VALUES('123'::int, 'Alex',
 'alex@example.com');
```

UPDATE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
UPDATE customers SET email = 'alex_new_email@example.com';

-- This statement is changed to:
UPDATE customers SET email = 'alex_new_email@example.com' WHERE customer_id =
 '123'::int;
```

DELETE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
DELETE FROM customers;

-- This statement is changed to:
DELETE FROM customers WHERE customer_id = '123'::int;
```

조인

활성 샤드 키를 사용하여 테이블에 대한 조인 작업을 수행할 때 샤드 키 조건자는 조인에 관련된 모든 테이블에 자동으로 추가됩니다. 샤드 키 조건자의 이 자동 추가는 쿼리의 모든 테이블이 동일한 공동 배치 그룹에 속하는 경우에만 발생합니다. 쿼리에 서로 다른 공동 배치 그룹의 테이블이 포함된 경우 오류가 발생합니다.

`customers` 테이블과 공동 배치된 `orders` 및 `order_details` 테이블도 있다고 가정합니다.

```
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_collocate_with='customers';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE orders (id int , customer_id int, total_amount int, date date);
CREATE TABLE order_details (id int , order_id int, customer_id int, product_name
 VARCHAR(100), price int);
COMMIT;
```

고객 ID가 10인 고객의 마지막 주문 인보이스 10개를 검색합니다.

```
SET rds_aurora.limitless_active_shard_key = '{"10"}';
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    order_details.order_id = orders.order_id AND
    customers.customer_id = 10
```

```
    order by order_date limit 10;
```

이 쿼리는 다음과 같이 변환됩니다.

```
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    orders.order_id = order_details.order_id AND
    customers.customer_id = 10 AND
    order_details.customer_id = 10 AND
    orders.customer_id = 10 AND
    ORDER BY "order_date" LIMIT 10;
```

활성 샤드 키 호환 테이블

샤드 키 조건자는 활성 샤드 키와 호환되는 테이블에만 추가됩니다. 테이블의 샤드 키에 `rds_aurora.limitless_active_shard_key` 변수에 지정된 것과 동일한 수의 열이 있는 경우 호환 가능한 것으로 간주됩니다. 쿼리에 활성 샤드 키와 호환되지 않는 테이블이 포함된 경우 쿼리를 진행하는 대신 오류를 발생시킵니다.

예시:

```
-- Compatible table
SET rds_aurora.limitless_active_shard_key = '{"10"}';

-- The following query works because the customers table is sharded on one column.
SELECT * FROM customers;

-- Incompatible table
SET rds_aurora.limitless_active_shard_key = '{"10","20"}';

-- The following query raises a error because the customers table isn't sharded on
 two columns.
 SELECT * FROM customers;
```

Aurora PostgreSQL Limitless Database의 분산 쿼리

분산 쿼리는 라우터와 둘 이상의 샤드에서 실행됩니다. 쿼리는 라우터 중 하나에서 수신합니다. 라우터는 분산 트랜잭션을 만들고 관리하며, 이는 참여 샤드로 전송됩니다. 샤드는 라우터에서 제공하는 컨텍스트로 로컬 트랜잭션을 만들고 쿼리가 실행됩니다.

트랜잭션이 커밋되면 라우터는 필요한 경우 최적화된 두 단계 커밋 프로토콜과 시간 기반 다중 버전 동시성 제어(MVCC)를 사용하여 분산 데이터베이스 시스템에서 ACID 의미 체계를 제공합니다.

시간 기반 MVCC는 각 트랜잭션의 커밋 시간을 기록하고 트랜잭션 시작 시간을 사용하여 데이터 스냅샷 시간을 생성합니다. 리더의 스냅샷이 주어졌을 때 트랜잭션이 커밋(가시적)되었는지를 식별하기 위해 데이터베이스는 커밋 시간을 스냅샷 시간과 비교합니다. 커밋 시간이 리더의 스냅샷 시간보다 전이면 보이지만 그렇지 않으면 보이지 않습니다. 이 프로토콜에서는 항상 Aurora PostgreSQL Limitless Database에서 매우 일관된 데이터를 볼 수 있습니다.

Aurora PostgreSQL Limitless Database의 PostgreSQL 로그에서 분산 쿼리 추적

분산 쿼리 추적은 Aurora PostgreSQL Limitless Database에서 PostgreSQL 로그의 쿼리를 추적하고 상호 연관시키는 도구입니다. Aurora PostgreSQL에서는 트랜잭션 ID를 사용하여 트랜잭션을 식별합니다. 그러나 Aurora PostgreSQL Limitless Database에서는 트랜잭션 ID가 다양한 라우터에서 반복될 수 있습니다. 따라서 Limitless Database에서는 대신 추적 ID를 사용하는 것이 좋습니다.

주요 사용 사례는 다음과 같습니다.

- `rds_aurora.limitless_get_last_trace_id()` 함수를 사용하여 현재 세션에서 마지막으로 실행된 쿼리의 고유한 추적 ID를 찾습니다. 그런 다음 해당 추적 ID를 사용하여 Amazon CloudWatch Logs에서 DB 클러스터 로그 그룹을 검색하여 모든 관련 로그를 찾습니다.

  `log_min_messages` 및 `log_min_error_statement` 파라미터를 함께 사용하여 인쇄된 로그 볼륨을 제어하고 추적 ID가 포함된 문을 인쇄할 수 있습니다.
- `log_min_duration_statement` 파라미터를 사용하여 모든 쿼리가 실행 기간 및 추적 ID를 인쇄하는 기준이 되는 실행 시간을 결정합니다. 그런 다음 CloudWatch Logs의 DB 클러스터 로그 그룹에서 이 실행 시간을 검색하여 병목 노드와 플래너 최적화 노력을 확인할 수 있습니다.

  `log_min_duration_statement` 파라미터는 `log_min_messages` 및 `log_min_error_statement` 파라미터의 값에 관계없이 모든 노드에 대해 추적 ID를 활성화합니다.

주제

- 추적 ID
- 쿼리 추적 사용
- 로그 예제

추적 ID

이 기능의 핵심은 추적 ID라고 하는 고유 식별자입니다. 추적 ID는 PostgreSQL 로그의 STATEMENT 로그 줄에 추가된 31자리 문자열로, 특정 쿼리 실행과 관련된 로그를 상호 연관시키기 위한 정확한 식별자 역할을 합니다. 대표적인 예는 `1126253375719408504000000000011` 및 `1126253375719408495000000000090`입니다.

추적 ID는 다음으로 구성됩니다.

- 트랜잭션 식별자 - 트랜잭션을 고유하게 식별하는 처음 20자리입니다.
- 명령 식별자 - 트랜잭션 내의 개별 쿼리를 나타내는 처음 30자리입니다.

  명시적 트랜잭션 블록 내에서 4,294,967,294개 이상의 쿼리가 실행되는 경우 명령 식별자는 1로 래핑됩니다. 이 경우 PostgreSQL 로그에 다음 `LOG` 메시지가 표시됩니다.

```
wrapping around the tracing ID back to 1 after running 4294967294 (4.2 billion or
 2^32-2) queries inside of an explicit transaction block
```

- 노드 유형 식별자 - 노드가 코디네이터 라우터(1) 또는 참가자 노드(`0`) 중 무엇으로 작동하는지 나타내는 마지막 숫자입니다.

다음 예시에서는 추적 ID의 구성 요소를 보여줍니다.

- `1126253375719408504000000000011`:
  - 트랜잭션 식별자 - `1126253375719408504`
  - 명령 식별자 - `112625337571940850400000000001`은 트랜잭션 블록의 첫 번째 명령을 나타냅니다.
  - 노드 유형 식별자 - `1`은 코디네이터 라우터를 나타냅니다.
- `1126253375719408495000000000090`:
  - 트랜잭션 식별자 - `1126253375719408495`
  - 명령 식별자 - `112625337571940849500000000009`는 트랜잭션 블록의 9번째 명령을 나타냅니다.
  - 노드 유형 식별자 - `0`은 참가자 노드를 나타냅니다.

쿼리 추적 사용

쿼리 추적을 사용하려면 다음 작업을 수행합니다.

1. 추적이 활성화되어 있는지 확인합니다.

   다음 명령을 사용하여 확인할 수 있습니다.

   ```
   SHOW rds_aurora.limitless_log_distributed_trace_id;
   ```

   기본적으로 활성화(on)되어 있습니다. 활성화되지 않은 경우 다음 명령을 사용하여 설정합니다.

```
SET rds_aurora.limitless_log_distributed_trace_id = on;
```

2. 로그 심각도 수준을 구성하여 인쇄된 로그 볼륨을 제어합니다.

   로그 볼륨은 `log_min_messages` 파라미터로 제어됩니다. `log_min_error_statement` 파라미터는 추적 ID가 있는 `STATEMENT` 줄을 인쇄하는 데 사용됩니다. 둘 다 기본적으로 `ERROR`로 설정됩니다. 다음 명령을 사용하여 확인할 수 있습니다.

   ```
   SHOW log_min_messages;
   SHOW log_min_error_statement;
   ```

   심각도 수준을 업데이트하고 현재 세션의 `STATEMENT` 줄을 인쇄하려면 다음 심각도 수준 중 하나와 함께 다음 명령을 사용합니다.

```
SET log_min_messages = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO | NOTICE
 | WARNING | ERROR | LOG | FATAL | PANIC';
SET log_min_error_statement = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO
 | NOTICE | WARNING | ERROR | LOG | FATAL | PANIC';
```

   예시:

   ```
   SET log_min_messages = 'WARNING';
   SET log_min_error_statement = 'WARNING';
   ```

3. 특정 실행 시간을 초과하는 로그에서 추적 ID 인쇄를 활성화합니다.

   `log_min_duration_statement` 파라미터를 최소 쿼리 실행 시간으로 변경할 수 있습니다. 이 시간을 초과하면 쿼리가 DB 클러스터 전반의 추적 ID와 함께 실행 기간이 포함된 로그 줄을 인쇄합니다. 이 파라미터는 기본적으로 `-1`로 설정되어 있으며 비활성화되어 있다는 뜻입니다. 다음 명령을 사용하여 확인할 수 있습니다.

   ```
   SHOW log_min_duration_statement;
   ```

   `0`으로 변경하면 DB 클러스터 전체의 모든 쿼리에 대한 로그에 기간 및 추적 ID를 인쇄합니다. 다음 명령을 사용하여 현재 세션에 대해 `0`으로 설정할 수 있습니다.

   ```
   SET log_min_duration_statement = 0;
   ```

4. 추적 ID를 가져옵니다.

   쿼리를 실행한 후(명시적 트랜잭션 블록 내에서도) `rds_aurora.limitless_get_last_trace_id` 함수를 직접 호출하여 마지막 쿼리 실행의 추적 ID를 가져옵니다.

```
SELECT * FROM rds_aurora.limitless_get_last_trace_id();
```

   이 함수는 트랜잭션 식별자와 명령 식별자를 반환합니다. 노드 유형 식별자는 반환되지 않습니다.

```
=> SELECT * FROM customers;
 customer_id | fname | lname
-------------+-------+-------
(0 rows)

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier |       command_identifier
------------------------+--------------------------------
 10104661421959001813   | 101046614219590018130000000001
(1 row)
```

   추적 ID가 없으므로 함수는 분산되지 않은 쿼리에 대해 빈 줄을 반환합니다.

```
=> SET search_path = public;
SET

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier | command_identifier
------------------------+--------------------
                        |
(1 row)
```

   Note

   VACUUM 및 ANALYZE 쿼리의 경우 기간 문은 추적 ID와 함께 로깅되지 않습니다. 따라서 `limitless_get_last_trace_id()`는 추적 ID를 반환하지 않습니다. VACUUM 또는 ANALYZE가 장기 실행 작업인 경우 다음 쿼리를 사용하여 해당 작업에 대한 추적 ID를 가져올 수 있습니다.

```
SELECT * FROM rds_aurora.limitless_stat_activity
```

```
WHERE distributed_tracing_id IS NOT NULL;
```

마지막 추적 ID를 찾기 전에 서버가 중지되면 PostgreSQL 로그를 수동으로 검색하여 실패 직전의 로그에서 추적 ID를 찾아야 합니다.

5. CloudWatch를 사용하여 DB 클러스터 로그에서 추적 ID를 검색합니다.

   다음 예시와 같이 CloudWatch Insights를 사용하여 DB 클러스터의 로그 그룹을 쿼리합니다.

   - 특정 트랜잭션 식별자 및 그 안에서 실행되는 모든 명령에 대한 쿼리:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813/
     | sort @timestamp desc
     ```

   - 특정 명령 식별자에 대한 쿼리:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813000000001/
     | sort @timestamp desc
     ```

6. 분산 쿼리에서 생성된 DB 클러스터의 모든 로그를 검사합니다.

로그 예제

다음 예시는 쿼리 추적 사용을 보여줍니다.

오류가 발생하기 쉬운 쿼리에 대한 로그 상호 연관

이 예시에서 TRUNCATE 명령은 해당 테이블이 없으면 customers 테이블에서 실행됩니다.

쿼리 추적 없음

코디네이터 라우터의 PostgreSQL 로그 파일:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:CONTEXT: remote
 SQL command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:STATEMENT:
 truncate customers;
```

참가자 샤드의 PostgreSQL 로그 파일:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:STATEMENT:
 truncate customers;
```

이러한 로그는 일반적인 로그입니다. DB 클러스터 전체에서 쿼리를 쉽게 상호 연관시키는 데 필요한 정확한 식별자가 부족합니다.

쿼리 추적 있음

코디네이터 라우터의 PostgreSQL 로그 파일:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:CONTEXT: remote SQL
 command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT: /* tid =
 1126253375719408502700000000011 */ truncate customers;
```

참가자 샤드의 PostgreSQL 로그 파일:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR:  failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT:  /* tid
 = 1126253375719408502700000000010 */ truncate customers;
```

쿼리 추적이 있는 경우 각 로그 라인에 31자리 고유 식별자가 추가됩니다. 여기서 `1126253375719408502700000000011` 및 `1126253375719408502700000000010`은 각각 코디네이터 및 참가자 노드의 추적 ID를 나타냅니다.

- 트랜잭션 식별자 - `11262533757194085027`
- 명령 식별자 –`112625337571940850270000000001`
- 노드 유형 식별자 - 마지막 자리인 `1` 또는 `0`은 각각 코디네이터 라우터와 참가자 노드를 나타냅니다.

로그를 상호 연관시켜 다양한 노드에서 쿼리 실행 시간 찾기

이 예시에서 `log_min_duration_statement` 파라미터는 모든 쿼리의 기간을 인쇄하기 위해 `0`으로 업데이트되었습니다.

쿼리 추적 없음

```
2024-01-15 07:28:46 UTC:[local]:postgres@postgres_limitless:[178322]:LOG: duration:
 12.779 ms statement: select * from customers;
```

쿼리 추적 있음

코디네이터 라우터의 PostgreSQL 로그 파일:

```
2024-01-15 07:32:08 UTC:[local]:postgres@postgres_limitless:[183877]:LOG: duration:
 12.618 ms statement: /* tid = 0457669566240497088400000000011 */ select * from
 customers;
```

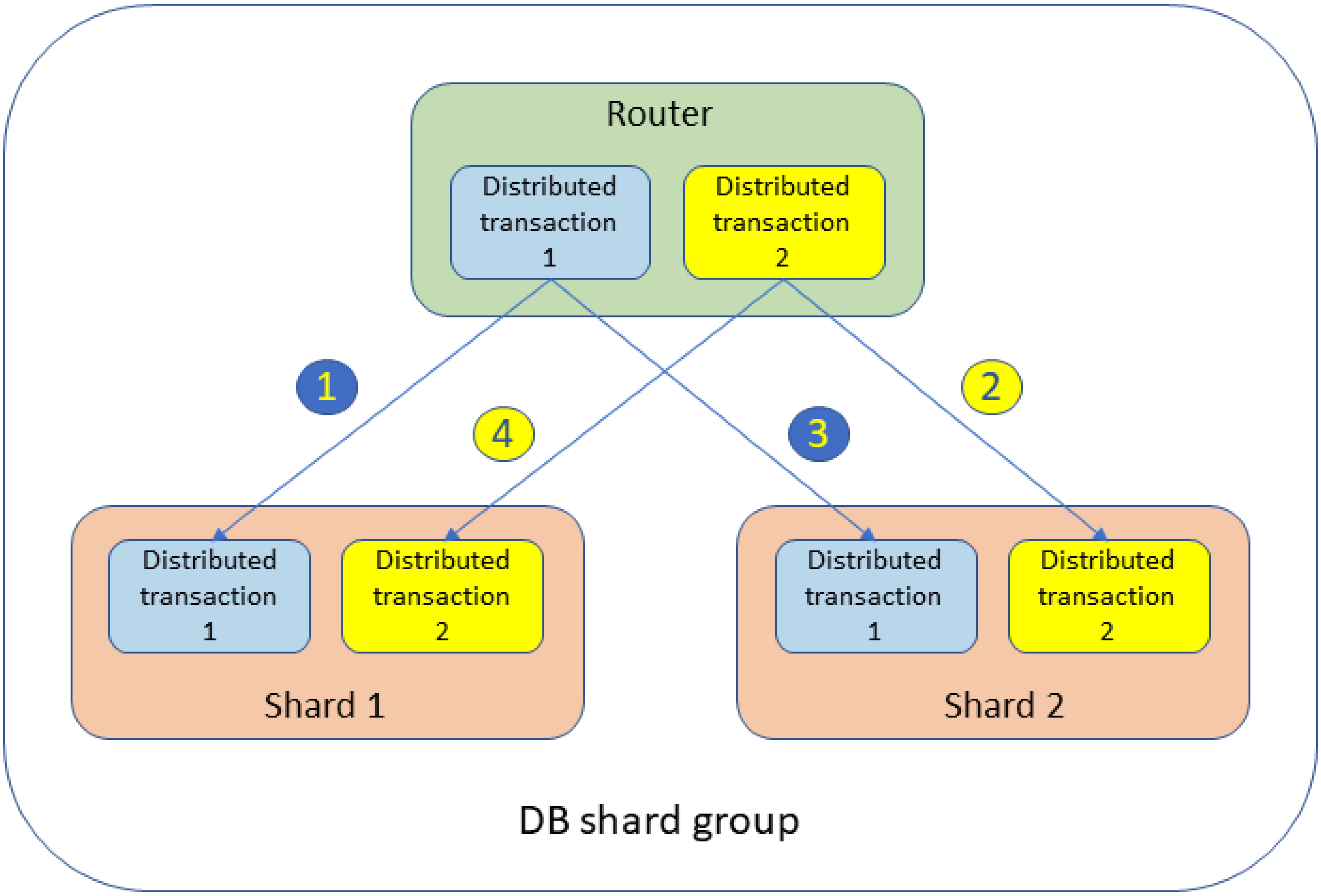
참가자 샤드의 PostgreSQL 로그 파일:

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.279 ms statement: /* tid = 0457669566240497088400000000010 */ START
 TRANSACTION ISOLATION LEVEL READ COMMITTED
```

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.249 ms parse <unnamed>: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.398 ms bind <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.019 ms execute <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.073 ms statement: /* tid = 0457669566240497088400000000010 */ COMMIT
 TRANSACTION
```

Aurora PostgreSQL Limitless Database의 분산 교착

DB 샤드 그룹에서는 서로 다른 라우터와 샤드에 분산된 트랜잭션 간에 교착이 발생할 수 있습니다. 예를 들어 다음 그림과 같이 샤드 두 개에 걸쳐 있는 동시 분산 트랜잭션 두 개가 실행됩니다.



트랜잭션은 테이블을 잠그고 다음과 같이 두 샤드에서 대기 이벤트를 만듭니다.

1. 분산 트랜잭션 1:

```
UPDATE table SET value = 1 WHERE key = 'shard1_key';
```

   샤드 1에 잠금을 보유합니다.

2. 분산 트랜잭션 2:

```
UPDATE table SET value = 2 WHERE key = 'shard2_key';
```

샤드 2에 잠금을 보유합니다.

3. 분산 트랜잭션 1:

```
UPDATE table SET value = 3 WHERE key = 'shard2_key';
```

분산 트랜잭션 1이 샤드 2를 기다리고 있습니다.

4. 분산 트랜잭션 2:

```
UPDATE table SET value = 4 WHERE key = 'shard1_key';
```

분산 트랜잭션 2가 샤드 1을 기다리고 있습니다.

이 시나리오에서는 샤드 1과 샤드 2 모두 문제를 확인하지 못합니다. 트랜잭션 1은 샤드 2에서 트랜잭션 2를 기다리고 있고 트랜잭션 2는 샤드 1에서 트랜잭션 1을 기다리고 있습니다. 전역 뷰에서 트랜잭션 1은 트랜잭션 2를 기다리고 있고 트랜잭션 2는 트랜잭션 1을 기다리고 있습니다. 두 개의 서로 다른 샤드에 있는 두 트랜잭션이 서로를 기다리고 있는 이 상황을 분산 교착이라고 합니다.

Aurora PostgreSQL Limitless Database는 분산 교착을 자동으로 감지하고 해결할 수 있습니다. DB 샤드 그룹의 라우터는 트랜잭션이 리소스를 획득하기 위해 너무 오래 대기하면 알림을 받습니다. 알림을 수신하는 라우터는 DB 샤드 그룹 내의 모든 라우터 및 샤드에서 필요한 정보를 수집하기 시작합니다. 그런 다음 DB 샤드 그룹의 나머지 트랜잭션이 서로 차단되지 않고 진행될 수 있을 때까지 라우터는 분산 교착에 참여하는 트랜잭션을 종료합니다.

트랜잭션이 분산 교착의 일부이고 라우터에 의해 종료되면 다음 오류가 발생합니다.

```
ERROR: aborting transaction participating in a distributed deadlock
```

`rds_aurora.limitless_distributed_deadlock_timeout` DB 클러스터 파라미터는 라우터에 분산 교착을 확인하도록 알리기 전에 각 트랜잭션이 리소스를 기다리는 시간을 설정합니다. 워크로드에서 교착이 덜 일어나면 파라미터 값을 늘릴 수 있습니다. 기본값은 `1000`밀리초(1초)입니다.

교차 노드 교착이 발견되고 해결되면 분산 교착 주기가 PostgreSQL 로그에 게시됩니다. 교착의 일부인 각 프로세스에 대한 정보는 다음과 같습니다.

- 트랜잭션을 시작한 코디네이터 노드
- 코디네이터 노드에 있는 트랜잭션의 *backend_id*/*backend_local_xid* 형식 가상 트랜잭션 ID(xid)

- 트랜잭션의 분산 세션 ID

Aurora PostgreSQL Limitless Database 관리

다음 주제에서는 Aurora PostgreSQL Limitless Database DB 클러스터를 관리하는 방법을 설명합니다.

주제

데이터베이스 및 테이블 크기 고려 사항

Aurora PostgreSQL Limitless Database의 경우 각 샤드에서 샤딩된 테이블은 DB 샤드 그룹에서 사용할 수 있는 샤드 수에 따라 달라지는 여러 테이블 조각으로 나뉩니다. 각 테이블 조각은 최대 32TiB까지 증가할 수 있지만 각 샤드의 최대 용량은 128TiB입니다. 참조 테이블의 크기 한도는 전체 DB 샤드 그룹에 대해 32TiB입니다.

> **Note**
> 각 노드(라우터 또는 샤드)의 최대 용량은 Aurora PostgreSQL DB 클러스터의 최대 용량이므로 128TiB입니다.

Aurora PostgreSQL 및 Aurora PostgreSQL Limitless Database의 데이터베이스당 최대 관계 수(테이블, 뷰 및 인덱스 포함)는 1,431,650,303개입니다.

자세한 내용은 PostgreSQL 설명서의 Appendix K. PostgreSQL limits 및 Amazon Aurora 크기 제한 섹션을 참조하세요.

Vacuum 작업을 통한 스토리지 공간 회수

PostgreSQL Multiversion Concurrency Control(MVCC)은 트랜잭션이 커밋되거나 롤백될 때까지 업데이트되거나 삭제된 행의 내부 복사본을 저장하여 데이터 무결성을 유지하는 데 도움이 됩니다. 튜플이라고도 하는 이러한 복사본은 정기적으로 정리하지 않으면 테이블 팽창을 야기할 수 있습니다. PostgreSQL 인스턴스는 트랜잭션 ID를 기준으로 트랜잭션 순서를 지정하며, PostgreSQL은 트랜잭션 ID 기반 MVCC를 사용하여 튜플 가시성을 제어하고 트랜잭션 격리를 제공합니다. 각 트랜잭션은 데이터의 스냅샷을 설정하고 각 튜플에는 버전이 있습니다. 스냅샷과 버전 모두 트랜잭션 ID 기반입니다.

`VACUUM` 유틸리티는 데이터를 정리하기 위해 PostgreSQL에서 네 가지 주요 함수를 수행합니다.

- `VACUUM` - 만료된 행 버전을 제거하여 공간을 재사용할 수 있게 합니다.
- `VACUUM FULL` - 데드 열 버전을 제거하고 테이블을 압축하여 완전한 조각 모음을 제공함으로써 크기를 줄이고 효율성을 높입니다.
- `VACUUM FREEZE` - 이전 행 버전을 고정으로 표시하여 트랜잭션 ID 랩어라운드 문제를 방지합니다.
- `VACUUM ANALYZE` - 데드 행 버전을 제거하고 데이터베이스의 쿼리 계획 통계를 업데이트합니다. `VACUUM` 및 `ANALYZE` 함수의 조합입니다. Aurora PostgreSQL Limitless Database에서 `ANALYZE`가 작동하는 방법에 대한 자세한 내용은 ANALYZE 섹션을 참조하세요.

MVCC와 마찬가지로 Aurora PostgreSQL에서 vacuum 작업은 트랜잭션 ID 기반입니다. Vacuum이 시작될 때 진행 중인 트랜잭션이 있는 경우 해당 트랜잭션에 계속 표시되는 행은 제거되지 않습니다.

`VACUUM` 유틸리티에 대한 자세한 내용은 PostgreSQL 설명서의 VACUUM을 참조하세요. Aurora PostgreSQL Limitless Database의 `VACUUM` 지원에 대한 자세한 내용은 VACUUM 섹션을 참조하세요.

주제

AUTOVACUUM

Aurora PostgreSQL은 `VACUUM` 및 `AUTOVACUUM` 유틸리티를 사용하여 불필요한 튜플을 제거합니다. `AUTOVACUUM`의 기본 메커니즘과 수동 `VACUUM`은 동일하며 유일한 차이점은 자동화입니다.

Aurora PostgreSQL 및 Aurora PostgreSQL Limitless Database의 AUTOVACUUM은 VACUUM 및 ANALYZE 유틸리티의 조합입니다. AUTOVACUUM은 데드 튜플의 백분율 및 삽입 수와 같은 사전 정의된 규칙에 따라 정리할 데이터베이스 및 테이블을 결정합니다.

예를 들어, AUTOVACUUM은 '다시 시작'을 주기적으로 실행하여 정리를 수행합니다. 간격은 autovacuum_naptime 파라미터로 제어됩니다. 기본값은 1분입니다. AUTOVACUUM 및 VACUUM 구성 파라미터의 기본값은 Aurora PostgreSQL Limitless Database와 Aurora PostgreSQL이 동일합니다.

AUTOVACUUM 대몬은 활성화된 경우 테이블의 내용이 충분히 변경될 때마다 ANALYZE 명령을 자동으로 실행합니다. Aurora PostgreSQL Limitless Database에서 AUTOVACUUM은 라우터와 샤드 모두에 ANALYZE를 발급합니다.

AUTOVACUUM과 연결된 AUTOVACUUM 대몬 및 테이블 스토리지 파라미터에 대한 자세한 내용은 PostgreSQL 설명서의 The autovacuum daemon 및 Storage parameters를 참조하세요.

Aurora PostgreSQL Limitless Database의 시간 기반 vacuum 처리

Aurora PostgreSQL Limitless Database는 분산 시스템이므로 여러 인스턴스가 하나의 트랜잭션에 관여할 수 있습니다. 따라서 트랜잭션 ID 기반 가시성은 적용되지 않습니다. 대신 Aurora PostgreSQL Limitless Database는 시간 기반 가시성을 사용합니다. 트랜잭션 IDs는 인스턴스 간에 '통합'되지 않지만 인스턴스 간에 '통합'될 수 있기 때문입니다. 각 트랜잭션 스냅샷과 각 튜플 버전은 트랜잭션 ID 대신 시간을 따릅니다. 보다 구체적으로 설명하자면 트랜잭션 스냅샷에는 스냅샷 시작 시간이 있고, 튜플에는 만든 시간(INSERT 또는 UPDATE가 발생한 시점)과 삭제 시간(DELETE가 발생한 시점)이 있습니다.

DB 샤드 그룹의 인스턴스 간에 데이터 일관성을 유지하려면 Aurora PostgreSQL Limitless Database는 vacuum 작업이 DB 샤드 그룹의 활성 트랜잭션에 여전히 표시되는 튜플을 제거하지 않도록 해야 합니다. 따라서 Aurora PostgreSQL Limitless Database의 vacuum도 시간 기반입니다. 특정 테이블에서 VACUUM을 실행하는 것을 포함하여 VACUUM의 다른 측면은 동일하게 유지되며 사용자는 해당 테이블에 액세스할 수 있어야 합니다.

> **Note**
>
> 트랜잭션을 장기간 열어 두지 않는 것이 좋습니다.
> 시간 기반 vacuum은 트랜잭션 ID 기반 vacuum보다 더 많은 메모리를 소비합니다.

다음 예시에서는 시간 기반 vacuum의 작동 방식을 보여줍니다.

1. 고객 테이블은 네 개의 샤드에 분산됩니다.

2. 트랜잭션 1은 반복 가능한 읽기로 시작하며 샤드(샤드 1) 하나만 대상으로 합니다. 이 트랜잭션은 열린 상태로 유지됩니다.

   트랜잭션 1은 그 이후에 시작된 다른 트랜잭션보다 연령이 더 많습니다.
3. 트랜잭션 2는 나중에 시작되고 테이블에서 모든 튜플을 삭제한 다음 커밋합니다.
4. AUTOVACUUM 또는 수동 VACUUM이 데드 튜플을 정리하려고 하면(트랜잭션 2로 인해 데드 상태) 아무것도 제거되지 않습니다.

   트랜잭션 1이 여전히 이러한 튜플에 액세스해야 할 수 있기 때문에 샤드 1뿐만 아니라 샤드 2~4에 대해서도 마찬가지입니다. MVCC로 인해 트랜잭션 1에도 계속 표시됩니다.

마지막 단계는 동기화를 통해 이루어지므로 트랜잭션 1이 모든 샤드를 터치하지 않더라도 모든 샤드가 트랜잭션 1을 인식합니다.

데이터베이스 통계를 사용하여 vacuum 처리

정리해야 할 수 있는 튜플에 대한 정보를 가져오려면 pg_stat_all_tables과 유사하게 작동하는 limitless_stat_all_tables 뷰를 사용합니다. 다음 예시는 뷰를 쿼리합니다.

```
SELECT * FROM rds_aurora.limitless_stat_all_tables WHERE relname LIKE '%customer%';
```

마찬가지로 데이터베이스 통계의 경우 pg_stat_database 대신 limitless_stat_database를 사용하고 pg_stat_activity 대신 limitless_stat_activity를 사용합니다.

테이블 디스크 사용량을 확인하려면 pg_relation_size와 유사하게 작동하는 limitless_stat_relation_sizes 함수를 사용합니다. 다음 예시는 함수를 쿼리합니다.

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customer');
```

Aurora PostgreSQL Limitless Database에서 VACUUM 작업의 진행 상황을 추적하려면 pg_stat_progress_vacuum 대신 limitless_stat_progress_vacuum 뷰를 사용합니다. 다음 예시는 뷰를 쿼리합니다.

```
SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;
```

자세한 내용은 Aurora PostgreSQL Limitless Database 뷰 및 Aurora PostgreSQL Limitless Database 함수 단원을 참조하세요.

Aurora PostgreSQL과 Aurora PostgreSQL Limitless Database의 vacuum 동작 차이점

Vacuum 작동 방식과 관련하여 Aurora PostgreSQL과 Aurora PostgreSQL Limitless Database의 또 다른 차이점은 다음과 같습니다.

- Aurora PostgreSQL은 가장 오래 지속되는 트랜잭션까지 트랜잭션 ID에 대한 `VACUUM` 작업을 수행합니다. 데이터베이스에 진행 중인 트랜잭션이 없는 경우 `VACUUM`은 마지막 트랜잭션까지 작업을 수행합니다.
- Aurora PostgreSQL Limitless Database는 가장 오래된 시간 스냅샷을 10초마다 동기화합니다. 따라서 `VACUUM`은 지난 10초 이내에 실행된 트랜잭션에 대해 작업을 수행하지 않을 수 있습니다.

Aurora PostgreSQL Limitless Database의 `VACUUM` 지원에 대한 자세한 내용은 Aurora PostgreSQL Limitless Database 참조의 VACUUM 섹션을 참조하세요.

Aurora PostgreSQL Limitless Database 모니터링

Amazon CloudWatch, 향상된 모니터링 및 성능 개선 도우미를 사용하여 Aurora PostgreSQL Limitless Database를 모니터링할 수 있습니다. 또한 모니터링 및 진단에 사용할 수 있는 Aurora PostgreSQL Limitless Database에 대한 새로운 통계 함수 및 뷰와 대기 이벤트도 있습니다.

주제

Amazon CloudWatch를 사용하여 Aurora PostgreSQL Limitless Database 모니터링

Aurora PostgreSQL Limitless Database에 대한 CloudWatch 지표는 다음 차원으로 보고됩니다.

- DBShardGroup
- DBShardGroupRouterAggregation
- DBShardGroupInstance
- DBClusterIdentifier

CloudWatch 지표에 대한 자세한 내용은 Amazon CloudWatch로 Amazon Aurora 지표 모니터링 섹션을 참조하세요.

DBShardGroup 지표

CloudWatch 콘솔에서 Aurora PostgreSQL Limitless Database에 대한 `DBShardGroup` 지표를 보려면 RDS를 선택한 다음 DBShardGroup을 선택합니다.

다음 CloudWatch 지표를 추적할 수 있습니다.

- `DBShardGroupACUUtilization` - Aurora 용량 단위(ACU) 사용량이 `DBShardGroupCapacity`를 `DBShardGroupMaxACU`로 나누어 계산한 백분율로 표시됩니다.
- `DBShardGroupCapacity` - DB 샤드 그룹의 라이터 인스턴스에서 사용하는 ACU의 개수입니다.
- `DBShardGroupComputeRedundancyCapacity` - DB 샤드 그룹의 대기 인스턴스에서 사용하는 ACU의 개수입니다.
- `DBShardGroupMaxACU` – DB 샤드 그룹에 대해 구성된 ACU의 최대 개수입니다.
- `DBShardGroupMinACU` - DB 샤드 그룹에 필요한 ACU의 최소 개수입니다.

`DBShardGroupIdentifier` 차원 키는 `DBShardGroup` 지표를 집계하는 데 사용할 수 있습니다.

DBShardGroupRouterAggregation 지표

CloudWatch 콘솔에서 Aurora PostgreSQL Limitless Database에 대한 `DBShardGroupRouterAggregation` 지표를 보려면 RDS를 선택한 다음 DBShardGroupRouterAggregation을 선택합니다.

다음 CloudWatch 지표를 추적할 수 있습니다.

- `CommitThroughput` - DB 샤드 그룹에 있는 모든 라우터 노드의 초당 평균 커밋 작업 수입니다.
- `DatabaseConnections` - DB 샤드 그룹의 모든 라우터 노드에 대한 모든 연결의 합계입니다.

DBShardGroupInstance 지표

DBShardGroupInstance는 각 샤드 또는 라우터 하위 클러스터 내의 개별 DB 인스턴스입니다.

CloudWatch 콘솔에서 Aurora PostgreSQL Limitless Database에 대한 `DBShardGroupInstance` 지표를 보려면 RDS를 선택한 다음 DBShardGroupInstance를 선택합니다.

다음 CloudWatch 지표를 추적할 수 있습니다.

- `ACUUtilization` - `ServerlessDatabaseCapacity` 지표를 하위 클러스터의 할당된 최대 ACU 값으로 나누어 계산된 백분율입니다.
- `AuroraReplicaLag` - 컴퓨팅 중복이 활성화된 Limitless 클러스터의 경우 이는 하위 클러스터의 기본 인스턴스에서 업데이트를 복제할 때의 지연 시간입니다.
- `AuroraReplicaLagMaximum` - 컴퓨팅 중복이 활성화된 Limitless 클러스터의 경우 이는 하위 클러스터의 기본 인스턴스에서 업데이트를 복제할 때의 최대 지연 시간입니다. 읽기 복제본이 삭제되거나 이름이 변경되면 이전 리소스가 재활용되면서 복제 지연 시간이 일시적으로 급증할 수 있습니다. 이 지표를 사용하여 리더 중 하나의 높은 복제 지연 시간으로 인해 장애 조치가 발생했는지 확인합니다.
- `AuroraReplicaLagMinimum` - 컴퓨팅 중복이 활성화된 무제한 클러스터의 경우 이는 하위 클러스터의 기본 인스턴스에서 업데이트를 복제할 때의 최소 지연 시간입니다.
- `BufferCacheHitRatio` - 인스턴스의 메모리 캐시(스토리지 볼륨이 아님)에서 제공된 데이터 및 인덱스의 백분율입니다.
- `CommitLatency` - 엔진 및 스토리지가 특정 노드(라우터 또는 샤드)에 대한 커밋 작업을 완료하는 평균 기간입니다.
- `CommitThroughput` - 초당 커밋 작업의 평균 수
- `CPUUtilization` - CPU 사용량은 하위 클러스터에 할당된 최대 ACU 값의 백분율입니다.
- `FreeableMemory` - 샤드 그룹이 최대 용량으로 조정될 때 사용 가능한 미사용 메모리의 양입니다. 이는 샤드 그룹의 할당된 ACU에 의해 결정됩니다. 현재 용량이 최대 용량 미만인 모든 ACU에 대해 이 값은 약 2GiB씩 증가합니다. 따라서 이 지표는 DB 샤드 그룹이 최대 한도까지 스케일 업될 때까지 0에 가까워지지 않습니다.
- `MaximumUsedTransactionIDs` - 가장 오랫동안 vacuum되지 않은 트랜잭션 ID의 연령(트랜잭션 수)입니다. 이 값이 2,146,483,648(2^31 - 1,000,000)에 도달하면 트랜잭션 ID의 랩어라운드를 방지

할 목적으로 데이터베이스가 읽기 전용 모드로 강제 전환됩니다. 자세한 내용은 PostgreSQL 설명서의 P트랜잭션 ID 랩어라운드 실패 방지를 참조하세요.

- `NetworkReceiveThroughput` - DB 샤드 그룹의 각 인스턴스가 클라이언트에서 수신하는 네트워크 처리량입니다. 이 처리량에서 DB 샤드 그룹의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- `NetworkThroughput` - DB 샤드 그룹의 클라이언트와 라우터, 라우터와 샤드 간에 집계된 네트워크 처리량(전송 및 수신 모두)입니다. 이 처리량에서 DB 샤드 그룹의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- `NetworkTransmitThroughput` - DB 샤드 그룹의 각 인스턴스가 클라이언트에 전송하는 네트워크 처리량입니다. 이 처리량에서 DB 샤드 그룹의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- `ReadIOPS` - 디스크 읽기 초당 입출력 작업량(IOPS)의 평균 수입니다.
- `ReadLatency` - 디스크 읽기 입출력(I/O) 작업당 평균 처리 시간입니다.
- `ReadThroughput` - 초당 디스크에서 읽은 평균 바이트 수입니다.
- `ServerlessDatabaseCapacity` - DB 샤드 그룹 내 DB 샤드 또는 라우터 하위 클러스터의 현재 용량입니다.
- `StorageNetworkReceiveThroughput` - DB 샤드 그룹의 각 인스턴스가 Aurora 스토리지 하위 시스템에서 수신하는 네트워크 처리량입니다.
- `StorageNetworkThroughput` - DB 샤드 그룹의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송하고 해당 시스템에서 수신한 네트워크 처리량의 집계입니다.
- `StorageNetworkTransmitThroughput` - DB 샤드 그룹의 각 인스턴스가 Aurora 스토리지 하위 시스템으로 전송하는 네트워크 처리량입니다.
- `SwapUsage` - DB 샤드 그룹에서 사용하는 스왑 공간의 양입니다.
- `TempStorageIOPS` - DB 인스턴스에 연결된 로컬 스토리지에서 수행된 I/O 작업의 평균 수입니다. 여기에는 읽기 및 쓰기 I/O 작업이 모두 포함됩니다.

  `TempStorageIOPS`는 `TempStorageThroughput`과 함께 사용되어 DB 인스턴스와 로컬 스토리지 디바이스 간의 전송에 대한 네트워크 활동이 예상치 못한 용량 증가의 원인이 되는 드문 경우를 진단하는 데 도움이 될 수 있습니다.
- `TempStorageThroughput` - 라우터 또는 샤드와 연결된 로컬 스토리지로 전송되고 해당 스토리지에서 전송되는 데이터의 양입니다.
- `WriteIOPS` - 디스크 쓰기 IOPS의 평균 수입니다.
- `WriteLatency` - 디스크 쓰기 I/O 작업당 평균 처리 시간입니다.

- `WriteThroughput` - 초당 디스크에 쓴 평균 바이트 수입니다.

다음 차원 키는 `DBShardGroupInstance` 지표를 집계하는 데 사용할 수 있습니다.

- `DBClusterIdentifier` - Aurora PostgreSQL DB 클러스터입니다.
- `DBShardGroupIdentifier` - 인스턴스가 속한 DB 샤드 그룹입니다.
- `DBShardGroupSubClusterType` - 노드 유형으로, `Distributed Transaction Router`(라우터) 또는 `Data Access Shard`(샤드)입니다.
- `DBShardGroupSubClusterIdentifier` - 인스턴스가 속한 라우터 또는 샤드의 이름입니다.

다음은 CloudWatch 지표 집계의 예입니다.

- DB 샤드 그룹의 특정 샤드 또는 라우터에 속하는 모든 인스턴스 총 `CPUUtilization`입니다.
- DB 샤드 그룹에 있는 모든 인스턴스의 총 `CPUUtilization`입니다.

DBClusterIdentifier 지표

CloudWatch 콘솔에서 Aurora PostgreSQL Limitless Database에 대한 `DBClusterIdentifier` 지표를 보려면 RDS를 선택한 다음 DBClusterIdentifier를 선택합니다.

Aurora PostgreSQL Limitless Database를 사용하는 경우 Aurora DB 클러스터에 비해 입출력(I/O) 작업이 더 많을 수 있습니다. Limitless Database 클러스터에 대해 다음 CloudWatch 지표를 추적할 수 있습니다.

- `VolumeReadIops` - 클러스터 볼륨에서 요금이 청구된 읽기 I/O 작업의 수로, 5분마다 보고됩니다.
- `VolumeWriteIops` - 클러스터 볼륨에 대한 평균 디스크 쓰기 I/O 작업 수로, 5분마다 보고됩니다.

Aurora PostgreSQL Limitless Database는 Aurora I/O-Optimized 클러스터 스토리지 구성을 사용합니다. Aurora I/O-Optimized를 사용하면 I/O 요청 100만 건당 요금을 지불하지 않고 모든 I/O 작업에 대해 월간 고정 요금을 지불합니다. 자세한 내용은 [Amazon Aurora DB 클러스터의 스토리지 구성](#) 섹션을 참조하세요.

Aurora DB 클러스터에 비해 더 많은 스토리지를 사용할 수도 있습니다. 스토리지에 대한 다음 CloudWatch 지표를 추적할 수 있습니다.

- `BackupRetentionPeriodStorageUsed` - Aurora PostgreSQL Limitless Database 클러스터의 청구된 총 연속 백업 스토리지 사용량입니다.

- `SnapshotStorageUsed` - Aurora PostgreSQL Limitless Database 클러스터의 청구된 총 스냅샷 스토리지 사용량입니다.
- `TotalBackupStorageBilled` - 자동 백업 보존 및 DB 클러스터 스냅샷에 대한 비용의 합계입니다.

  백업 스토리지 비용에 대한 자세한 내용은 [Amazon Aurora 백업 스토리지 사용량 파악](#) 섹션을 참조하세요.
- `VolumeBytesUsed` - Aurora PostgreSQL Limitless Database 클러스터에서 사용하는 스토리지 양으로, 5분마다 보고됩니다.

CloudWatch Database Insights를 사용하여 Aurora PostgreSQL Limitless Database 모니터링

Aurora PostgreSQL Limitless Database를 활성화하려면 Database Insights의 표준 모드가 필요합니다. 이를 사용하여 Limitless Database DB 인스턴스의 데이터베이스 로드(DB 로드)를 실시간으로 모니터링할 수 있습니다. DB 로드는 데이터베이스의 세션 활동 수준을 측정합니다. Database Insights를 사용하여 Aurora PostgreSQL Limitless Database DB 인스턴스의 성능을 대규모로 분석하고 문제를 해결할 수 있습니다.

CloudWatch Database Insights에 대한 자세한 내용은 다음을 참조하세요.

- CloudWatch Database Insights를 사용하여 Amazon Aurora 데이터베이스 모니터링
- Amazon CloudWatch 사용 설명서의 CloudWatch Database Insights
- Amazon CloudWatch 사용 설명서의 Get started with CloudWatch Database Insights
- Amazon Aurora용 Database Insights를 사용하여 느린 SQL 쿼리를 모니터링하도록 데이터베이스 구성

Database Insights의 고급 모드 또는 표준 모드를 켜는 방법에 대한 내용은 다음 항목을 참조하세요.

주제

Aurora PostgreSQL Limitless Database용 Database Insights의 고급 모드 켜기

Aurora PostgreSQL Limitless Database에서 Database Insights의 고급 모드를 켜려면 다음 절차를 따르세요.

Aurora PostgreSQL Limitless Database용 DB 클러스터를 생성할 때 Database Insights의 고급 모드 켜기

Aurora PostgreSQL Limitless Database에 대한 데이터베이스를 만들 때 Database Insights의 고급 모드를 켭니다.

Console

콘솔에서 DB 클러스터를 만들 때 Database Insights의 고급 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 데이터베이스 생성를 선택합니다.
4. Database Insights 섹션에서 고급 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.
   - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Database Insights의 고급 모드에서 보존 기간은 15~24개월이어야 합니다.
   - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
5. 데이터베이스 생성를 선택합니다.

AWS CLI

DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜려면 create-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--database-insights-mode advanced`를 사용하여 Database Insights의 고급 모드를 켭니다.
- `--engine` – DB 클러스터는 `aurora-postgresql` DB 엔진을 사용해야 합니다.
- `--engine-version` – DB 클러스터는 다음과 같은 DB 엔진 버전 중 하나를 사용해야 합니다.
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – DB 클러스터는 `aurora-iopt1` DB 클러스터 스토리지 구성을 사용해야 합니다.
- `--cluster-scalability-type` - Aurora DB 클러스터의 확장성 모드를 지정합니다. `limitless`로 설정하면 클러스터가 Aurora PostgreSQL Limitless Database로 작동합니다. `standard`(기본값)로 설정하면 클러스터가 일반 DB 인스턴스 만들기를 사용합니다.

> **Note**
>
> DB 클러스터를 만든 후에는 이 설정을 수정할 수 없습니다.

- `--master-username` - DB 클러스터의 마스터 사용자 이름입니다.
- `--master-user-password` - 마스터 사용자의 암호입니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Database Insights를 켜려면 보존 기간이 최소 465일이어야 합니다.
- `--monitoring-interval` - DB 클러스터에 대한 확장 모니터링 지표를 수집하는 시점 사이의 간격(초)입니다. 이 값은 `0`일 수 없습니다.
- `--monitoring-role-arn` - RDS에서 Amazon CloudWatch Logs로 확장 모니터링 지표를 전송할 수 있도록 하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.
- `--enable-cloudwatch-logs-exports` - `postgresql` 로그를 CloudWatch Logs로 내보내야 합니다.

다음 예시에서는 DB 클러스터를 만들 때 Database Insights의 고급 모드를 활성화합니다.

Linux, macOS, Unix의 경우:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode advanced \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 465 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Windows의 경우:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode advanced ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 465 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

DB 클러스터를 만들 때 Database Insights의 고급 모드를 켜려면 CreateDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~advanced`
- `Engine~aurora-postgresql`
- `EngineVersion`을 Limitless Database에 사용 가능한 엔진 버전으로
- `StorageType~aurora-iopt1`
- `ClusterScalabilityType~limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights~True`
- `PerformanceInsightsRetentionPeriod`를 최소 465일로
- `MonitoringInterval`을 `0`이 아닌 값으로
- `MonitoringRoleArn`을 RDS에서 Amazon CloudWatch Logs로 확장 모니터링 지표를 전송할 수 있도록 하는 IAM 역할의 Amazon 리소스 이름(ARN)으로

Aurora PostgreSQL Limitless Database용 DB 클러스터를 수정할 때 Database Insights의 고급 모드 켜기

Aurora PostgreSQL Limitless Database에 대한 데이터베이스를 수정할 때 Database Insights를 켭니다.

> **Note**
>
> Database Insights를 사용 설정하려면 DB 클러스터의 각 DB 인스턴스에 동일한 성능 개선 도우미 및 확장 모니터링 설정이 있어야 합니다.

Console

콘솔에서 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. DB 클러스터를 선택하고 수정을 선택합니다.
4. Database Insights 섹션에서 고급 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.
   - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Database Insights의 고급 모드에서 보존 기간은 15~24개월이어야 합니다.
   - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
5. Continue(계속)를 선택합니다.
6. 수정 스케줄링에 대해 즉시 적용을 선택합니다. 예약된 다음 유지 관리 기간에 적용을 선택하면 데이터베이스에서 이 설정을 무시하고 Database Insights의 고급 모드를 즉시 켭니다.
7. 클러스터 수정을 선택합니다.

AWS CLI

DB 클러스터를 수정 때 Database Insights의 고급 모드를 켜려면 modify-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--database-insights-mode advanced`를 사용하여 Database Insights의 고급 모드를 켭니다.
- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Database Insights의 고급 모드를 켜려면 보존 기간이 최소 465일이어야 합니다.

다음 예시에서는 DB 클러스터를 수정할 때 Database Insights의 고급 모드를 활성화합니다.

Linux, macOS, Unix의 경우:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

DB 클러스터를 수정할 때 Database Insights의 고급 모드를 켜려면 ModifyDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~advanced`
- `EnablePerformanceInsights~True`

- `PerformanceInsightsRetentionPeriod`를 최소 465일로

Aurora PostgreSQL Limitless Database용 Database Insights의 표준 모드 켜기

Aurora PostgreSQL Limitless Database에서 Database Insights의 표준 모드를 켜려면 다음 절차를 따르세요.

Aurora PostgreSQL Limitless Database용 DB 클러스터를 생성할 때 Database Insights의 표준 모드 켜기

Aurora PostgreSQL Limitless Database에 대한 데이터베이스를 만들 때 Database Insights의 표준 모드를 켭니다.

Console

콘솔에서 DB 클러스터를 만들 때 Database Insights의 표준 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 데이터베이스 생성를 선택합니다.
4. Database Insights 섹션에서 표준 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.
   - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Aurora PostgreSQL Limitless Database용 DB 클러스터를 만들려면 보존 기간이 31일 이상이어야 합니다.
   - AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.
5. 데이터베이스 생성를 선택합니다.

AWS CLI

DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜려면 create-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--database-insights-mode standard`를 사용하여 Database Insights의 표준 모드를 켭니다.
- `--engine` – DB 클러스터는 `aurora-postgresql` DB 엔진을 사용해야 합니다.
- `--engine-version` – DB 클러스터는 다음과 같은 DB 엔진 버전 중 하나를 사용해야 합니다.
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – DB 클러스터는 `aurora-iopt1` DB 클러스터 스토리지 구성을 사용해야 합니다.
- `--cluster-scalability-type` - Aurora DB 클러스터의 확장성 모드를 지정합니다. `limitless`로 설정하면 클러스터가 Aurora PostgreSQL Limitless Database로 작동합니다. `standard`(기본값)로 설정하면 클러스터가 일반 DB 인스턴스 만들기를 사용합니다.

> **Note**
>
> DB 클러스터를 만든 후에는 이 설정을 수정할 수 없습니다.

- `--master-username` - DB 클러스터의 마스터 사용자 이름입니다.
- `--master-user-password` - 마스터 사용자의 암호입니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Aurora PostgreSQL Limitless Database용 DB 클러스터를 만들려면 보존 기간이 31일 이상이어야 합니다.
- `--monitoring-interval` - DB 클러스터에 대한 확장 모니터링 지표를 수집하는 시점 사이의 간격(초)입니다. 이 값은 `0`일 수 없습니다.
- `--monitoring-role-arn` - RDS에서 Amazon CloudWatch Logs로 확장 모니터링 지표를 전송할 수 있도록 하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.
- `--enable-cloudwatch-logs-exports` - `postgresql` 로그를 CloudWatch Logs로 내보내야 합니다.

다음 예시에서는 DB 클러스터를 만들 때 Database Insights의 표준 모드를 활성화합니다.

Linux, macOS, Unix의 경우:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode standard \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 31 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Windows의 경우:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode standard ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 31 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

DB 클러스터를 만들 때 Database Insights의 표준 모드를 켜려면 CreateDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~standard`
- `Engine~aurora-postgresql`
- `EngineVersion`을 Limitless Database에 사용 가능한 엔진 버전으로
- `StorageType~aurora-iopt1`

- `ClusterScalabilityType~limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights~True`
- `PerformanceInsightsRetentionPeriod`를 최소 31일로
- `MonitoringInterval`을 0이 아닌 값으로
- `MonitoringRoleArn`을 RDS에서 Amazon CloudWatch Logs로 확장 모니터링 지표를 전송할 수 있도록 하는 IAM 역할의 Amazon 리소스 이름(ARN)으로

Aurora PostgreSQL Limitless Database용 DB 클러스터를 수정할 때 Database Insights의 표준 모드 켜기

Aurora PostgreSQL Limitless Database에 대한 데이터베이스를 수정할 때 Database Insights를 켭니다.

> **Note**
>
> Database Insights를 사용 설정하려면 DB 클러스터의 각 DB 인스턴스에 동일한 성능 개선 도우미 및 향상된 모니터링 설정이 있어야 합니다.

Console

콘솔에서 DB 클러스터를 만들 때 Database Insights의 표준 모드를 켤 수 있습니다. Database Insights 설정은 DB 클러스터의 모든 DB 인스턴스에 적용됩니다.

콘솔을 사용하여 DB 클러스터를 수정할 때 Database Insights의 표준 모드를 켜는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. DB 클러스터를 선택하고 수정을 선택합니다.
4. Database Insights 섹션에서 표준 모드를 선택합니다. 그런 다음, 다음 옵션 중 하나를 선택합니다.

   - 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. Aurora PostgreSQL Limitless Database용 DB 클러스터를 만들려면 보존 기간이 31일 이상이어야 합니다.

- AWS KMS key - KMS 키를 지정합니다. 성능 개선 도우미는 KMS 키를 사용하여 잠재적으로 민감한 데이터를 모두 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 Amazon Aurora 리소스 암호화 섹션을 참조하세요.

5. Continue(계속)를 선택합니다.
6. 수정 스케줄링에 대해 즉시 적용을 선택합니다. 예약된 다음 유지 관리 기간에 적용을 선택하면 데이터베이스에서 이 설정을 무시하고 Database Insights의 표준 모드를 즉시 켭니다.
7. 클러스터 수정을 선택합니다.

AWS CLI

DB 클러스터를 수정 때 Database Insights의 표준 모드를 켜려면 modify-db-cluster AWS CLI 명령을 직접 호출하고 다음 값을 제공합니다.

- `--db-cluster-identifier` - DB 클러스터의 식별자입니다.
- `--database-insights-mode standard`를 사용하여 Database Insights의 표준 모드를 켭니다.
- `--enable-performance-insights`를 사용하여 Database Insights에 대한 성능 개선 도우미를 켭니다.
- `--performance-insights-retention-period` - DB 클러스터의 데이터 보존 기간입니다. Database Insights의 표준 모드를 켜려면 보존 기간이 최소 31일이어야 합니다.

다음 예시에서는 DB 클러스터를 수정할 때 Database Insights의 표준 모드를 활성화합니다.

Linux, macOS, Unix의 경우:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 31
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
```

```
    --performance-insights-retention-period 31
```

RDS API

DB 클러스터를 수정할 때 Database Insights의 표준 모드를 켜려면 ModifyDBCluster Amazon RDS API 작업에 대해 다음 파라미터를 지정합니다.

- `DatabaseInsightsMode~standard`
- `EnablePerformanceInsights~True`
- `PerformanceInsightsRetentionPeriod`를 최소 31일로

Amazon CloudWatch Logs를 사용하여 Aurora PostgreSQL Limitless Database 모니터링

Aurora PostgreSQL Limitless Database를 활성화하려면 PostgreSQL 로그를 CloudWatch Logs로 내보내야 합니다. 표준 Aurora PostgreSQL DB 클러스터에 대한 PostgreSQL 로그에 액세스하는 것과 마찬가지로 CloudWatch Logs Insights에서 이러한 로그에 액세스하고 분석할 수 있습니다. 자세한 내용은 CloudWatch Logs Insights를 사용하여 PostgreSQL 로그 분석 단원을 참조하십시오.

DB 클러스터의 로그 그룹 이름은 Aurora PostgreSQL과 동일합니다.

```
/aws/rds/cluster/DB_cluster_ID/postgresql
```

DB 샤드 그룹의 로그 그룹 이름 형식은 다음과 같습니다.

```
/aws/rds/cluster/DB_cluster_ID/DB_shard_group_ID/postgresql
```

각 노드(라우터 또는 샤드)에 대한 로그 스트림이 있습니다. 이름의 형식은 다음과 같습니다.

```
[DistributedTransactionRouter|
DataAccessShard]/node_cluster_serial_ID-node_instance_serial_ID/n
```

예시:

- 라우터 - `DistributedTransactionRouter/6-6.2`
- 샤드 - `DataAccessShard/22-22.0`

> **Note**
>
> DB 클러스터의 경우처럼 RDS 콘솔, AWS CLI 또는 RDS API에서 직접 DB 샤드 그룹의 PostgreSQL 로그 파일을 볼 수 없습니다. CloudWatch Logs Insights를 사용하여 확인해야 합니다.

향상된 모니터링을 사용한 Aurora PostgreSQL Limitless Database 모니터링

향상된 모니터링은 Aurora PostgreSQL Limitless Database 활성화할 경우 필수입니다. 이를 사용하여 Limitless Database DB 인스턴스의 운영 체제를 실시간으로 모니터링할 수 있습니다.

Aurora는 CloudWatch Logs에 향상된 모니터링 지표를 게시합니다. 사용 가능한 주요 지표로는 데이터베이스 연결, 스토리지 사용량, 쿼리 지연 시간 등이 있습니다. 이러한 지표는 성능 병목 현상을 파악하는 데 도움이 됩니다.

향상된 모니터링 지표에 대한 자세한 내용은 Aurora의 지표 섹션을 참조하세요.

성능 개선 도우미를 사용한 Aurora PostgreSQL Limitless Database 모니터링

성능 개선 도우미를 사용하여 Aurora PostgreSQL Limitless Database 클러스터를 모니터링합니다. 성능 개선 도우미는 Aurora PostgreSQL Limitless Database에서도 표준 Aurora DB 클러스터에서와 마찬가지로 작동합니다. 그러나 Aurora PostgreSQL Limitless Database에서는 샤드 그룹 수준에서 지표를 추적합니다.

추적할 두 가지 주요 성능 개선 도우미 지표는 다음과 같습니다.

- 데이터베이스 로드 - 데이터베이스의 활동 수준을 측정합니다. 성능 개선 도우미의 핵심 지표는 `DBLoad`이며, 1초 간격으로 수집됩니다.

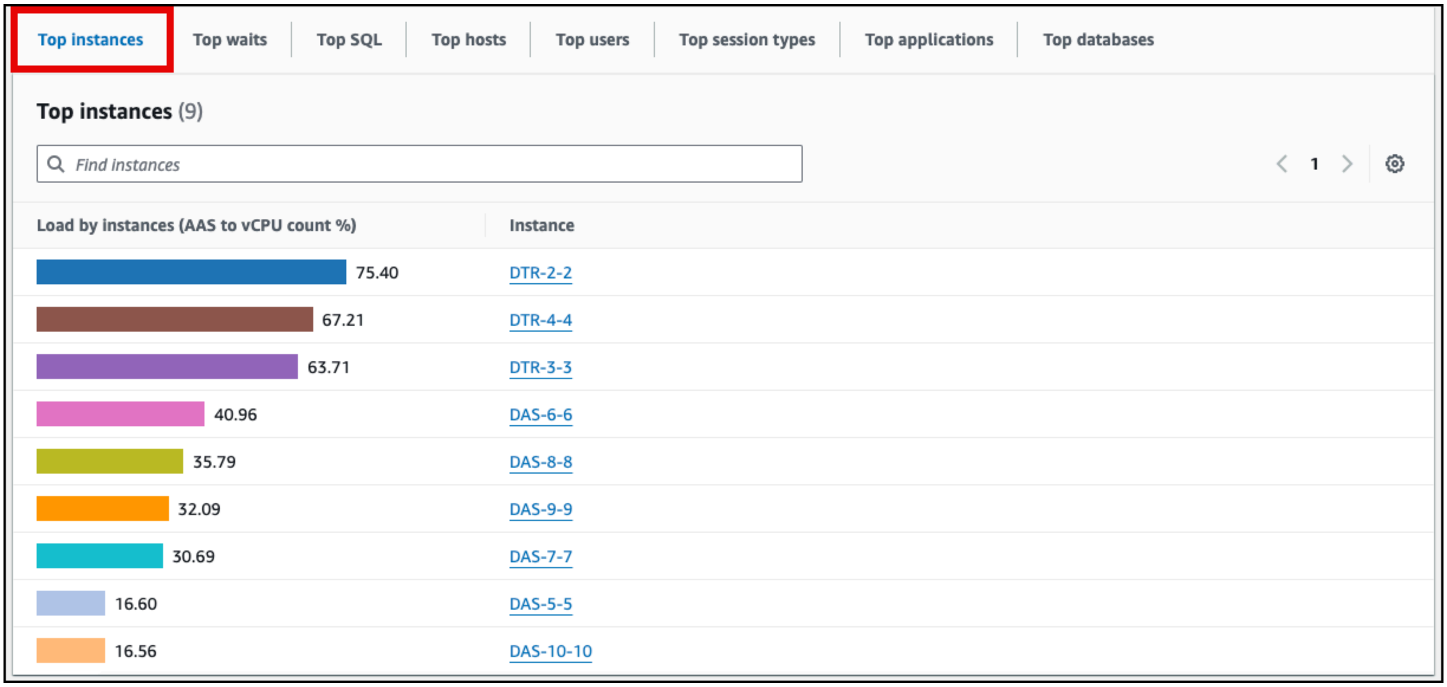
  성능 개선 도우미에서 `DBLoad` 지표의 단위는 평균 활성 세션(AAS)입니다. 평균 활성 세션을 구하기 위해 성능 개선 도우미는 쿼리를 동시에 실행하는 세션 수를 샘플링합니다. AAS는 특정 기간 동안의 총 세션 수를 총 샘플 수로 나눈 값입니다. `DBLoad` 및 AAS에 대한 자세한 내용은 데이터베이스 부하 섹션을 참조하세요.
- 최대 CPU - 데이터베이스에서 사용할 수 있는 최대 컴퓨팅 성능입니다. 활성 세션이 최대 CPU를 초과하는지 확인하려면 `Max vCPU` 줄과의 관계를 확인합니다. `Max vCPU` 값은 DB 인스턴스에서 vCPU(가상 CPU) 코어의 수로 결정됩니다. `Max vCPU`에 대한 자세한 내용은 최대 CPU 섹션을 참조하세요.

또한 `DBLoad` 지표를 이 지표의 하위 범주인 차원으로 '분할'할 수 있습니다. 가장 유용한 차원은 다음과 같습니다.

- 상위 인스턴스 - 인스턴스(샤드 및 라우터)의 상대적 DB 로드를 내림차순으로 표시합니다.
- 대기 이벤트 - SQL 문이 계속 실행되기 전에 특정 이벤트가 발생할 때까지 기다리도록 합니다. 대기 이벤트는 작업이 방해받는 위치를 나타냅니다.
- 상위 SQL - DB 로드에 가장 많이 기여하는 쿼리를 표시합니다.

성능 개선 도우미에 대한 자세한 내용은 차원 섹션을 참조하세요.

다음 그림은 DB 샤드 그룹의 상위 인스턴스 차원을 보여줍니다.



주제

- 성능 개선 도우미 대시보드를 사용하여 Aurora PostgreSQL Limitless Database의 DB 로드 분석

성능 개선 도우미 대시보드를 사용하여 Aurora PostgreSQL Limitless Database의 DB 로드 분석

성능 개선 도우미를 사용하면 Aurora PostgreSQL Limitless Database의 샤드 그룹 수준 및 인스턴스 수준에서 지표를 추적할 수 있습니다. Aurora PostgreSQL Limitless Database의 DB 로드를 분석할 때 각 샤드 및 라우터의 DB 로드를 최대 vCPU와 비교해볼 수 있습니다.

> **Note**
>
> Aurora PostgreSQL Limitless Database에는 항상 성능 개선 도우미 및 향상된 모니터링이 활성화되어 있습니다. Limitless Database에 대한 성능 개선 도우미 데이터의 최소 보존 기간은 31일(1개월)입니다.

절대 뷰에는 평균 활성 세션(AAS) 수와 예상 vCPU가 표시됩니다. 상대 뷰는 AAS 대 추정 vCPU의 비율을 보여줍니다.

주제

성능 개선 도우미 대시보드를 사용하여 Aurora PostgreSQL Limitless Database의 상대 DB 로드 분석

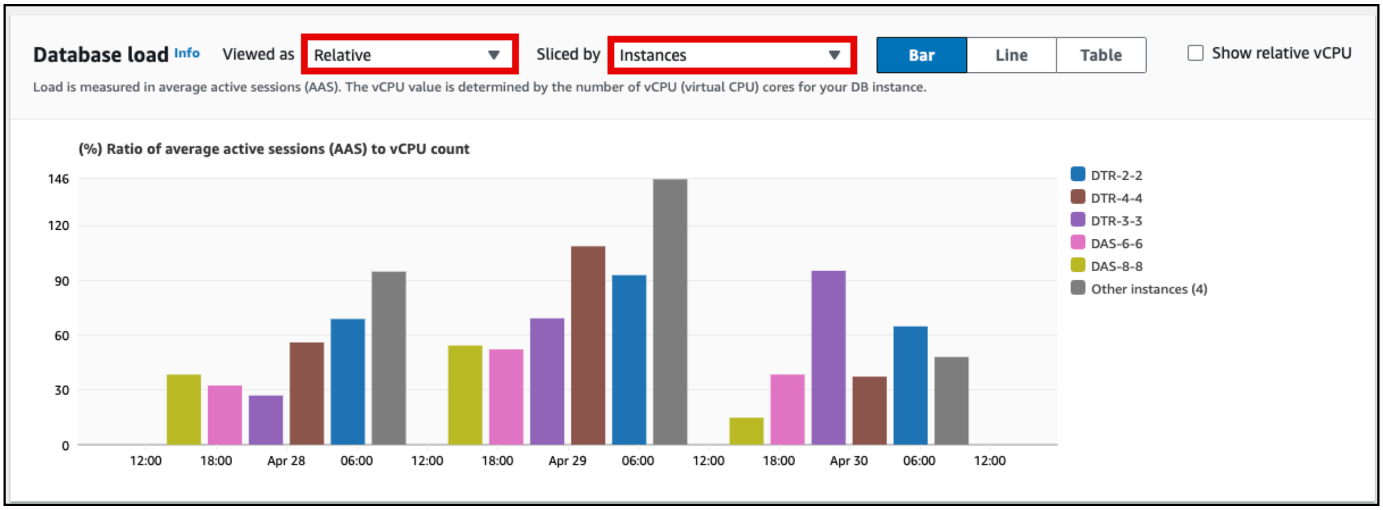
상대 DB 로드를 추적하여 Aurora PostgreSQL Limitless Database의 성능을 개선하고 싶을 수 있습니다. 인스턴스별로 Aurora PostgreSQL Limitless Database의 상대 DB 로드를 분석하려면 다음 프로시저를 사용합니다.

콘솔을 사용하여 Aurora PostgreSQL Limitless Database의 상대 DB 로드를 분석하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. Aurora PostgreSQL Limitless Database를 선택합니다. 해당 Aurora PostgreSQL Limitless Database에 대한 성능 개선 도우미 대시보드가 표시됩니다.

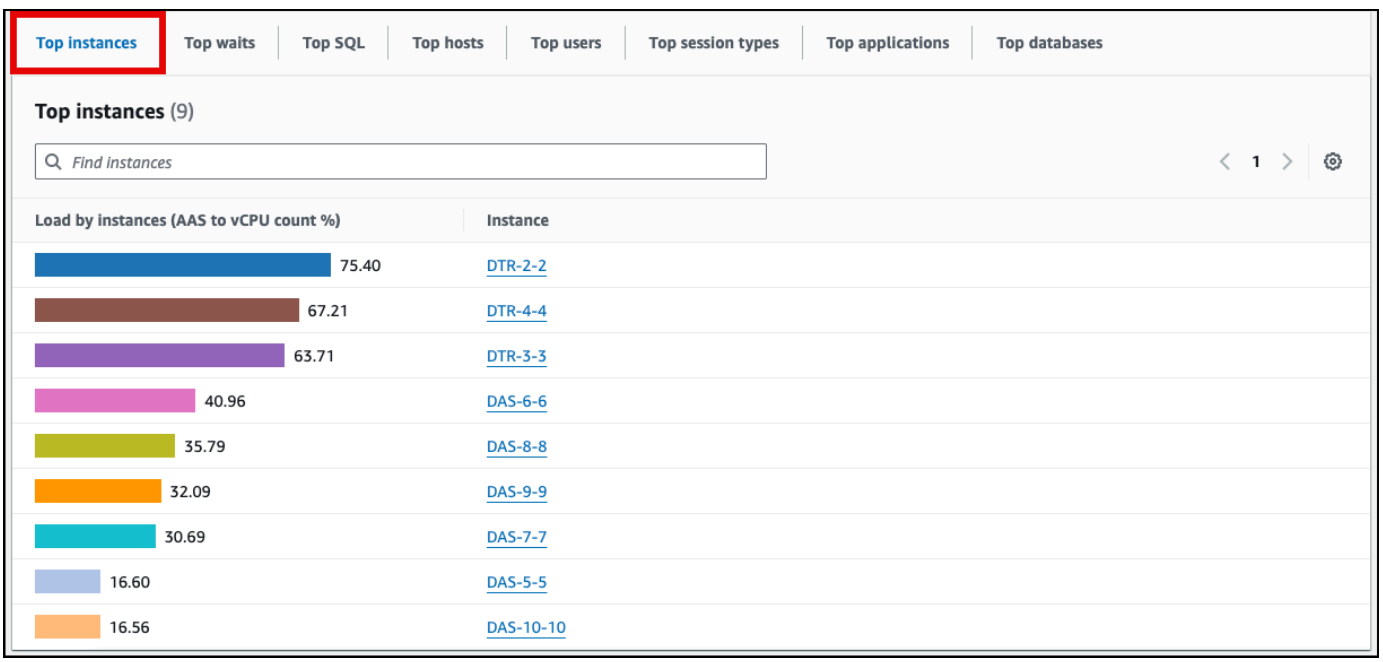
4. 데이터베이스 로드(DB 로드) 섹션에서 분할 기준에서 인스턴스를 선택합니다. Aurora PostgreSQL Limitless Database의 모든 인스턴스에 대해 평균 활성 세션(AAS) 대 vCPU 코어의 비율을 보려면 다음으로 보기에서 상대를 선택합니다.

   평균 활성 세션 차트는 Aurora PostgreSQL Limitless Database의 인스턴스에 대한 DB 로드를 보여줍니다.



5. 최상위 인스턴스를 보려면 상위 인스턴스 탭을 선택합니다.

   다음 예시에서 DB 로드가 가장 높은 인스턴스는 DTR-2-2입니다.



6. (선택 사항) Aurora PostgreSQL Limitless Database의 인스턴스에 대한 DB 로드를 분석하려면 인스턴스 열에서 인스턴스 이름을 선택합니다. DTR-2-2의 DB 로드를 보려면 인스턴스 열에서 DTR-2-2를 선택합니다.

> **Note**
>
> Aurora PostgreSQL Limitless Database의 인스턴스에 대해서만 성능 개선 도우미 지표를 볼 수 있습니다.

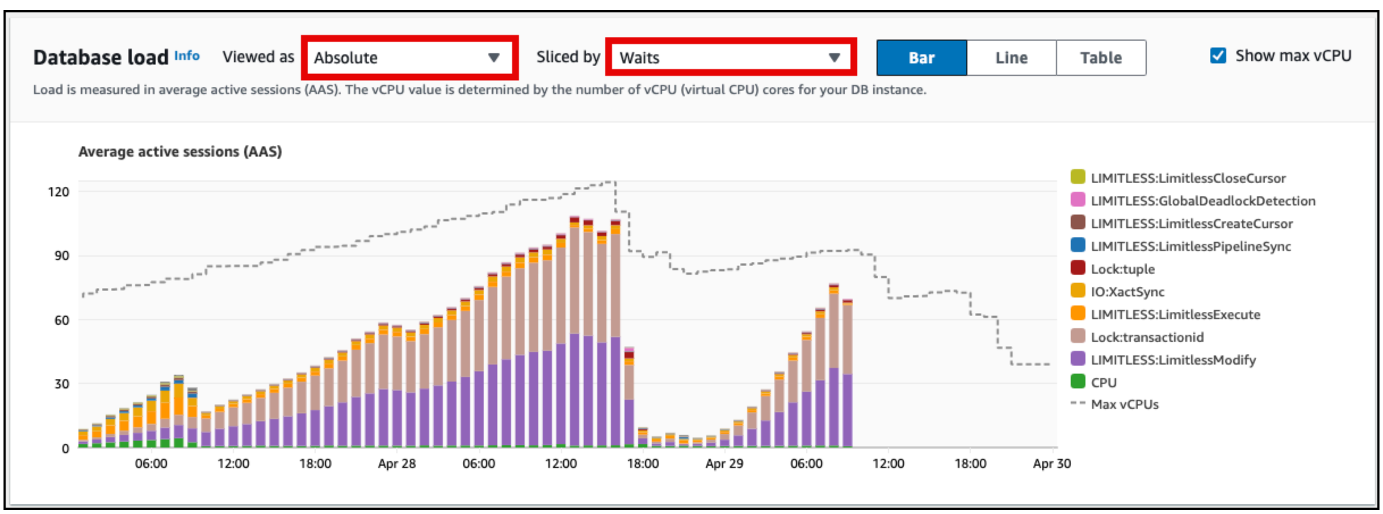
성능 개선 도우미 대시보드를 사용하여 대기를 기준으로 Aurora PostgreSQL Limitless Database의 DB 로드 분석

대기 이벤트를 추적하여 Aurora PostgreSQL Limitless Database의 성능을 개선하고 싶을 수 있습니다. 대기 이벤트별로 Aurora PostgreSQL Limitless Database의 DB 로드를 분석하려면 다음 프로시저를 사용합니다.

콘솔을 사용하여 대기를 기준으로 Aurora PostgreSQL Limitless Database의 DB 로드를 분석하는 방법

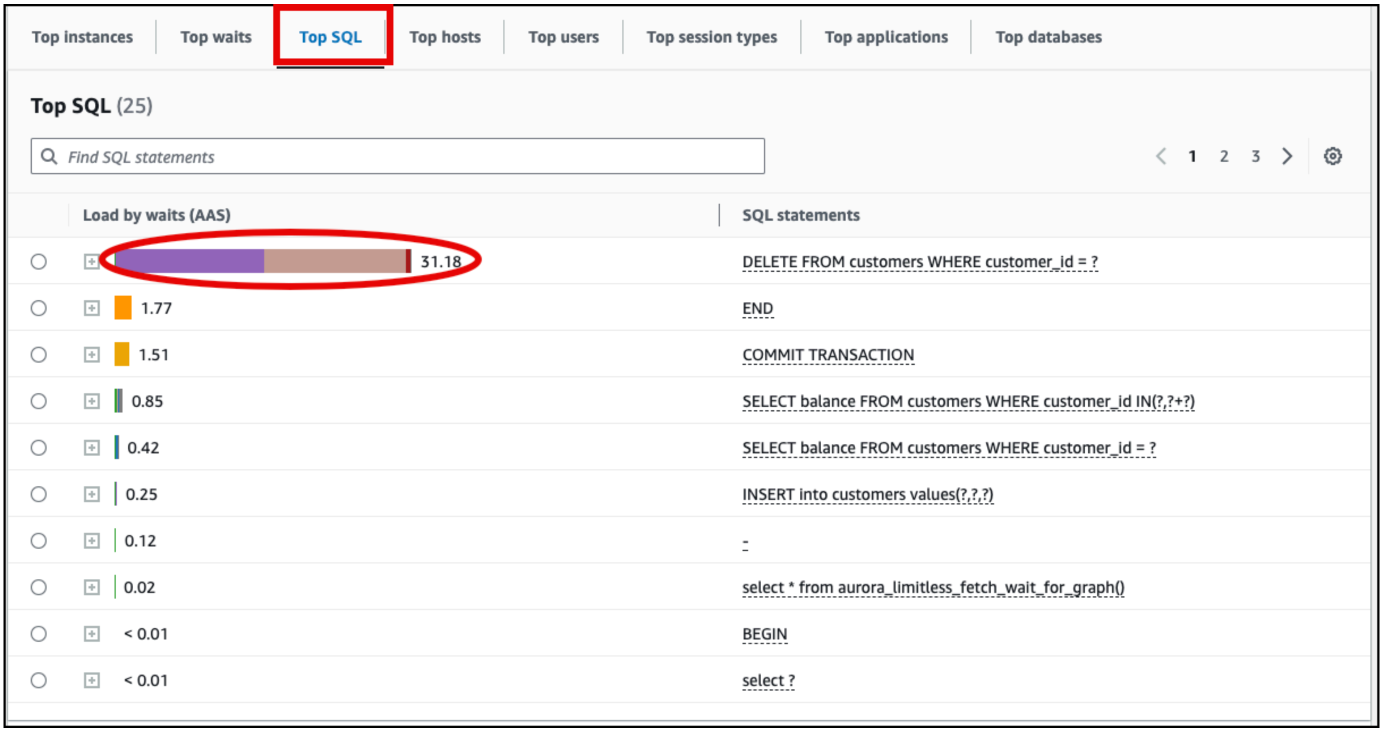
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. Aurora PostgreSQL Limitless Database를 선택합니다. 해당 Aurora PostgreSQL Limitless Database에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(DB 로드) 섹션에서 분할 기준에서 대기를 선택합니다. AAS 수와 예상 vCPU를 보려면 다음으로 보기보기에서 절대를 선택합니다.

   평균 활성 세션 차트는 Aurora PostgreSQL Limitless Database의 인스턴스에 대한 DB 로드를 보여줍니다.



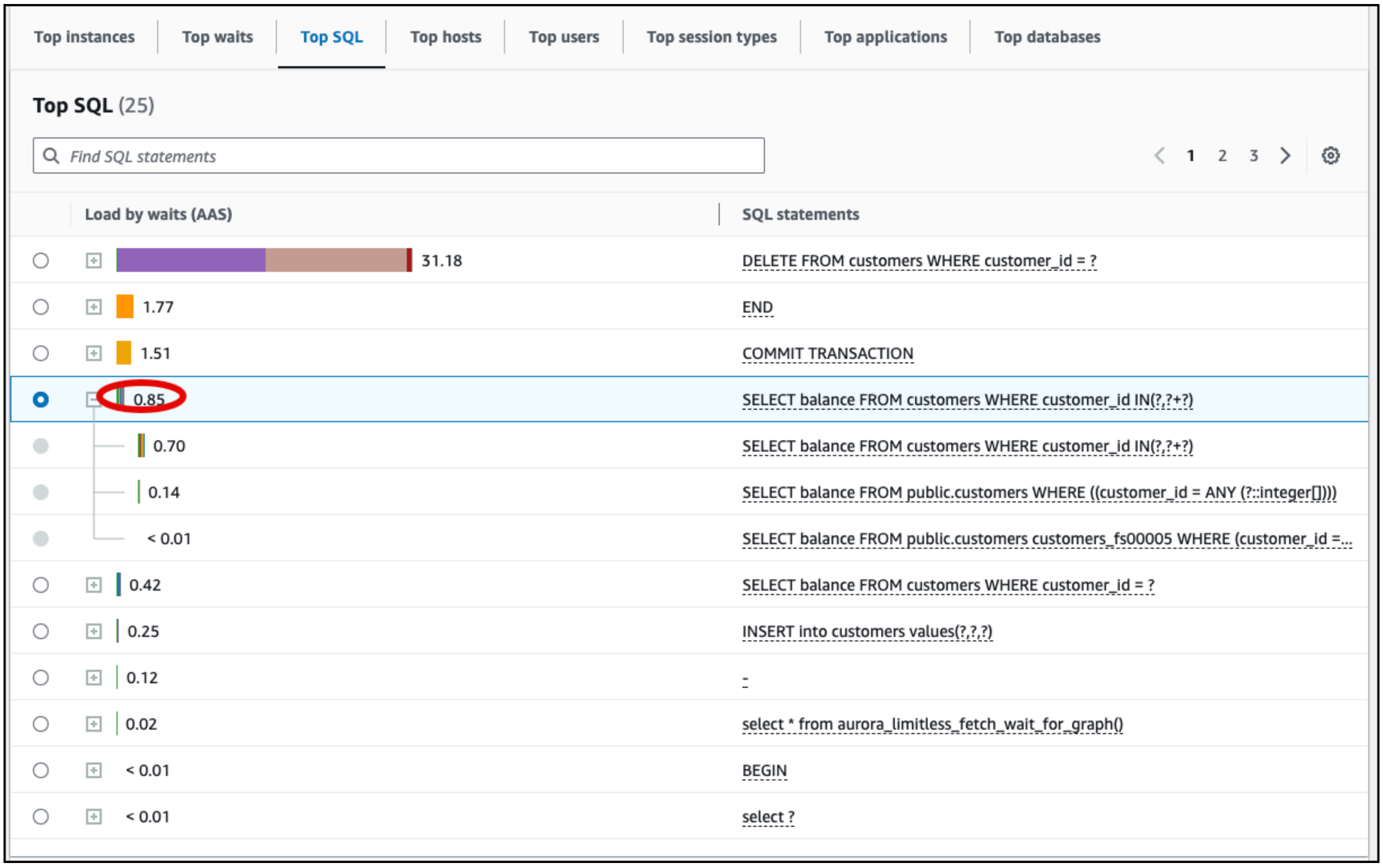
5. 상위 SQL 탭까지 아래로 스크롤합니다.

다음 예시에서 대기를 기준으로 로드가 가장 높은 SQL 문은 DELETE 문입니다.



6. SQL 문을 선택하여 구성 요소 문으로 확장합니다.

   다음 예시에서 SELECT 문에는 3개의 구성 요소 문이 있습니다.



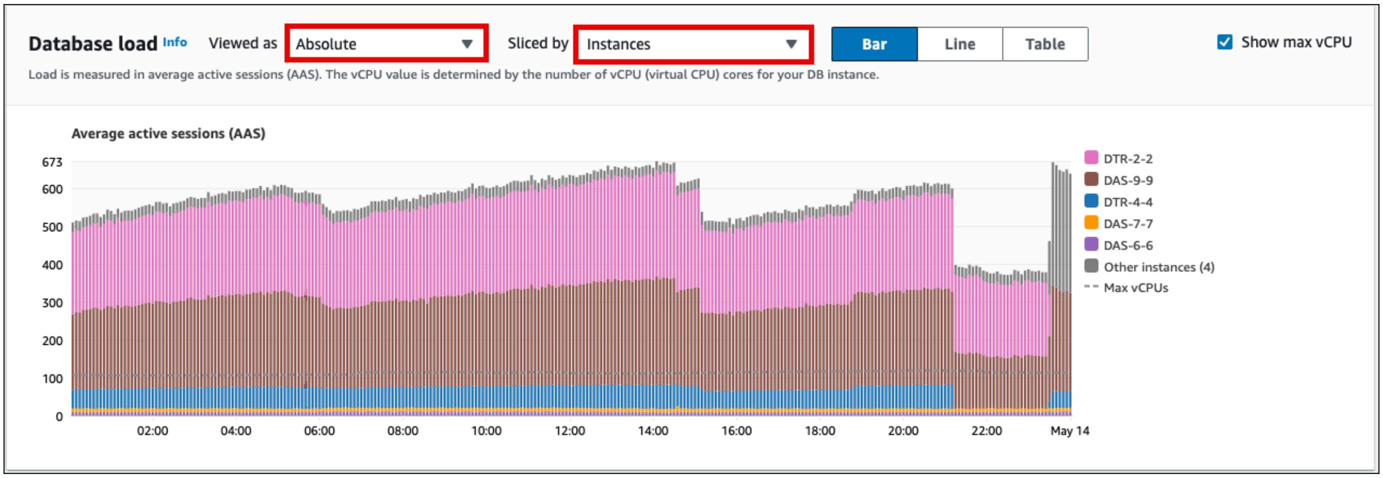
성능 개선 도우미 대시보드를 사용하여 Aurora PostgreSQL Limitless Database의 로드 분산 분석

Aurora PostgreSQL Limitless Database의 인스턴스에 대한 로드 분산의 균형을 맞추고 싶을 수 있습니다. Aurora PostgreSQL Limitless Database에서 인스턴스의 로드 분산을 분석하려면 다음 프로시저를 사용합니다.

콘솔을 사용하여 Aurora PostgreSQL Limitless Database에서 인스턴스의 로드 분산을 분석하는 방법

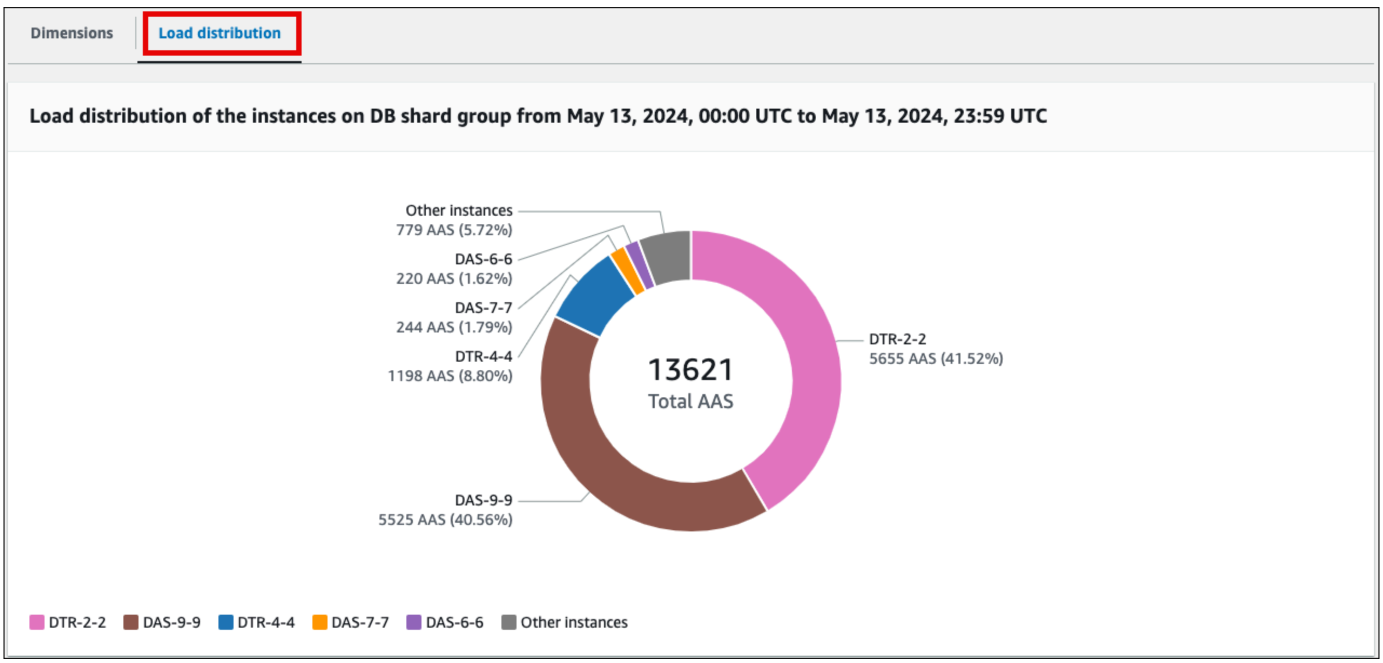
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. Aurora PostgreSQL Limitless Database를 선택합니다. 해당 Aurora PostgreSQL Limitless Database에 대한 성능 개선 도우미 대시보드가 표시됩니다.
4. 데이터베이스 로드(DB 로드) 섹션에서 분할 기준에서 인스턴스를 선택합니다. Aurora PostgreSQL Limitless Database의 모든 인스턴스에 대한 AAS 수와 예상 vCPU를 보려면 다음으로 보기를 절대로 선택합니다.

   평균 활성 세션 차트는 Aurora PostgreSQL Limitless Database의 인스턴스에 대한 DB 로드를 보여줍니다.



5. Aurora PostgreSQL Limitless Database에서 인스턴스의 로드 분산 차트를 보려면 로드 분산 탭을 선택합니다.

   다음 예시에서 DB 로드가 가장 높은 인스턴스는 DTR-2-2입니다.



Amazon GuardDuty RDS Protection을 사용하여 Aurora PostgreSQL Limitless Database 모니터링

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

GuardDuty RDS Protection은 Amazon Aurora 데이터베이스에 대한 잠재적 액세스 위협 관련 로그인 이벤트를 분석하고 프로파일링합니다. RDS 보호를 켜면 GuardDuty는 Aurora 데이터베이스에서 RDS 로그인 이벤트를 사용합니다. RDS Protection은 이러한 이벤트를 모니터링하고 잠재적인 내부자 위협 또는 외부 행위자가 있는지 프로파일링합니다.

Aurora의 GuardDuty RDS Protection에 대한 자세한 내용은 [Amazon Aurora용 Amazon GuardDuty RDS Protection을 이용한 위협 모니터링](#) 섹션을 참조하세요.

GuardDuty RDS Protection 활성화에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 [GuardDuty RDS Protection](#)을 참조하세요.

Aurora PostgreSQL Limitless Database의 함수 및 뷰

Aurora PostgreSQL Limitless Database에 함수와 뷰가 추가되었습니다. 이는 해당 Aurora PostgreSQL 함수 및 뷰를 기반으로 합니다.

> **Note**
> 진행 중인 트랜잭션이 있는 경우 일부 통계는 일관되지 않은 결과를 반환할 수 있습니다.

주제

- Aurora PostgreSQL Limitless Database 함수
- Aurora PostgreSQL Limitless Database 뷰

Aurora PostgreSQL Limitless Database 함수

다음 표에는 Aurora PostgreSQL Limitless Database의 새로운 함수가 나와 있습니다.

> **Note**
> 이 표에 나열된 함수는 `rds_aurora` 스키마에 있습니다. Limitless Database 함수를 사용할 때는 `rds_aurora.`*`object_name`*.과 같은 정규화된 객체 이름을 포함해야 합니다.

Aurora PostgreSQL Limitless Database 함수	해당 Aurora PostgreSQL 함수
limitless_backend_dsid	pg_backend_pid
limitless_cancel_session	pg_cancel_backend
limitless_stat_clear_snapshot	pg_stat_clear_snapshot
limitless_stat_database_size	pg_database_size
limitless_stat_get_snapshot_timestamp	pg_stat_get_snapshot_timestamp
limitless_stat_prepared_xacts	pg_prepared_xacts

Aurora PostgreSQL Limitless Database 함수	해당 Aurora PostgreSQL 함수
limitless_stat_relation_sizes	pg_indexes_size, pg_relation_size, pg_table_size, pg_total_relation_size
limitless_stat_reset	pg_stat_reset
limitless_stat_statements_reset	pg_stat_statements_reset
limitless_stat_system_waits	aurora_stat_system_waits
limitless_terminate_session	pg_terminate_backend
limitless_wait_report	aurora_wait_report

다음 예시에서는 Aurora PostgreSQL Limitless Database 함수에 대한 세부 정보를 제공합니다. PostgreSQL 함수에 대한 자세한 내용은 PostgreSQL 설명서에서 Functions and operators를 참조하세요.

limitless_backend_dsid

`limitless_backend_dsid` 함수는 현재 세션의 분산 세션 ID를 반환합니다. 분산 세션은 DB 샤드 그룹의 라우터에서 실행되며 DB 샤드 그룹의 샤드 중 하나 이상에서 이루어지는 백엔드 프로세스를 포함합니다.

다음 예시에서는 `limitless_backend_dsid` 함수를 사용하는 방법을 보여줍니다.

```
SELECT rds_aurora.limitless_backend_dsid();

limitless_backend_dsid
------------------------
8CACD7B04D0FC2A5
(1 row)
```

limitless_cancel_session

`limitless_cancel_session` 함수는 `pg_cancel_backend`와 비슷하게 작동하지만 `SIGINT`(중단 신호)를 전송하여 제공된 분산 세션 ID와 관련된 모든 백엔드 프로세스를 취소하려고 합니다.

입력 파라미터는 다음과 같습니다.

- distributed_session_id(text) - 취소할 분산 세션의 ID입니다.

출력 파라미터는 다음과 같습니다.

- subcluster_id(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- pid(text) - 백엔드 프로세스 ID입니다.
- success(boolean) - 취소의 성공 여부입니다.

다음 예시에서는 limitless_cancel_session 함수를 사용하는 방법을 보여줍니다.

```
SELECT * FROM rds_aurora.limitless_cancel_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_stat_clear_snapshot

limitless_stat_clear_snapshot 함수는 모든 노드에서 현재 통계 스냅샷 또는 캐싱된 정보를 삭제합니다.

다음 예시에서는 limitless_stat_clear_snapshot 함수를 사용하는 방법을 보여줍니다.

```
SELECT rds_aurora.limitless_stat_clear_snapshot();
```

limitless_stat_database_size

limitless_stat_database_size 함수는 DB 샤드 그룹의 데이터베이스 크기를 반환합니다.

입력 파라미터는 다음과 같습니다.

- dbname(name) - 크기를 가져올 데이터베이스입니다.

출력 파라미터는 다음과 같습니다.

- subcluster_id(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- subcluster_type(text) - 이 프로세스가 속한 하위 클러스터 유형으로, router 또는 shard입니다.
- db_size - 이 하위 클러스터의 데이터베이스 크기를 바이트 단위로 표시합니다.

다음 예시에서는 limitless_stat_database_size 함수를 사용하는 방법을 보여줍니다.

```
SELECT * FROM rds_aurora.limitless_stat_database_size('postgres_limitless');

 subcluster_id | subcluster_type | db_size
---------------+-----------------+----------
             1 | router          |  8895919
             2 | router          |  8904111
             3 | shard           | 21929391
             4 | shard           | 21913007
             5 | shard           | 21831087
(5 rows)
```

limitless_stat_get_snapshot_timestamp

`limitless_stat_get_snapshot_timestamp` 함수는 현재 통계 스냅샷의 타임스탬프를 반환하거나 통계 스냅샷이 생성되지 않은 경우 NULL을 반환합니다. `stats_fetch_consistency`가 `snapshot`으로 설정된 경우 트랜잭션에서 누적 통계에 처음 액세스할 때 스냅샷이 생성됩니다. 모든 노드의 스냅샷 타임스탬프에 대한 통합 뷰를 반환합니다. `subcluster_id` 및 `subcluster_type` 열에는 데이터가 소스 노드가 표시됩니다.

다음 예시에서는 `limitless_stat_get_snapshot_timestamp` 함수를 사용하는 방법을 보여줍니다.

```
SELECT * FROM rds_aurora.limitless_stat_get_snapshot_timestamp();

 subcluster_id | subcluster_type | snapshot_timestamp
---------------+-----------------+--------------------
             1 | router          |
             2 | router          |
             3 | shard           |
             4 | shard           |
             5 | shard           |
(5 rows)
```

limitless_stat_prepared_xacts

`limitless_stat_prepared_xacts` 함수는 현재 두 단계 커밋을 위해 준비된 모든 노드의 트랜잭션에 대한 정보를 반환합니다. 자세한 내용은 PostgreSQL 설명서에서 pg_prepared_xacts를 참조하세요.

다음 예시에서는 `limitless_stat_prepared_xacts` 함수를 사용하는 방법을 보여줍니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_prepared_xacts;
```

```
 subcluster_id | subcluster_type | transaction_id |              gid               |
         prepared              |  owner_id  |    database_id
---------------+-----------------+----------------+--------------------------------
+-------------------------------+------------+--------------------
 8             | shard           |        5815978 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659603+00 | auroraperf | postgres_limitless
 12            | shard           |        4599138 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659637+00 | auroraperf | postgres_limitless
(2 rows)
```

limitless_stat_relation_sizes

`limitless_stat_relation_sizes` 함수는 DB 샤드 그룹에 있는 테이블의 다양한 크기를 반환합니다.

입력 파라미터는 다음과 같습니다.

- `relnspname`(name) - 테이블이 포함된 스키마의 이름입니다.
- `relname`(name) - 테이블의 이름입니다.

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.
- `main_size` - 이 노드의 기본 데이터 포크 크기를 바이트 단위로 표시합니다.
- `fsm_size` - 이 노드의 테이블에 대한 여유 공간 맵의 크기를 바이트 단위로 표시합니다.
- `vm_size` - 이 노드의 테이블에 대한 가시성 맵의 크기를 바이트 단위로 표시합니다.
- `init_size` - 이 노드에 있는 테이블 초기화의 크기를 바이트 단위로 표시합니다.
- `toast_size` - 이 포크의 테이블과 연결된 토스트 테이블의 크기를 바이트 단위로 표시합니다.
- `index_size` - 이 노드의 테이블에 대한 모든 인덱스의 크기를 바이트 단위로 표시합니다.
- `total_size` - 이 노드에 있는 테이블의 모든 세그먼트의 크기를 바이트 단위로 표시합니다.

다음 예시에서는 `limitless_stat_relation_sizes` 함수를 사용하는 방법을 보여줍니다(일부 열은 생략됨).

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customers');
```

```
 subcluster_id | subcluster_type | main_size | fsm_size | vm_size | toast_size |
 table_size | total_size
---------------+-----------------+-----------+----------+---------+------------
+------------+------------
             1 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             2 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             3 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             4 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             5 | shard           |   3981312 |  4227072 | 1409024 |    1409024 |
 11026432 |   11026432
(5 rows)
```

limitless_stat_reset

`limitless_stat_reset` 함수는 현재 데이터베이스의 모든 통계 카운터를 0으로 재설정합니다. `track_functions`이 활성화되면 `limitless_stat_database`의 `stats_reset` 열에 데이터베이스에 대한 통계가 마지막으로 재설정된 시간이 표시됩니다. 기본적으로 `limitless_stat_reset`은 슈퍼 사용자만 실행할 수 있습니다. EXECUTE 권한을 사용하여 다른 사용자에게 권한을 부여할 수 있습니다.

다음 예시에서는 `limitless_stat_reset` 함수를 사용하는 방법을 보여줍니다.

```
SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

 tup_inserted | tup_deleted
--------------+-------------
          896 |           0
(1 row)

SELECT rds_aurora.limitless_stat_reset();

limitless_stat_reset
---------------------
(1 row)

SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';
```

```
tup_inserted | tup_deleted
-------------+-------------
           0 |           0
(1 row)
```

limitless_stat_statements_reset

limitless_stat_statements_reset 함수는 limitless_stat_statements가 지금까지 수집한 지정된 username, dbname, distributed_query_id 및 queryid 파라미터에 해당하는 통계를 삭제합니다. 파라미터가 지정되지 않은 경우 각각에 기본값 "" 또는 0(유효하지 않음)이 사용되며 다른 파라미터와 일치하는 통계가 재설정됩니다. 파라미터가 지정되지 않았거나 지정된 모든 파라미터가 "" 또는 0(유효하지 않음)인 경우 함수는 모든 통계를 삭제합니다. limitless_stat_statements 뷰의 모든 통계가 삭제되면 함수는 limitless_stat_statements_info 뷰의 통계도 재설정합니다.

입력 파라미터는 다음과 같습니다.

- username(name) - 문을 쿼리한 사용자입니다.
- dbname(name) - 쿼리가 실행된 데이터베이스입니다.
- distributed_query_id(bigint) - 코디네이터 노드에서 상위 쿼리의 쿼리 ID입니다. 이 열은 상위 쿼리인 경우 NULL입니다. 코디네이터 노드는 분산 쿼리 ID를 참가자 노드로 푸시다운합니다. 따라서 참가자 노드의 경우 분산 쿼리 ID와 쿼리 ID의 값이 다릅니다.
- queryid(bigint) - 문의 쿼리 ID입니다.

다음 예시에서는 limitless_stat_statements_reset 함수를 사용하여 limitless_stat_statements가 수집한 모든 통계를 재설정하는 방법을 보여줍니다.

```
SELECT rds_aurora.limitless_stat_statements_reset();
```

limitless_stat_system_waits

limitless_stat_system_waits 함수는 모든 노드에서 인스턴스의 시스템 전체 대기 활동을 보고하는 aurora_stat_system_waits의 대기 이벤트 데이터에 대한 통합 뷰를 반환합니다. subcluster_id 및 subcluster_type 열에는 데이터가 소스 노드가 표시됩니다.

다음 예시에서는 limitless_stat_system_waits 함수를 사용하는 방법을 보여줍니다.

```
postgres_limitless=> SELECT *
FROM rds_aurora.limitless_stat_system_waits() lssw, pg_catalog.aurora_stat_wait_event()
 aswe
```

```
WHERE lssw.event_id=aswe.event_id and aswe.event_name='LimitlessTaskScheduler';

 subcluster_id | subcluster_type | type_id | event_id  | waits  |  wait_time   |
 event_name
---------------+-----------------+---------+-----------+--------+--------------
+------------------------
             1 | router          |      12 | 201326607 | 677068 | 616942216307 |
 LimitlessTaskScheduler
             2 | router          |      12 | 201326607 | 678586 | 616939897111 |
 LimitlessTaskScheduler
             3 | shard           |      12 | 201326607 | 756640 | 616965545172 |
 LimitlessTaskScheduler
             4 | shard           |      12 | 201326607 | 755184 | 616958057620 |
 LimitlessTaskScheduler
             5 | shard           |      12 | 201326607 | 757522 | 616963183539 |
 LimitlessTaskScheduler
(5 rows)
```

limitless_terminate_session

limitless_terminate_session 함수는 pg_terminate_backend와 비슷하게 작동하지만 SIGTERM(종료 신호)를 전송하여 제공된 분산 세션 ID와 관련된 모든 백엔드 프로세스를 종료하려고 합니다.

입력 파라미터는 다음과 같습니다.

- distributed_session_id(text) - 종료할 분산 세션의 ID입니다.

출력 파라미터는 다음과 같습니다.

- subcluster_id(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- pid(text) - 백엔드 프로세스 ID입니다.
- success(boolean) - 프로세스가 성공적으로 종료되었는지를 나타냅니다.

다음 예시에서는 limitless_terminate_session 함수를 사용하는 방법을 보여줍니다.

```
SELECT * FROM rds_aurora.limitless_terminate_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_wait_report

`limitless_wait_report` 함수는 모든 노드에서 일정 기간 동안의 대기 이벤트 활동을 반환합니다. `subcluster_id` 및 `subcluster_type` 열에는 데이터가 소스 노드가 표시됩니다.

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.

나머지 열은 `aurora_wait_report`의 열과 동일합니다.

다음 예시에서는 `limitless_wait_report` 함수를 사용하는 방법을 보여줍니다.

```
postgres_limitless=> select * from rds_aurora.limitless_wait_report();

 subcluster_id | subcluster_type | type_name | event_name | waits | wait_time |
 ms_per_wait | waits_per_xact | ms_per_xact
---------------+-----------------+-----------+------------+-------+-----------
+-------------+--------------- +-------------
             1 | router          | Client    | ClientRead |    57 | 741550.14 |
 13009.652 |           0.19 |    2505.237
             5 | shard           | Client    | ClientRead |    54 | 738897.68 |
 13683.290 |           0.18 |    2496.276
             4 | shard           | Client    | ClientRead |    54 | 738859.53 |
 13682.584 |           0.18 |    2496.147
             2 | router          | Client    | ClientRead |    53 | 719223.64 |
 13570.257 |           0.18 |    2429.810
             3 | shard           | Client    | ClientRead |    54 | 461720.40 |
 8550.378 |           0.18 |    1559.86
```

Aurora PostgreSQL Limitless Database 뷰

다음 표에는 Aurora PostgreSQL Limitless Database의 새 뷰가 나와 있습니다.

> **Note**
>
> 이 테이블에 나열된 뷰는 `rds_aurora` 스키마에 있습니다. Limitless Database 뷰를 사용할 때는 `rds_aurora.object_name`과 같은 정규화된 객체 이름을 포함해야 합니다.

Aurora PostgreSQL Limitless Database 뷰	해당 Aurora PostgreSQL 뷰
limitless_database	pg_database
limitless_locks	pg_locks
limitless_stat_activity	pg_stat_activity
limitless_stat_all_indexes	pg_stat_all_indexes
limitless_stat_all_tables	pg_stat_all_tables
limitless_stat_database	pg_stat_database
limitless_stat_progress_vacuum	pg_stat_progress_vacuum
limitless_stat_statements	pg_stat_statements
limitless_stat_subclusters	없음
limitless_stat_statements_info	pg_stat_statements_info
limitless_statio_all_indexes	pg_statio_all_indexes
limitless_statio_all_tables	pg_statio_all_tables
limitless_tables	pg_tables
limitless_table_collocations	없음
limitless_table_collocation_distributions	없음

다음 예시에서는 Aurora PostgreSQL Limitless Database 뷰에 대한 세부 정보를 제공합니다. PostgreSQL 뷰에 대한 자세한 내용은 PostgreSQL 설명서에서 Viewing statistics을 참조하세요.

> **Note**
> 일부 통계 뷰는 진행 중인 트랜잭션이 있는 경우 일관되지 않은 결과를 반환할 수 있습니다.

limitless_database

이 뷰에는 DB 샤드 그룹에서 사용 가능한 데이터베이스에 대한 정보가 포함되어 있습니다. 예시:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, oid, datname, datacl
 FROM rds_aurora.limitless_database;

 subcluster_id | subcluster_type |  oid  |       datname       |
                                       datacl

---------------+-----------------+-------+--------------------
+-------------------------------------------------------------------------------------
 2             | router          |     4 | template0           | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 2             | router          |     5 | postgres            |
 2             | router          | 16384 | rdsadmin            |
 {rdsadmin=CTc/rdsadmin,rds_aurora_limitless_metadata_admin=c/
rdsadmin,rds_aurora_limitless_heat_mgmt_admin=c/rdsadmin}
 2             | router          | 16477 | postgres_limitless |
 2             | router          |     1 | template1           | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 6             | shard           |     4 | template0           | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(텍스트) - 하위 클러스터의 ID(노드)
- `subcluster_type`(텍스트) - 하위 클러스터(노드), 라우터 또는 샤드의 유형

나머지 열은 `pg_database`의 열과 동일합니다.

limitless_locks

이 뷰에는 노드마다 프로세스당 행이 하나씩 포함됩니다. 데이터베이스 서버의 활성 프로세스가 보유한 잠금 정보에 대한 액세스를 제공합니다.

Example 두 트랜잭션으로 잠금 만들기

이 예시에서는 두 라우터에서 두 트랜잭션을 동시에 실행합니다.

```
# Transaction 1 (run on router 1)
BEGIN;
SET search_path = public;
SELECT * FROM customers;
INSERT INTO customers VALUES (400,'foo','bar');

# Transaction 2 (run on router 2)
BEGIN;
SET search_path = public;
ALTER TABLE customers ADD COLUMN phone VARCHAR;
```

첫 번째 트랜잭션이 실행됩니다. 후속 트랜잭션은 첫 번째 트랜잭션이 완료될 때까지 기다려야 합니다. 따라서 두 번째 트랜잭션은 잠금으로 차단됩니다. 근본 원인을 확인하기 위해 `limitless_locks`을 `limitless_stat_activity`와 조인하여 명령을 실행합니다.

```
# Run on router 2
SELECT distributed_session_id, state, usename, query, query_start
FROM rds_aurora.limitless_stat_activity
WHERE distributed_session_id in (
SELECT distributed_session_id
FROM rds_aurora.limitless_locks
WHERE relname = 'customers'
);

 distributed_session_id | state               | usename                  | query
                                    | query_start
------------------------+---------------------+--------------------------
+----------------------------------- -------------+-------------------------------
 47BDE66E9A5E8477       | idle in transaction | limitless_metadata_admin | INSERT INTO
 customers VALUES (400,'foo','bar'); | 2023-04-13 17:44:45.152244+00
 2AD7F370202D0FA9       | active              | limitless_metadata_admin | ALTER TABLE
 customers ADD COLUMN phone VARCHAR; | 2023-04-13 17:44:55.113388+00
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                      |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                      |
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                      |
```

```
 2AD7F370202D0FA9       |                        | limitless_auth_admin     |
 <insufficient privilege>                           |
(6 rows)
```

Example 잠금을 명시적으로 만들기

이 예시에서는 잠금을 명시적으로 만든 다음 `limitless_locks` 뷰를 사용하여 잠금을 확인합니다(일부 열은 생략됨).

```
BEGIN;
SET search_path = public;
LOCK TABLE customers IN ACCESS SHARE MODE;
SELECT * FROM rds_aurora.limitless_locks WHERE relname = 'customers';

 subcluster_id | subcluster_type | distributed_session_id | locktype |      datname
   | relnspname |  relname   | virtualtransaction |  pid   |      mode
---------------+-----------------+------------------------+----------
+-------------------+------------+ ----------+-------------------+-------
+-----------------
             1 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600787          | 59564 |
 AccessShareLock
             2 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600405          | 67130 |
 AccessShareLock
             3 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 15/473401          | 27735 |
 AccessShareLock
             4 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473524          | 27734 |
 AccessShareLock
             5 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/472935          | 27737 |
 AccessShareLock
             6 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473015          | 48660 |
 AccessShareLock
(6 rows)
```

limitless_stat_activity

이 뷰에는 노드마다 프로세스당 행이 하나씩 포함됩니다. 이는 전체 시스템 상태를 추적하고 시간이 오래 걸리는 프로세스를 분류하는 데 사용할 수 있습니다. 예시:

```
postgres=# SELECT
    subcluster_id,
    subcluster_type,
    distributed_session_id,
    distributed_session_state,
    datname,
    distributed_query_id,
    is_sso_query
FROM
    rds_aurora.limitless_stat_activity
WHERE
    distributed_session_id in ('D2470C97E3D07E06', '5A3CD7B8E5FD13FF')
    order by  distributed_session_id;

 subcluster_id | subcluster_type | distributed_session_id | distributed_session_state |
      datname       | distributed_query_id | is_sso_query
---------------+-----------------+------------------------+---------------------------
+--------------------+----------------------+--------------
 2             | router          | 5A3CD7B8E5FD13FF       | coordinator               |
 postgres_limitless |                      | f
 3             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 4             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 2             | router          | D2470C97E3D07E06       | coordinator               |
 postgres_limitless |                      | t
 3             | shard           | D2470C97E3D07E06       | participant               |
 postgres_limitless |  4058400544464210222 |
(5 rows)
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.
- `distributed_session_id`(text) - 이 프로세스가 속한 분산 세션의 ID입니다.
- `distributed_session_state`(text) - 코디네이터, 참가자 또는 독립 실행형/미분산 프로세스(NULL로 표시됨) 중 무엇인지를 나타냅니다.
- `datname`(text) - 이 프로세스가 연결된 데이터베이스입니다.

- distributed_query_id(bigint) - 코디네이터 노드에서 상위 쿼리의 쿼리 ID입니다. 이 열은 상위 쿼리인 경우 NULL입니다. 코디네이터 노드는 분산 쿼리 ID를 참가자 노드로 푸시다운합니다. 따라서 참가자 노드의 경우 분산 쿼리 ID와 쿼리 ID의 값이 다릅니다.
- is_sso_query(text) - 쿼리가 단일 샤드에 최적화되었는지, 아닌지를 알려줍니다.

나머지 열은 pg_stat_activity의 열과 동일합니다.

limitless_stat_all_indexes

이 뷰에는 DB 샤드 그룹의 인덱스에 대한 사용량 통계가 포함되어 있습니다. 예시:

```
postgres_limitless=> SELECT schemaname, relname, indexrelname, idx_scan
  FROM rds_aurora.limitless_stat_all_indexes
  WHERE relname LIKE 'orders_ts%' ORDER BY indexrelname LIMIT 10;

 schemaname |    relname     |    indexrelname     | idx_scan
------------+----------------+---------------------+----------
 ec_sample  | orders_ts00001 | orders_ts00001_pkey |   196801
 ec_sample  | orders_ts00002 | orders_ts00002_pkey |   196703
 ec_sample  | orders_ts00003 | orders_ts00003_pkey |   196376
 ec_sample  | orders_ts00004 | orders_ts00004_pkey |   197966
 ec_sample  | orders_ts00005 | orders_ts00005_pkey |   195301
 ec_sample  | orders_ts00006 | orders_ts00006_pkey |   195673
 ec_sample  | orders_ts00007 | orders_ts00007_pkey |   194475
 ec_sample  | orders_ts00008 | orders_ts00008_pkey |   191694
 ec_sample  | orders_ts00009 | orders_ts00009_pkey |   193744
 ec_sample  | orders_ts00010 | orders_ts00010_pkey |   195421
(10 rows)
```

limitless_stat_all_tables

이 뷰에는 DB 샤드 그룹의 현재 데이터베이스에 있는 모든 테이블에 대한 통계가 포함되어 있습니다. 이는 vacuum 작업 및 데이터 조작 언어(DML) 작업을 추적할 때 유용합니다. 예시:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, relname,
 n_ins_since_vacuum, n_tup_ins, last_vacuum
  FROM rds_aurora.limitless_stat_all_tables
  WHERE relname LIKE 'orders_ts%' ORDER BY relname LIMIT 10;

 subcluster_id | subcluster_type |    relname     | n_ins_since_vacuum | n_tup_ins |
 last_vacuum
---------------+-----------------+----------------+--------------------+-----------
+-------------
```

```
 5             | shard           | orders_ts00001 |              34779 |     196083 |
 5             | shard           | orders_ts00002 |              34632 |     194721 |
 5             | shard           | orders_ts00003 |              34950 |     195965 |
 5             | shard           | orders_ts00004 |              34745 |     197283 |
 5             | shard           | orders_ts00005 |              34879 |     195754 |
 5             | shard           | orders_ts00006 |              34340 |     194605 |
 5             | shard           | orders_ts00007 |              33779 |     192203 |
 5             | shard           | orders_ts00008 |              33826 |     191293 |
 5             | shard           | orders_ts00009 |              34660 |     194117 |
 5             | shard           | orders_ts00010 |              34569 |     195560 |
(10 rows)
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.
- `relname`(name) - 테이블의 이름입니다.

나머지 열은 `pg_stat_all_tables`의 열과 동일합니다.

limitless_stat_database

이 뷰에는 DB 샤드 그룹의 모든 데이터베이스에 대한 통계가 포함되어 있습니다. 노드마다 데이터베이스당 행 하나를 반환합니다. 예시:

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    datname,
    blks_read,
    blks_hit
FROM
    rds_aurora.limitless_stat_database
WHERE
    datname='postgres_limitless';
 subcluster_id | subcluster_type |      datname       | blks_read | blks_hit
---------------+-----------------+--------------------+-----------+----------
             1 | router          | postgres_limitless |       484 | 34371314
             2 | router          | postgres_limitless |       673 | 33859317
             3 | shard           | postgres_limitless |      1299 | 17749550
             4 | shard           | postgres_limitless |      1094 | 17492849
             5 | shard           | postgres_limitless |      1036 | 17485098
```

```
              6 | shard           | postgres_limitless |      1040 | 17437257
(6 rows)
```

출력 파라미터는 다음과 같습니다.

- subcluster_id(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- subcluster_type(text) - 이 프로세스가 속한 하위 클러스터 유형으로, router 또는 shard입니다.
- datname(name) - 데이터베이스의 이름입니다.

나머지 열은 pg_stat_database의 열과 동일합니다.

limitless_stat_progress_vacuum

이 뷰에는 지속적 vacuum 작업에 대한 정보가 포함되어 있습니다. 예시:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;

-[ RECORD 1 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | A56D96E2A5C9F426
pid                    | 5270
datname                | postgres
nspname                | public
relname                | customer_ts2
phase                  | vacuuming heap
heap_blks_total        | 130500
heap_blks_scanned      | 100036
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 11184810
num_dead_tuples        | 0

-[ RECORD 2 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | 56DF26A89EC23AB5
pid                    | 6854
datname                | postgres
nspname                | public
relname                | sales_ts1
phase                  | vacuuming heap
heap_blks_total        | 43058
```

```
heap_blks_scanned        | 24868
heap_blks_vacuumed       | 0
index_vacuum_count       | 0
max_dead_tuples          | 8569523
num_dead_tuples          | 0
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.
- `distributed_session_id`(text) - Vacuum 작업을 시작한 세션의 식별자입니다.
- `datname`(name) - Vacuum이 수행되는 데이터베이스입니다.
- `nspname`(name) - Vacuum이 진행 중인 테이블의 스키마 이름입니다. Vacuum이 진행 중인 테이블이 사용자가 연결된 테이블과 동일한 데이터베이스에 있지 않은 경우 `null`입니다.
- `relname`(name) - Vacuum이 진행 중인 테이블의 이름입니다. Vacuum이 진행 중인 테이블이 사용자가 연결된 테이블과 동일한 데이터베이스에 있지 않은 경우 `null`입니다.

나머지 열은 `pg_stat_progress_vacuum`의 열과 동일합니다.

limitless_stat_statements

이 뷰는 모든 노드에서 실행되는 모든 SQL 문에 대한 계획 및 실행 통계를 추적하는 수단을 제공합니다.

> **Note**
>
> `limitless_stat_statements` 뷰를 사용하려면 [pg_stat_statements](https://www.postgresql.org/docs/current/pgstatstatements.html) 확장을 설치해야 합니다.
>
> ```
> -- CREATE EXTENSION must be run by a superuser
> CREATE EXTENSION pg_stat_statements;
>
> -- Verify that the extension is created on all nodes in the DB shard group
> SELECT distinct node_id
>     FROM rds_aurora.limitless_stat_statements
>     LIMIT 10;
> ```

다음 예시는 `limitless_stat_statements` 뷰 사용을 보여 줍니다.

```
postgres_limitless=> SELECT
 subcluster_id,
 subcluster_type,
 distributedqueryid,
 username,
 dbname,
 sso_calls
FROM
 rds_aurora.limitless_stat_statements;

 subcluster_id | subcluster_type |  distributedqueryid  |              username
       |       dbname       | sso_calls
---------------+-----------------+----------------------
+------------------------------------+--------------------+-----------
 2             | router          |                      | postgres
       | postgres_limitless |        0
 2             | router          |                      | postgres
       | postgres_limitless |        0
 2             | router          |                      | postgres
       | postgres_limitless |        0
 2             | router          |                      | postgres
       | postgres_limitless |        0
 2             | router          |                      | postgres
       | postgres_limitless |        0
 2             | router          |                      | postgres
       | postgres_limitless |        1
 3             | shard           | -7975178695405682176 | postgres
       | postgres_limitless |
[...]
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.
- `subcluster_type`(text) - 이 프로세스가 속한 하위 클러스터 유형으로, `router` 또는 `shard`입니다.
- `distributedqueryid`(bigint) - 코디네이터 노드에서 상위 쿼리의 쿼리 ID입니다. 이 열은 상위 쿼리인 경우 NULL입니다. 코디네이터 노드는 분산 쿼리 ID를 참가자 노드로 푸시다운합니다. 따라서 참가자 노드의 경우 분산 쿼리 ID와 쿼리 ID의 값이 다릅니다.
- `username`(name) - 문을 쿼리한 사용자입니다.
- `dbname`(name) - 쿼리가 실행된 데이터베이스입니다.
- `sso_calls`(name) - 문이 단일 샤드에 최적화된 횟수입니다.

나머지 열은 pg_stat_statements와 동일합니다.

limitless_stat_statements_info

이 뷰에는 `limitless_stat_statements` 뷰에 대한 통계가 포함되어 있습니다. 각 행에는 각 노드의 pg_stat_statements_info 뷰에 대한 데이터가 포함되어 있습니다. `subcluster_id` 열은 각 노드를 식별합니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_statements_info;

 subcluster_id | subcluster_type | dealloc |          stats_reset
---------------+-----------------+---------+-------------------------------
             1 | router          |       0 | 2023-06-30 21:22:09.524781+00
             2 | router          |       0 | 2023-06-30 21:21:40.834111+00
             3 | shard           |       0 | 2023-06-30 21:22:10.709942+00
             4 | shard           |       0 | 2023-06-30 21:22:10.740179+00
             5 | shard           |       0 | 2023-06-30 21:22:10.774282+00
             6 | shard           |       0 | 2023-06-30 21:22:10.808267+00
(6 rows)
```

출력 파라미터는 다음과 같습니다.

- `subcluster_id`(text) - 이 프로세스가 속한 하위 클러스터의 ID입니다.

나머지 열은 pg_stat_statements_info와 동일합니다.

limitless_stat_subclusters

이 뷰에는 라우터와 다른 노드 간의 네트워크 통계가 포함되어 있습니다. 여기에는 라우터 및 기타 노드 페어당 행이 포함됩니다. 예를 들면 다음과 같습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_subclusters;

 orig_subcluster | orig_instance_az | dest_subcluster | dest_instance_az |
 latency_us |       latest_collection       | failed_requests | received_bytes |
 sent_bytes | same_az_requests | cross_az_requests |     stat_reset_timestamp
-----------------+------------------+-----------------+------------------
+------------+-------------------------------+-----------------+----------------
+------------+------------------+-------------------+------------------------------
 3               | us-west-2b       | 2               | us-west-2a       |
 847 | 2024-10-07 17:25:39.518617+00 |               0 |       35668633 |   92090171
 |                0 |            302787 | 2024-10-05 12:39:55.239675+00
 3               | us-west-2b       | 4               | us-west-2b       |
 419 | 2024-10-07 17:25:39.546376+00 |               0 |      101190464 |  248795719
 |           883478 |                 0 | 2024-10-05 12:39:55.231218+00
```

```
3               | us-west-2b       | 5                | us-west-2c       |
   1396 | 2024-10-07 17:25:39.52122+00  |                0 |        72864849 |
172086292 |                0 |           557726 | 2024-10-05 12:39:55.196412+00
3               | us-west-2b       | 6                | us-west-2c       |
729 | 2024-10-07 17:25:39.54828+00  |                0 |        35668584 |   92090171
|                0 |           302787 | 2024-10-05 12:39:55.247334+00
3               | us-west-2b       | 7                | us-west-2a       |
   1702 | 2024-10-07 17:25:39.545307+00 |                0 |        71699576 |
171634844 |                0 |           556278 | 2024-10-05 12:39:52.715168+00
2               | us-west-2a       | 3                | us-west-2b       |
868 | 2024-10-07 17:25:40.293927+00 |                0 |        35659611 |   92011872
|                0 |           302817 | 2024-10-05 12:39:54.420758+00
2               | us-west-2a       | 4                | us-west-2b       |
786 | 2024-10-07 17:25:40.296863+00 |                0 |       102437253 |  251838024
|                0 |           895060 | 2024-10-05 12:39:54.404081+00
2               | us-west-2a       | 5                | us-west-2c       |
   1232 | 2024-10-07 17:25:40.292021+00 |                0 |        71990027 |
168828110 |                0 |           545453 | 2024-10-05 12:39:36.769549+00
```

출력 파라미터는 다음과 같습니다.

- `orig_subcluster`(text) - 통신이 시작되는 라우터의 ID입니다.
- `orig_subcluster_az`(text) - 발신자 라우터의 가용 영역(AZ)입니다.
- `dest_subcluster`(text) - 대상 노드의 ID입니다.
- `dest_subcluster_az`(text) - 대상 노드의 마지막으로 수집된 AZ입니다.
- `latency_us`(bigint) - 노드 간에 마지막으로 수집된 네트워크 지연 시간으로, 마이크로초 단위입니다. 노드에 연결할 수 없는 경우 값은 `0`입니다.
- `latest_collection`(timestamp) - 대상 노드의 최신 AZ 및 지연 시간 모음의 타임스탬프입니다.
- `failed_requests`(bigint) - 실패한 내부 요청의 누적 수입니다.
- `received_bytes`(bigint) - 이 노드에서 수신된 예상 누적 바이트 수입니다.
- `sent_bytes`(bigint) - 이 노드로 전송된 예상 누적 바이트 수입니다.
- `same_az_requests`(bigint) - 이 노드가 발신자 라우터와 동일한 AZ에 있는 경우 이 노드에 대한 내부 DB 요청의 누적 수입니다.
- `cross_az_requests`(bigint) - 이 노드가 발신자 라우터와 다른 AZ에 있을 때 이 노드에 대한 내부 DB 요청의 누적 수입니다.
- `stat_reset_timestamp`(timestamp) - 이 뷰의 누적 통계가 마지막으로 재설정된 시점의 타임스탬프입니다.

limitless_statio_all_indexes

이 뷰에는 DB 샤드 그룹의 모든 인덱스에 대한 입출력(I/O) 통계가 포함되어 있습니다. 예시:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_statio_all_indexes WHERE
 relname like'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      |
 indexrelname             | idx_blks_read | idx_blks_hit
---------------+-----------------+------------+-------------------
+-----------------------------------+ --------------+--------------
             3 | shard           | public     | customers_ts00002 |
 customers_ts00002_customer_name_idx |             1 |            0
             3 | shard           | public     | customers_ts00001 |
 customers_ts00001_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00003 |
 customers_ts00003_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00004 |
 customers_ts00004_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00005 |
 customers_ts00005_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00006 |
 customers_ts00006_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00007 |
 customers_ts00007_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00008 |
 customers_ts00008_customer_name_idx |             1 |            0
(8 rows)
```

limitless_statio_all_tables

이 뷰에는 DB 샤드 그룹의 모든 테이블에 대한 입출력(I/O) 통계가 포함되어 있습니다. 예시:

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit
FROM
    rds_aurora.limitless_statio_all_tables
WHERE
    relname LIKE 'customers_ts%';
```

```
 subcluster_id | subcluster_type | schemaname |      relname      | heap_blks_read |
 heap_blks_hit
---------------+-----------------+------------+-------------------+----------------
+---------------
             3 | shard           | public     | customers_ts00002 |            305 |
      57780
             3 | shard           | public     | customers_ts00001 |            300 |
      56972
             4 | shard           | public     | customers_ts00004 |            302 |
      57291
             4 | shard           | public     | customers_ts00003 |            302 |
      57178
             5 | shard           | public     | customers_ts00006 |            300 |
      56932
             5 | shard           | public     | customers_ts00005 |            302 |
      57386
             6 | shard           | public     | customers_ts00008 |            300 |
      56881
             6 | shard           | public     | customers_ts00007 |            304 |
      57635
(8 rows)
```

limitless_tables

이 뷰에는 Aurora PostgreSQL Limitless Database의 테이블에 대한 정보가 포함되어 있습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | placeholder | active       | placeholder |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     | HASH
 (order_id)
         2 |     18579 | public      | customer    | active       | sharded     | HASH
 (cust_id)
(4 rows)
```

limitless_table_collocations

이 뷰에는 공동 배치된 샤딩된 테이블에 대한 정보가 포함되어 있습니다.

다음 예시에서는 orders 및 customers 테이블이 공동 배치되고 users 및 followers 테이블이 공동 배치됩니다. 공동 배치된 테이블은 collocation_id가 같습니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
              2 | public      | orders
              2 | public      | customers
              5 | public      | users
              5 | public      | followers
(4 rows)
```

limitless_table_collocation_distributions

이 뷰는 각 공동 배치의 키 분배를 보여줍니다.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
              2 |             6 | -9223372036854775808 | -4611686018427387904
              2 |             5 | -4611686018427387904 |                    0
              2 |             4 |                    0 |  4611686018427387904
              2 |             3 |  4611686018427387904 |  9223372036854775807
              5 |             6 | -9223372036854775808 | -4611686018427387904
              5 |             5 | -4611686018427387904 |                    0
              5 |             4 |                    0 |  4611686018427387904
              5 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

Aurora PostgreSQL Limitless Database의 대기 이벤트

Aurora PostgreSQL의 대기 이벤트는 입출력(I/O) 및 잠금과 같이 세션이 대기 중인 리소스를 나타냅니다. 대기 이벤트는 세션이 리소스를 기다리는 이유를 파악하고 병목 현상을 식별하는 데 유용합니다. 자세한 내용은 Aurora PostgreSQL 대기 이벤트 단원을 참조하십시오.

Aurora PostgreSQL Limitless Database에는 라우터 및 샤드와 관련된 자체 대기 이벤트가 있습니다. 이 중 대부분은 작업을 완료하기 위해 샤드에서 기다리는 라우터를 위한 것입니다. 샤드 대기 이벤트에는 수행 중인 작업에 대한 세부 정보가 포함됩니다.

주제

- 대기 이벤트 쿼리
- Limitless Database 대기 이벤트

대기 이벤트 쿼리

다음 예시와 같이 limitless_stat_activity 뷰를 사용하여 대기 이벤트를 쿼리할 수 있습니다.

```
SELECT wait_event FROM rds_aurora.limitless_stat_activity WHERE
 wait_event_type='AuroraLimitless';

      wait_event
----------------------
 RemoteStatementSetup
 RemoteStatementSetup
(2 rows)
```

다음 예시와 같이 aurora_stat_system_waits 함수를 사용하여 각 대기 이벤트의 대기 수와 소요된 총 시간을 나열할 수도 있습니다.

```
postgres_limitless=> SELECT type_name,event_name,waits,wait_time
    FROM aurora_stat_system_waits()
    NATURAL JOIN aurora_stat_wait_event()
    NATURAL JOIN aurora_stat_wait_type()
    WHERE type_name='AuroraLimitless'
    ORDER BY wait_time DESC;

    type_name    |        event_name          |  waits  |  wait_time
-----------------+----------------------------+---------+-------------
 AuroraLimitless | RemoteStatementSetup       |    7518 | 75236507897
```

```
 AuroraLimitless | RemoteStatementExecution  |      40 |      132986
 AuroraLimitless | Connect                   |       5 |        1453
(3 rows)
```

Limitless Database 대기 이벤트

다음 대기 이벤트는 Aurora PostgreSQL Limitless Database에 적용됩니다. 이러한 대기 이벤트를 모니터링하여 Aurora PostgreSQL Limitless Database 처리의 병목 현상을 식별할 수 있습니다.

주제

- AuroraLimitless:GlobalSequenceRefresh 대기 이벤트
- AuroraLimitless:GlobalVacuumTimeExchange 대기 이벤트
- AuroraLimitless:DistributedTransactionMonitorGather 대기 이벤트
- AuroraLimitlessActivity:AdminTaskSchedulerMain 대기 이벤트
- AuroraLimitlessActivity:AdminTaskExecutorMain 대기 이벤트
- AuroraLimitlessActivity:AdminTaskMonitorMain 대기 이벤트
- AuroraLimitlessActivity:DatabaseCleanupMonitorMain 대기 이벤트
- AuroraLimitlessActivity:TopologyCleanupMonitorMain 대기 이벤트
- AuroraLimitlessActivity:ToplogyChangeMonitorMain 대기 이벤트
- AuroraLimitlessActivity:DistributedTransactionMonitorMain 대기 이벤트
- AuroraLimitlessActivity:GlobalVacuumMonitorMain 대기 이벤트

IO:TwophaseFilePoolWrite 대기 이벤트

두 단계 상태 파일 풀 내에서 두 단계 상태 파일의 쓰기를 기다리는 중입니다. 이는 Aurora 전용 이벤트입니다.

원인

Limitless Database 분산 트랜잭션의 참가자를 포함하여 `PREPARED TRANSACTION` 명령을 실행하는 프로세스는 두 단계 파일에서 트랜잭션 상태를 유지해야 합니다. Aurora는 파일 풀을 사용하여 이 작업의 성능을 개선합니다.

작업

이 작업은 동기식 쓰기 I/O 작업이므로 이 이벤트에서 지연 시간이 높으면 `IO:XactSync`와 원인이 유사하며 동일한 방식으로 조사할 수 있습니다. Limitless Database를 사용하는 경우 실행 중인 분산 트랜잭션 수를 줄여야 할 수 있습니다.

IO:TwophaseFilePoolRead 대기 이벤트

두 단계 상태 파일 풀 내에서 두 단계 상태 파일의 읽기를 기다리는 중입니다.

원인

Limitless Database 분산 트랜잭션의 참가자를 포함하여 이전에 준비된 트랜잭션에 대해 `COMMIT PREPARED` 명령을 실행하는 프로세스는 두 단계 파일에서 이전에 지속된 트랜잭션 상태를 읽어야 할 수 있습니다. Aurora는 파일 풀을 사용하여 이 작업의 성능을 개선합니다.

작업

읽기 I/O 작업입니다. 따라서 이 이벤트에서 지연 시간이 높으면 `IO:DataFileRead`와 원인이 유사하며 동일한 방식으로 조사할 수 있습니다. Limitless Database를 사용하는 경우 실행 중인 분산 트랜잭션 수를 줄여야 할 수 있습니다.

AuroraLimitless:Connect 대기 이벤트

프로세스가 클러스터의 다른 노드에 대한 연결이 설정되기를 기다리고 있습니다.

원인

쿼리, 분산 트랜잭션을 실행하고 DDL을 수행하기 위해 프로세스와 원격 노드 간에 연결이 설정됩니다.

작업

클러스터에 대한 동시 연결 수를 줄이거나 교차 샤드 쿼리 사용을 조정합니다.

AuroraLimitless:AsyncConnect 대기 이벤트

이 이벤트는 `Connect`와 유사하지만 노드 세트에 대한 병렬 연결이 설정되기를 기다리는 프로세스를 나타냅니다.

원인

병렬 연결 설정은 DDL 문을 실행할 때 가장 일반적으로 수행됩니다.

작업

DDL 문 수를 줄이거나 동일한 세션에서 여러 DDL을 결합하여 연결 재사용을 개선합니다.

AuroraLimitless:RemoteStatementSetup 대기 이벤트

프로세스가 커서 열기, 닫기 또는 준비된 문 만들기와 같은 원격 쿼리 실행 설정을 기다리고 있습니다.

원인

이 대기 이벤트는 문을 단일 샤드 최적화할 수 없는 샤딩된 테이블의 스캔 수에 따라 증가합니다.

작업

쿼리를 최적화하여 스캔 작업 수를 줄이거나 단일 샤드 최적화에 대한 자격을 높입니다.

AuroraLimitless:RemoteDDLExecution 대기 이벤트

프로세스가 원격 DDL 명령이 완료되기를 기다리고 있습니다.

원인

DB 샤드 그룹에서 DDL 명령을 실행할 때는 작업을 확인하기 전에 다른 라우터 및 샤드 노드에 DDL 명령을 배포해야 합니다. 일부 DDL 작업은 스키마 변경에 맞게 데이터를 조정해야 하므로 장기간 실행될 수 있습니다.

작업

최적화를 위해 장기 실행 DDL 명령을 식별합니다.

AuroraLimitless:RemoteStatementExecution 대기 이벤트

프로세스가 원격 명령이 완료되기를 기다리고 있습니다.

원인

SQL 명령이 원격 노드에서 실행 중입니다. 이 이벤트는 `auto_analyze` 및 하트비트 검사와 같은 내부 통신에 자주 표시됩니다.

작업

limitless_stat_statements 뷰를 사용하여 장기 실행 명령을 식별합니다. 대부분의 경우 이는 특히 백그라운드 작업자 또는 내부 프로세스에서 예상되는 이벤트이며 조치가 필요하지 않습니다.

AuroraLimitless:FetchRemoteResults 대기 이벤트

프로세스가 원격 노드에서 행을 검색하기 위해 기다리고 있습니다.

원인

이 대기 이벤트는 샤딩된 테이블 또는 참조 테이블과 같은 원격 테이블에서 많은 수의 행을 가져올 때 증가할 수 있습니다.

작업

`limitless_stat_statements` 뷰를 사용하여 최적화되지 않은 SELECT 쿼리를 식별합니다. 필요한 데이터만 검색하도록 쿼리를 최적화합니다. `rds_aurora.limitless_maximum_adaptive_fetch_size` 파라미터를 조정할 수도 있습니다.

AuroraLimitless:AsyncGetInitialResponse 대기 이벤트

프로세스가 쿼리 실행에 파이프라인 모드가 사용될 때 초기 응답을 기다리고 있습니다.

원인

이는 일반적으로 단일 샤드 데이터 배치를 사용하는 쿼리에 대한 샤드 실행을 라우터 중에 볼 수 있으며 정상적인 실행에서 예상되는 부분입니다.

작업

추가 조치가 필요하지 않습니다.

AuroraLimitless:AsyncGetNextResponse 대기 이벤트

프로세스가 쿼리 실행에 파이프라인 모드를 사용할 때 추가 응답을 기다리고 있습니다.

원인

이는 일반적으로 단일 샤드 데이터 배치를 사용하는 쿼리에 대한 샤드 실행을 라우터 중에 볼 수 있으며 정상적인 실행에서 예상되는 부분입니다.

작업

추가 조치가 필요하지 않습니다.

AuroraLimitless:AbortedCommandCleanup 대기 이벤트

프로세스가 원격 정리 쿼리의 결과를 기다리고 있습니다. 정리 쿼리는 분산 트랜잭션이 종료될 때 적절한 상태로 반환하기 위해 샤드 노드에 발급됩니다.

원인

트랜잭션 정리는 오류가 발견되었거나 사용자가 명시적 ABORT 명령을 실행했거나 실행 중인 쿼리를 취소했기 때문에 트랜잭션이 중단될 때 수행됩니다.

작업

트랜잭션이 취소되는 원인을 조사합니다.

AuroraLimitless:DistributedCommitPrepare 대기 이벤트

프로세스가 분산 트랜잭션을 커밋하고 있으며 모든 참가자가 준비 명령을 승인하기를 기다리고 있습니다.

원인

여러 노드를 수정하는 트랜잭션은 분산 커밋을 수행해야 합니다. 참여 노드의 `IO:TwophaseFilePoolWrite` 이벤트에서 대기 시간이 길면 `DistributedCommitPrepare`의 대기 시간이 길어질 수 있습니다.

작업

여러 노드의 데이터를 수정하는 트랜잭션 수를 줄입니다. 클러스터의 다른 노드에서 `IO:TwophaseFilePoolWrite` 이벤트를 조사합니다.

AuroraLimitless:DistributedCommit 대기 이벤트

프로세스가 분산 트랜잭션을 커밋하고 있으며 리드 참가자가 커밋 명령을 승인하기를 기다리고 있습니다.

원인

여러 노드를 수정하는 트랜잭션은 분산 커밋을 수행해야 합니다. 리드 참가자의 `IO:XactSync` 이벤트에서 대기 시간이 길면 `DistributedCommit`의 대기 시간이 길어질 수 있습니다.

작업

여러 노드의 데이터를 수정하는 트랜잭션 수를 줄입니다. 클러스터의 다른 노드에서 `IO:XactSync` 이벤트를 조사합니다.

AuroraLimitless:DistributedCommitPrepareThrottle 대기 이벤트

프로세스가 분산 트랜잭션을 준비하려고 시도하고 있으며 기존 준비된 트랜잭션으로 인해 스로틀링되고 있습니다.

원인

여러 노드를 수정하는 트랜잭션은 분산 커밋을 수행해야 합니다. 이러한 트랜잭션의 참가자는 커밋 프로토콜의 일부로 준비 작업을 수행해야 합니다. Aurora는 동시 준비 수를 제한하며, 이 한도를 초과하면 프로세스가 `DistributedCommitPrepareThrottle` 이벤트에서 대기합니다.

작업

여러 노드의 데이터를 수정하는 트랜잭션 수를 줄입니다. `IO:TwophaseFilePoolWrite` 이벤트에서 시간이 길어지면 기존에 준비된 트랜잭션이 누적되어 새 준비 시도에 대한 스로틀링이 발생할 수 있으므로 해당 이벤트를 조사합니다.

AuroraLimitless:PreparedTransactionResolution 대기 이벤트

프로세스가 준비된 상태의 분산 트랜잭션에 의해 수정된 튜플을 발견했습니다. 프로세스는 분산 트랜잭션이 스냅샷에 표시되는지 확인해야 합니다.

원인

여러 노드를 수정하는 트랜잭션은 준비 단계가 포함된 분산 커밋을 수행해야 합니다. 분산 트랜잭션 수가 많거나 분산 커밋에서 지연 시간이 증가하면 다른 프로세스에서 `PreparedTransactionResolution` 대기 이벤트가 발생할 수 있습니다.

작업

여러 노드의 데이터를 수정하는 트랜잭션 수를 줄입니다. 분산 커밋 관련 이벤트의 시간이 증가하면 분산 트랜잭션의 커밋 경로에서 지연 시간이 증가할 수 있으므로 해당 이벤트를 조사합니다. 네트워크 및 CPU 로드를 조사할 수도 있습니다.

AuroraLimitless:SendPreparedTransactionOutcome 대기 이벤트

프로세스가 분산 트랜잭션을 조정하는 노드에서 실행 중이고 다른 프로세스가 해당 트랜잭션의 상태에 대해 문의했거나 프로세스가 분산 트랜잭션을 커밋하여 참가자에게 결과를 보내고 있습니다.

원인

`PreparedTransactionResolution` 대기 이벤트가 발생하는 프로세스는 트랜잭션 코디네이터를 쿼리합니다. 트랜잭션 코디네이터에 대한 응답은 SendPreparedTransactionOutcome이 발생합니다.

작업

여러 노드의 데이터를 수정하는 트랜잭션 수를 줄입니다. 분산 커밋 관련 이벤트와 `IO:TwophaseFilePoolWrite` 및 `IO:TwophaseFilePoolRead` 이벤트의 시간이 증가하면 분산 트랜잭션의 커밋 경로에서 지연 시간이 증가할 수 있으므로 해당 이벤트를 조사합니다. 네트워크 및 CPU 로드를 조사할 수도 있습니다.

AuroraLimitless:CommitClockBarrier 대기 이벤트

프로세스가 트랜잭션을 커밋하고 있으며 클러스터의 모든 노드에 대해 할당된 커밋 시간이 과거가 되도록 기다려야 합니다.

원인

CPU 또는 네트워크 포화로 인해 클럭 드리프트가 증가하여 이 대기 이벤트에서 소요된 시간이 늘어날 수 있습니다.

작업

클러스터에서 CPU 또는 네트워크 포화를 조사합니다.

AuroraLimitless:SnapshotClockBarrier 대기 이벤트

프로세스가 미래의 클럭이 있는 다른 노드에서 스냅샷 시간을 수신했으며 자체 클럭이 해당 시간에 도달하기를 기다리고 있습니다.

원인

이는 일반적으로 프로세스가 샤드로 푸시다운된 함수의 결과를 수신했고 노드 간에 클럭 드리프트가 있으면 발생합니다. CPU 또는 네트워크 포화로 인해 클럭 드리프트가 증가하여 이 대기 이벤트에서 소요된 시간이 늘어날 수 있습니다.

작업

클러스터에서 CPU 또는 네트워크 상태를 조사합니다.

AuroraLimitless:ReaderSnapshotClockBarrier 대기 이벤트

이 이벤트는 읽기 노드에서 발생합니다. 프로세스가 프로세스 스냅샷 시간이 적용되기 전에 발생한 모든 쓰기가 적용되도록 읽기 노드가 쓰기 스트림을 다시 재생하기를 기다리고 있습니다.

원인

Aurora 복제본 지연이 증가하면 이 이벤트에서 대기 시간이 늘어날 수 있습니다.

작업

Aurora 복제본 지연을 조사합니다.

AuroraLimitless:GatherDistributedDeadlockGraph 대기 이벤트

프로세스가 분산 교착 상태 감지의 일부로 잠금 그래프를 수집하기 위해 다른 노드와 통신하고 있습니다.

원인

프로세스가 잠금을 기다리고 있는 경우 `rds_aurora.limitless_distributed_deadlock_timeout`보다 오래 기다린 후 분산 교착 상태 확인을 수행합니다.

작업

애플리케이션에서 잠금 경합의 원인을 조사하고 `rds_aurora.limitless_distributed_deadlock_timeout` 조정을 고려합니다.

AuroraLimitless:DistributedDeadlockDetection 대기 이벤트

프로세스가 분산 교착 상태를 감지하기 위해 다른 노드와 통신하고 있습니다.

원인

프로세스가 잠금을 기다리고 있는 경우 `rds_aurora.limitless_distributed_deadlock_timeout`보다 오래 기다린 후 분산 교착 상태 확인을 수행합니다.

작업

애플리케이션에서 잠금 경합의 원인을 조사하고 `rds_aurora.limitless_distributed_deadlock_timeout` 조정을 고려합니다.

AuroraLimitless:DistributedDeadlockAbort 대기 이벤트

프로세스가 분산 교착 상태의 피해자로 선택된 세션을 중단하기 위해 다른 노드와 통신하고 있습니다.

원인

애플리케이션 패턴으로 인해 분산 교착 상태가 발생합니다.

작업

분산 교착 상태를 초래하는 애플리케이션 패턴을 조사합니다.

AuroraLimitless:GatherRemoteStats 대기 이벤트

프로세스가 클러스터의 다른 노드에서 통계를 수집하고 있습니다.

원인

`limitless_stat_activity`와 같은 모니터링 또는 활동 쿼리 및 뷰는 다른 노드에서 통계를 검색합니다.

작업

추가 조치가 필요하지 않습니다.

AuroraLimitless:GlobalSequenceRefresh 대기 이벤트

프로세스가 새 시퀀스 값을 생성하고 있으며 글로벌 시퀀스에서 새 청크를 요청해야 합니다.

원인

`rds_aurora.limitless_sequence_chunk_size`가 충분하지 않을 경우 시퀀스 값 생성 속도가 높아지면 이 이벤트가 중단될 수 있습니다.

작업

이는 정상적인 현상입니다. 이 이벤트에서 과도한 시간이 표시되면 `rds_aurora.limitless_sequence_chunk_size` 조정을 고려합니다. Limitless Database의 시퀀스에 대한 설명서를 참조하세요.

AuroraLimitless:GlobalVacuumTimeExchange 대기 이벤트

프로세스가 vacuum을 지원하기 위해 스냅샷 데이터를 교환하고 있습니다.

원인

Limitless Database의 노드는 vacuum 실행을 위한 올바른 컷오프 시간을 계산하기 위해 가장 오래된 활성 스냅샷 시간 데이터를 다른 노드와 교환합니다.

작업

추가 조치가 필요하지 않습니다.

AuroraLimitless:DistributedTransactionMonitorGather 대기 이벤트

프로세스가 분산 트랜잭션 정리를 지원하기 위해 다른 노드에서 트랜잭션 메타데이터를 수집하고 있습니다.

원인

Limitless Database의 노드는 분산 트랜잭션 상태를 제거할 수 있는 시기를 결정하기 위해 트랜잭션 메타데이터를 다른 노드와 교환합니다.

작업

추가 조치가 필요하지 않습니다.

AuroraLimitlessActivity:AdminTaskSchedulerMain 대기 이벤트

작업 스케줄러 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:AdminTaskExecutorMain 대기 이벤트

작업 실행기 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:AdminTaskMonitorMain 대기 이벤트

작업 모니터 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:DatabaseCleanupMonitorMain 대기 이벤트

데이터베이스 정리 모니터 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:TopologyCleanupMonitorMain 대기 이벤트

토폴로지 정리 모니터 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:ToplogyChangeMonitorMain 대기 이벤트

토폴로지 변경 모니터 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:DistributedTransactionMonitorMain 대기 이벤트

분산 트랜잭션 모니터 프로세스의 기본 루프에서 대기 중입니다.

AuroraLimitlessActivity:GlobalVacuumMonitorMain 대기 이벤트

글로벌 vacuum 모니터 프로세스의 기본 루프에서 대기 중입니다.

Aurora PostgreSQL Limitless Database 백업 및 복원

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터를 백업하고 복원할 수 있습니다.

목차

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 백업

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터를 백업하는 것과 표준 Aurora DB 클러스터를 백업하는 것은 기능 측면에서 유사점과 차이점이 있습니다.

- Limitless Database를 사용하는 Aurora DB 클러스터의 수동 DB 클러스터 스냅샷을 만들면 스냅샷에 DB 샤드 그룹의 데이터가 포함됩니다.
- 연속 백업에는 DB 샤드 그룹의 데이터가 포함됩니다.
- 자동 일일 스냅샷에는 DB 샤드 그룹의 데이터가 포함됩니다.
- DB 클러스터 스냅샷 복사가 지원됩니다. 자세한 내용은 DB 클러스터 스냅샷 복사 단원을 참조하십시오.
- DB 클러스터 스냅샷 공유가 지원됩니다. 자세한 내용은 DB 클러스터 스냅샷 공유 단원을 참조하십시오.
- `pg_dump` 또는 `pg_dumpall` 유틸리티를 사용하여 DB 샤드 그룹의 데이터베이스를 백업할 수 없습니다.
- DB 클러스터를 삭제할 때 최종 스냅샷을 만드는 것은 Aurora PostgreSQL Limitless Database에서 지원됩니다.
- DB 클러스터를 삭제할 때 자동 백업을 유지하는 것은 Aurora PostgreSQL Limitless Database에서 지원되지 않습니다.

DB 클러스터 스냅샷 생성

다음 AWS CLI 예시와 같이 표준 Aurora DB 클러스터와 동일한 방식으로 Aurora PostgreSQL Limitless Database DB 클러스터 스냅샷을 만듭니다.

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier my-db-cluster \
    --db-cluster-snapshot-identifier my-db-cluster-snapshot
```

DB 클러스터 백업에 대한 자세한 정보는 [Aurora DB 클러스터 백업 및 복원에 대한 개요](#) 섹션을 참조하세요.

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 복원

Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터를 복원하는 것과 표준 Aurora DB 클러스터를 복원하는 것은 기능 측면에서 유사점과 차이점이 있습니다.

- Limitless Database DB 클러스터는 `16.4-limitless`와 같이 Limitless Database와 호환되는 DB 엔진 버전을 사용하는 소스 DB 클러스터에서만 복원할 수 있습니다.
- Limitless Database를 사용하는 DB 클러스터의 수동 스냅샷에서 DB 클러스터를 복원하면 전체 DB 클러스터 스토리지가 복원됩니다. 여기에는 DB 샤드 그룹의 스토리지가 포함됩니다.

  Limitless Database의 스토리지에 액세스하려면 DB 샤드 그룹을 만들어야 합니다.
- 시점 복구(PITR)를 사용하여 보존 기간 내의 원하는 시점으로 DB 클러스터를 복원할 수 있습니다. 복원된 DB 클러스터에는 DB 샤드 그룹의 스토리지가 포함됩니다.

  Limitless Database의 스토리지에 액세스하려면 DB 샤드 그룹을 만들어야 합니다.
- 삭제된 Aurora PostgreSQL Limitless Database DB 클러스터에는 PITR이 지원되지 않습니다.
- 자동 일일 스냅샷에서 DB 클러스터를 복원하면 DB 샤드 그룹의 스토리지도 복원됩니다.
- Aurora PostgreSQL Limitless Database DB 클러스터를 복원할 때는 향상된 모니터링 및 성능 개선 도우미를 활성화해야 합니다. 성능 개선 도우미 KMS 키 ID를 포함해야 합니다.

Aurora PostgreSQL Limitless Database DB 클러스터를 복원한 후에는 쿼리를 실행하여 해당 클러스터의 기능을 확인해야 합니다.

DB 스냅샷에서 DB 클러스터 복원

다음 AWS CLI 예시에서는 DB 클러스터 스냅샷에서 Aurora PostgreSQL Limitless Database DB 클러스터를 복원하는 방법을 보여줍니다.

`16.4-limitless` DB 엔진 버전을 사용해야 합니다.

DB 클러스터 스냅샷에서 Limitless Database DB 클러스터를 복원하는 방법

1. DB 클러스터 복원:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-db-cluster \
    --snapshot-identifier my-db-cluster-snapshot \
    --engine aurora-postgresql \
    --engine-version 16.4-limitless \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. DB 샤드 그룹 만들기:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

자세한 내용은 기존 Aurora PostgreSQL Limitless Database DB 클러스터에 DB 샤드 그룹 추가 단원을 참조하십시오.

DB 클러스터 스냅샷에서 Aurora DB 클러스터를 복원하는 방법에 대한 자세한 내용은 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.

시점 복구를 사용하여 DB 클러스터 복원

다음 AWS CLI 예시에서는 시점 복구(PITR)를 사용하여 Aurora PostgreSQL Limitless Database DB 클러스터를 복원하는 방법을 보여줍니다.

PITR을 사용하여 Limitless Database DB 클러스터를 복원하는 방법

1. DB 클러스터 복원:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-db-cluster \
    --db-cluster-identifier my-new-db-cluster \
    --use-latest-restorable-time \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. DB 샤드 그룹 만들기:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

자세한 내용은 기존 Aurora PostgreSQL Limitless Database DB 클러스터에 DB 샤드 그룹 추가 단원을 참조하십시오.

PITR에 대한 자세한 내용은 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요.

PostgreSQL 백업 및 복원 유틸리티가 지원되지 않음

다음 PostgreSQL 유틸리티는 기본 DB 클러스터 또는 DB 샤드 그룹에서 지원되지 않습니다.

- `pg_dump`
- `pg_dumpall`
- `pg_restore`

오픈 소스 바이너리 또는 대체 방법을 통해 사용할 수 있지만 이렇게 하면 일관성 없는 결과가 나올 수 있습니다.

Amazon Aurora PostgreSQL Limitless Database 업그레이드

다음은 Aurora PostgreSQL Limitless Database 업그레이드에 적용됩니다.

- 마이너 버전 업그레이드가 지원됩니다.
- Aurora PostgreSQL Limitless Database 패치 적용이 지원됩니다. 패치는 유지 관리 기간 동안 적용할 보류 중인 유지 관리로 표시됩니다.

Amazon Aurora PostgreSQL Limitless Database를 사용하는 DB 클러스터 업그레이드

DB 클러스터를 수정하고 새 DB 엔진 버전을 선택하여 업그레이드합니다. AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

콘솔

Limitless Database DB 클러스터를 업그레이드하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스 페이지로 이동합니다.
3. 업그레이드하려는 Limitless Database DB 클러스터를 선택합니다.
4. 수정을 선택합니다.

   DB 클러스터 수정 페이지가 표시됩니다.
5. DB 엔진 버전에서 새로운 DB 엔진 버전을 선택합니다. 예를 들어 Limitless Database가 있는 Aurora PostgreSQL(PostgreSQL 16.6과 호환)을 선택합니다.
6. 계속을 선택합니다.
7. 요약 페이지에서 변경 사항을 즉시 적용할지 아니면 예약된 다음 유지 관리 기간 중에 적용할지 선택한 다음 클러스터 수정을 선택합니다.

CLI

다음 예시와 같이 modify-db-cluster AWS CLI 명령을 사용합니다.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-sv2-cluster \
    --engine-version 16.6-limitless \
    --apply-immediately
```

Aurora PostgreSQL Limitless Database 참조

Aurora PostgreSQL Limitless Database에 대해 다음 참조 주제를 제공합니다.

주제

지원되는/지원되지 않는 데이터 정의 언어(DDL) SQL 명령

다음 표에는 Aurora PostgreSQL Limitless Database에서 지원되는/지원되지 않는 DDL 명령과 제한 사항 또는 자세한 정보에 대한 참조가 나와 있습니다.

Command	지원?	제한 사항 또는 추가 정보
ALTER AGGREGATE	No	해당 사항 없음
ALTER COLLATION	예	없음
ALTER CONVERSION	예	없음
ALTER DATABASE	No	해당 사항 없음
ALTER DEFAULT PRIVILEGES	No	해당 사항 없음
ALTER DOMAIN	No	해당 사항 없음
ALTER EVENT TRIGGER	No	해당 사항 없음
ALTER EXTENSION	예	확장 프로그램
ALTER FOREIGN DATA WRAPPER	No	해당 사항 없음

Command	지원?	제한 사항 또는 추가 정보
ALTER FOREIGN TABLE	No	해당 사항 없음
ALTER FUNCTION	예	함수
ALTER GROUP	예	없음
ALTER INDEX	예	없음
ALTER LANGUAGE	No	해당 사항 없음
ALTER LARGE OBJECT	No	해당 사항 없음
ALTER MATERIALIZED VIEW	No	해당 사항 없음
ALTER OPERATOR	예	없음
ALTER OPERATOR CLASS	예	없음
ALTER OPERATOR FAMILY	예	없음
ALTER POLICY	No	해당 사항 없음
ALTER PROCEDURE	예	없음
ALTER PUBLICATION	No	해당 사항 없음
역할 변경	예	없음
ALTER ROUTINE	No	해당 사항 없음
ALTER RULE	No	해당 사항 없음
ALTER SCHEMA	예	없음
ALTER SEQUENCE	예	시퀀스
ALTER SERVER	No	해당 사항 없음
ALTER STATISTICS	No	해당 사항 없음

Command	지원?	제한 사항 또는 추가 정보
ALTER SUBSCRIPTION	No	해당 사항 없음
ALTER SYSTEM	No	해당 사항 없음
ALTER TABLE	예	ALTER TABLE
ALTER TABLESPACE	No	해당 사항 없음
ALTER TEXT SEARCH CONFIGURATION	No	해당 사항 없음
ALTER TEXT SEARCH DICTIONARY	No	해당 사항 없음
ALTER TEXT SEARCH PARSER	No	해당 사항 없음
ALTER TEXT SEARCH TEMPLATE	No	해당 사항 없음
ALTER TRIGGER	No	해당 사항 없음
ALTER TYPE	예	없음
ALTER USER	예	없음
ALTER USER MAPPING	No	해당 사항 없음
ALTER VIEW	예	없음
COMMENT	No	해당 사항 없음
CREATE ACCESS METHOD	No	해당 사항 없음
CREATE AGGREGATE	No	해당 사항 없음
CREATE CAST	예	없음
CREATE COLLATION	예	없음

Command	지원?	제한 사항 또는 추가 정보
CREATE CONVERSION	예	없음
데이터베이스 생성	예	데이터베이스 생성
CREATE DOMAIN	No	해당 사항 없음
CREATE EVENT TRIGGER	No	해당 사항 없음
CREATE EXTENSION	예	확장 프로그램
CREATE FOREIGN DATA WRAPPER	No	해당 사항 없음
CREATE FOREIGN TABLE	No	해당 사항 없음
CREATE FUNCTION	예	함수
create group	예	없음
CREATE INDEX	예	CREATE INDEX
CREATE LANGUAGE	No	해당 사항 없음
CREATE MATERIALIZED VIEW	No	해당 사항 없음
CREATE OPERATOR	예	없음
CREATE OPERATOR CLASS	예	없음
CREATE OPERATOR FAMILY	예	없음
CREATE POLICY	예	없음
CREATE PROCEDURE	예	없음
CREATE PUBLICATION	No	해당 사항 없음
역할 생성	예	없음

Command	지원?	제한 사항 또는 추가 정보
CREATE RULE	No	해당 사항 없음
CREATE SCHEMA	예	CREATE SCHEMA
CREATE ANY SEQUENCE	예	시퀀스
CREATE SERVER	No	해당 사항 없음
CREATE STATISTICS	No	해당 사항 없음
CREATE SUBSCRIPTION	No	해당 사항 없음
CREATE TABLE	예	CREATE TABLE
CREATE TABLE AS	예	CREATE TABLE AS
CREATE TABLESPACE	No	해당 사항 없음
CREATE TEMPORARY TABLE	No	해당 사항 없음
CREATE TEMPORARY TABLE AS	No	해당 사항 없음
CREATE TEXT SEARCH CONFIGURATION	No	해당 사항 없음
CREATE TEXT SEARCH DICTIONARY	No	해당 사항 없음
CREATE TEXT SEARCH PARSER	No	해당 사항 없음
CREATE TEXT SEARCH TEMPLATE	No	해당 사항 없음
CREATE TRANSFORM	No	해당 사항 없음
CREATE TRIGGER	No	해당 사항 없음

Command	지원?	제한 사항 또는 추가 정보
CREATE TYPE	예	없음
사용자 생성	예	없음
CREATE USER MAPPING	No	해당 사항 없음
CREATE VIEW	예	없음
DROP ACCESS METHOD	No	해당 사항 없음
DROP AGGREGATE	예	없음
DROP CAST	예	없음
DROP COLLATION	예	없음
DROP CONVERSION	예	없음
DROP DATABASE	예	DROP DATABASE
DROP DOMAIN	No	해당 사항 없음
DROP EVENT TRIGGER	No	해당 사항 없음
DROP EXTENSION	예	확장 프로그램
DROP FOREIGN DATA WRAPPER	No	해당 사항 없음
DROP FOREIGN TABLE	No	해당 사항 없음
DROP FUNCTION	예	함수
DROP GROUP	예	없음
DROP INDEX	예	없음
DROP LANGUAGE	No	해당 사항 없음
DROP MATERIALIZED VIEW	No	해당 사항 없음

Command	지원?	제한 사항 또는 추가 정보
DROP OPERATOR	예	없음
DROP OPERATOR CLASS	예	없음
DROP OPERATOR FAMILY	예	없음
DROP OWNED	No	해당 사항 없음
DROP POLICY	No	해당 사항 없음
DROP PROCEDURE	예	없음
DROP PUBLICATION	No	해당 사항 없음
DROP ROLE	예	없음
DROP ROUTINE	No	해당 사항 없음
DROP RULE	No	해당 사항 없음
DROP SCHEMA	예	없음
DROP SEQUENCE	예	없음
DROP SERVER	No	해당 사항 없음
DROP STATISTICS	No	해당 사항 없음
DROP SUBSCRIPTION	No	없음
DROP TABLE	예	없음
DROP TABLESPACE	No	해당 사항 없음
DROP TEXT SEARCH CONFIGURATION	No	해당 사항 없음
DROP TEXT SEARCH DICTIONARY	No	해당 사항 없음

Command	지원?	제한 사항 또는 추가 정보
DROP TEXT SEARCH PARSER	No	해당 사항 없음
DROP TEXT SEARCH TEMPLATE	No	해당 사항 없음
DROP TRANSFORM	No	해당 사항 없음
DROP TRIGGER	No	해당 사항 없음
DROP TYPE	예	없음
DROP USER	예	없음
DROP USER MAPPING	No	해당 사항 없음
DROP VIEW	예	없음
GRANT	예	없음
REASSIGN OWNED	No	해당 사항 없음
REVOKE	예	없음
SECURITY LABEL	No	해당 사항 없음
SELECT INTO	예	SELECT INTO
SET	예	없음
SET CONSTRAINTS	No	해당 사항 없음
SET ROLE	예	없음
SET SESSION AUTHORIZATION	예	없음
SET TRANSACTION	예	없음
TRUNCATE	예	없음

Aurora PostgreSQL Limitless Database에 대한 DDL 제한 사항 및 기타 정보

다음 주제에서는 Aurora PostgreSQL Limitless Database의 DDL SQL 명령에 대한 제한 사항을 설명하거나 자세한 정보를 제공합니다.

주제

ALTER TABLE

`ALTER TABLE` 명령은 일반적으로 Aurora PostgreSQL Limitless Database에서 지원됩니다. 자세한 내용은 PostgreSQL 설명서의 ALTER TABLE을 참조하세요.

제한 사항

`ALTER TABLE`에는 지원되는 옵션에 대한 다음과 같은 제한 사항이 있습니다.

열 제거

- 샤딩된 테이블에서는 샤드 키의 일부인 열을 제거할 수 없습니다.
- 참조 테이블에서는 프라이머리 키 열을 제거할 수 없습니다.

열의 데이터 형식 변경

- `USING` 표현식은 지원되지 않습니다.
- 샤딩된 테이블에서는 샤드 키의 일부인 열의 유형을 변경할 수 없습니다.

제약 추가 또는 제거

지원되지 않는 사항에 대한 자세한 내용은 제약 조건 섹션을 참조하세요.

열의 기본값 변경

기본값이 지원됩니다. 자세한 내용은 기본값 섹션을 참조하세요.

지원되지 않는 옵션

일부 옵션은 트리거와 같은 지원되지 않는 기능에 의존하므로 지원되지 않습니다.

`ALTER TABLE`에 대한 다음 테이블 수준 옵션은 지원되지 않습니다.

- `ALL IN TABLESPACE`
- `ATTACH PARTITION`
- `DETACH PARTITION`
- `ONLY` 플래그
- `RENAME CONSTRAINT`

`ALTER TABLE`에 대한 다음 열 수준 옵션은 지원되지 않습니다.

- ADD GENERATED

- DROP EXPRESSION [IF EXISTS]
- DROP IDENTITY [IF EXISTS]
- reset
- RESTART
- SET
- SET COMPRESSION
- SET STATISTICS

데이터베이스 생성

Aurora PostgreSQL Limitless Database에서는 무한 데이터베이스만 지원됩니다.

`CREATE DATABASE`가 실행되는 동안 데이터베이스 만들기는 비트랜잭션 작업이므로 하나 이상의 노드에서 성공적으로 만들어진 데이터베이스가 다른 노드에서 실패할 수 있습니다. 이 경우 DB 샤드 그룹 내의 일관성을 유지하기 위해 성공적으로 만들어진 데이터베이스 객체는 미리 정해진 시간 내에 모든 노드에서 자동으로 제거됩니다. 이 시간 동안 이름이 같은 데이터베이스를 다시 만들려면 데이터베이스가 이미 있음을 나타내는 오류가 발생할 수 있습니다.

지원되는 옵션은 다음과 같습니다.

- 데이터 정리:

```
CREATE DATABASE name WITH
    [LOCALE = locale]
    [LC_COLLATE = lc_collate]
    [LC_CTYPE = lc_ctype]
    [ICU_LOCALE = icu_locale]
    [ICU_RULES = icu_rules]
    [LOCALE_PROVIDER = locale_provider]
    [COLLATION_VERSION = collation_version];
```

- `CREATE DATABASE WITH OWNER`:

```
CREATE DATABASE name WITH OWNER = user_name;
```

다음 옵션은 JSON에서 지원되지 않습니다.

- `CREATE DATABASE WITH TABLESPACE`:

```
CREATE DATABASE name WITH TABLESPACE = tablespace_name;
```

- `CREATE DATABASE WITH TEMPLATE`:

```
CREATE DATABASE name WITH TEMPLATE = template;
```

CREATE INDEX

`CREATE INDEX CONCURRENTLY`는 샤딩된 테이블에 대해 지원됩니다.

```
CREATE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX`는 모든 테이블 유형에서 지원됩니다.

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX CONCURRENTLY`는 지원되지 않습니다.

```
CREATE UNIQUE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

자세한 내용은 UNIQUE를 참조하세요. 인덱스 만들기에 대한 일반적인 내용은 PostgreSQL 설명서에서 CREATE INDEX를 참조하세요.

인덱스 표시

`\d` *table_name* 또는 유사한 명령을 사용할 때 라우터에 모든 인덱스가 표시되는 것은 아닙니다. 대신 다음 예시와 같이 `pg_catalog.pg_indexes` 뷰를 사용하여 인덱스를 가져옵니다.

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items (id int PRIMARY KEY, val int);
CREATE INDEX items_my_index on items (id, val);

postgres_limitless=> SELECT * FROM pg_catalog.pg_indexes WHERE tablename='items';

 schemaname | tablename |    indexname     | tablespace |
   indexdef
------------+-----------+----------------+------------
+-----------------------------------------------------------------
```

```
 public      | items      | items_my_index |              | CREATE INDEX items_my_index
 ON ONLY public.items USING btree (id, val)
 public      | items      | items_pkey     |              | CREATE UNIQUE INDEX
 items_pkey ON ONLY public.items USING btree (id)
(2 rows)
```

CREATE SCHEMA

스키마 요소가 있는 `CREATE SCHEMA`는 지원되지 않습니다.

```
CREATE SCHEMA my_schema CREATE TABLE (column_name INT);
```

이것은 다음과 비슷한 오류를 생성합니다.

```
ERROR: CREATE SCHEMA with schema elements is not supported
```

CREATE TABLE

`CREATE TABLE` 문에서의 관계는 지원되지 않습니다. 예를 들면 다음과 같습니다.

```
CREATE TABLE orders (orderid int, customerId int, orderDate date) WITH
 (autovacuum_enabled = false);
```

`IDENTITY` 열은 지원되지 않습니다. 예를 들면 다음과 같습니다.

```
CREATE TABLE orders (orderid INT GENERATED ALWAYS AS IDENTITY);
```

Aurora PostgreSQL Limitless Database는 샤딩된 테이블 이름에 최대 54자를 지원합니다.

CREATE TABLE AS

`CREATE TABLE AS`를 사용하여 테이블을 만들려면 `rds_aurora.limitless_create_table_mode` 변수를 사용해야 합니다. 샤딩된 테이블의 경우 `rds_aurora.limitless_create_table_shard_key` 변수도 사용해야 합니다. 자세한 내용은 변수를 사용하여 무한 테이블 만들기 섹션을 참조하세요.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
```

```
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

CREATE TABLE ctas_table AS SELECT 1 a;

-- "source" is the source table whose columns and data types are used to create the new
 "ctas_table2" table.
CREATE TABLE ctas_table2 AS SELECT a,b FROM source;
```

참조 테이블에는 프라이머리 키 제약이 필요하므로 `CREATE TABLE AS`를 사용하여 참조 테이블을 만들 수 없습니다. `CREATE TABLE AS`는 프라이머리 키를 새 테이블에 전파하지 않습니다.

일반적인 내용은 PostgreSQL 설명서에서 [CREATE TABLE AS](https://www.postgresql.org/docs/current/sql-createtableas.html)를 참조하세요.

DROP DATABASE

만든 데이터베이스를 삭제할 수 있습니다.

`DROP DATABASE` 명령은 백그라운드에서 비동기식으로 실행됩니다. 실행 중일 때 동일한 이름으로 새 데이터베이스를 만들려고 하면 오류가 발생합니다.

SELECT INTO

`SELECT INTO`는 CREATE TABLE AS와 기능적으로 유사합니다. `rds_aurora.limitless_create_table_mode` 변수를 사용해야 합니다. 샤딩된 테이블의 경우 `rds_aurora.limitless_create_table_shard_key` 변수도 사용해야 합니다. 자세한 내용은 변수를 사용하여 무한 테이블 만들기 섹션을 참조하세요.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

-- "source" is the source table whose columns and data types are used to create the new
 "destination" table.
SELECT * INTO destination FROM source;
```

현재 `SELECT INTO` 작업은 샤드를 직접 통하지 않고 라우터를 통해 수행됩니다. 따라서 성능이 느릴 수 있습니다.

일반적인 내용은 PostgreSQL 설명서에서 SELECT INTO를 참조하세요.

제약 조건

Aurora PostgreSQL Limitless Database의 제약에는 다음 제한 사항이 적용됩니다.

CHECK

리터럴과 비교 연산자를 포함하는 간단한 제약이 지원됩니다. 다음 예시와 같이 함수 검증이 필요한 더 복잡한 표현식 및 제약은 지원되지 않습니다.

```
CREATE TABLE my_table (
    id  INT CHECK (id > 0)                                    -- supported
  , val INT CHECK (val > 0 AND val < 1000)                    -- supported
  , tag TEXT CHECK (length(tag) > 0)                          -- not supported:
 throws "Expression inside CHECK constraint is not supported"
  , op_date TIMESTAMP WITH TIME ZONE CHECK (op_date <= now())  -- not supported:
 throws "Expression inside CHECK constraint is not supported"
);
```

다음 예시와 같이 제약에 명시적 이름을 지정할 수 있습니다.

```
CREATE TABLE my_table (
    id  INT CONSTRAINT positive_id  CHECK (id > 0)
  , val INT CONSTRAINT val_in_range CHECK (val > 0 AND val < 1000)
);
```

다음 예시와 같이 테이블 수준 제약 구문을 CHECK 제약과 함께 사용할 수 있습니다.

```
CREATE TABLE my_table (
    id INT CONSTRAINT positive_id  CHECK (id > 0)
  , min_val INT CONSTRAINT min_val_in_range CHECK (min_val > 0 AND min_val < 1000)
  , max_val INT
  , CONSTRAINT max_val_in_range CHECK (max_val > 0 AND max_val < 1000 AND max_val >
 min_val)
);
```

EXCLUDE

Aurora PostgreSQL Limitless Database에서는 제외 제약이 지원되지 않습니다.

FOREIGN KEY

자세한 내용은 외래 키 섹션을 참조하세요.

Null이 아님

`NOT NULL` 제약은 제한 없이 지원됩니다.

PRIMARY KEY

프라이머리 키는 고유한 제약을 의미하므로 고유한 제약에 대한 제한이 프라이머리 키에도 적용됩니다. 이는 다음을 의미합니다.

- 테이블이 샤딩된 테이블로 변환되는 경우 샤드 키는 프라이머리 키의 하위 집합이어야 합니다. 즉, 프라이머리 키에는 샤드 키의 모든 열이 포함됩니다.
- 테이블이 참조 테이블로 변환되는 경우 프라이머리 키가 있어야 합니다.

다음 예시에서는 프라이머리 키의 사용을 보여줍니다.

```
-- Create a standard table.
CREATE TABLE public.my_table (
    item_id INT
  , location_code INT
  , val INT
  , comment text
);

-- Change the table to a sharded table using the 'item_id' and 'location_code'
 columns as shard keys.
CALL rds_aurora.limitless_alter_table_type_sharded('public.my_table',
 ARRAY['item_id', 'location_code']);
```

샤드 키가 포함되지 않은 프라이머리 키를 추가하려고 시도하는 경우:

```
-- Add column 'item_id' as the primary key.
-- Invalid because the primary key doesnt include all columns from the shard key:
-- 'location_code' is part of the shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR

-- add column "val" as primary key
-- Invalid because primary key does not include all columns from shard key:
--  item_id and location_code iare part of shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR
```

샤드 키가 포함된 프라이머리 키를 추가하려고 시도하는 경우:

```
-- Add the 'item_id' and 'location_code' columns as the primary key.
```

```
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code); -- OK

-- Add the 'item_id', 'location_code', and 'val' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code, val); -- OK
```

표준 테이블을 참조 테이블로 변경합니다.

```
-- Create a standard table.
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);

-- Convert the table to a reference table.
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcode');
```

샤딩된 테이블 및 참조 테이블 만들기에 대한 자세한 내용은 Aurora PostgreSQL Limitless Database 테이블 만들기 섹션을 참조하세요.

UNIQUE

샤딩된 테이블에서 고유 키에는 샤드 키가 포함되어야 합니다. 즉, 샤드 키는 고유 키의 하위 집합이어야 합니다. 테이블 유형을 샤딩됨으로 변경할 때 확인됩니다. 참조 테이블에는 제한이 없습니다.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT UNIQUE
);
```

다음 예시와 같이 테이블 수준 UNIQUE 제약이 지원됩니다.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT
  , CONSTRAINT zipcode_and_email UNIQUE (zipcode, email)
);
```

다음 예시에서는 프라이머리 키와 고유 키를 함께 사용하는 것을 보여줍니다. 두 키 모두 샤드 키를 포함해야 합니다.

```
SET rds_aurora.limitless_create_table_mode='sharded';
```

```
SET rds_aurora.limitless_create_table_shard_key='{"p_id"}';
CREATE TABLE t1 (
p_id BIGINT NOT NULL,
c_id BIGINT NOT NULL,
PRIMARY KEY (p_id),
UNIQUE (p_id, c_id)
);
```

자세한 내용은 PostgreSQL 설명서에서 Constraints를 참조하세요.

기본값

Aurora PostgreSQL Limitless Database는 기본값의 표현식을 지원합니다.

다음 예시는 기본값 사용을 보여줍니다.

```
CREATE TABLE t (
    a INT DEFAULT 5,
    b TEXT DEFAULT 'NAN',
    c NUMERIC
);

CALL rds_aurora.limitless_alter_table_type_sharded('t', ARRAY['a']);
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c
---+-----+---
 5 | NAN |
(1 row)
```

다음 예시와 같이 표현식이 지원됩니다.

```
CREATE TABLE t1 (a NUMERIC DEFAULT random());
```

다음 예시에서는 `NOT NULL`이고 기본값이 있는 새 열을 추가합니다.

```
ALTER TABLE t ADD COLUMN d BOOLEAN NOT NULL DEFAULT FALSE;
SELECT * FROM t;

 a |  b  | c | d
```

```
---+-----+---+---
 5 | NAN |   | f
(1 row)
```

다음 예시에서는 기존 열을 기본값으로 변경합니다.

```
ALTER TABLE t ALTER COLUMN c SET DEFAULT 0.0;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
(2 rows)
```

다음 예시에서는 기본값을 삭제합니다.

```
ALTER TABLE t ALTER COLUMN a DROP DEFAULT;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
   | NAN | 0.0 | f
(3 rows)
```

자세한 내용은 PostgreSQL 설명서에서 Default values를 참조하세요.

확장 프로그램

Aurora PostgreSQL Limitless Database에서 지원되는 PostgreSQL 확장은 다음과 같습니다.

- `aurora_limitless_fdw` - 이 확장은 사전 설치되어 있습니다. 삭제할 수 없습니다.
- `aws_s3` - 이 확장은 Aurora PostgreSQL에서와 유사한 방식으로 Aurora PostgreSQL Limitless Database에서 작동합니다.

  Amazon S3 버킷에서 Aurora PostgreSQL Limitless Database DB 클러스터로 데이터를 가져오거나 Aurora PostgreSQL Limitless Database DB 클러스터에서 Amazon S3 버킷으로 데이터를 내보낼 수 있습니다. 자세한 내용은 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터 및 Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기(을)를 참조하세요.
- `btree_gin`
- `citext`
- `ip4r`
- `pg_buffercache` - 이 확장은 Aurora PostgreSQL Limitless Database에서 커뮤니티 PostgreSQL과는 다르게 작동합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database의 pg_buffercache 차이점 섹션을 참조하세요.
- `pg_stat_statements`
- `pg_trgm`
- `pgcrypto`
- `pgstattuple` - 이 확장은 Aurora PostgreSQL Limitless Database에서 커뮤니티 PostgreSQL과는 다르게 작동합니다. 자세한 내용은 Aurora PostgreSQL Limitless Database의 pgstattuple 차이점 섹션을 참조하세요.
- `pgvector`
- `plpgsql` - 이 확장은 사전 설치되어 있지만 삭제할 수 있습니다.
- `PostGIS` - 긴 트랜잭션 및 테이블 관리 함수는 지원되지 않습니다. 공간 참조 테이블 수정은 지원되지 않습니다.
- `unaccent`
- `uuid`

현재 대부분의 PostgreSQL 확장은 Aurora PostgreSQL Limitless Database에서 지원되지 않습니다. 그러나 shared_preload_libraries(SPL) 구성 설정을 사용하여 Aurora PostgreSQL 기본 DB 클러스터로

확장을 로드할 수 있습니다. 확장은 Aurora PostgreSQL Limitless Database에도 로드되지만 제대로 작동하지 않을 수 있습니다.

예를 들어 pg_hint_plan 확장을 로드할 수 있지만 로드한다고 해서 쿼리 주석에 전달된 힌트가 사용되는 것은 아닙니다.

> **Note**
>
> pg_stat_statements 확장과 연결된 객체는 수정할 수 없습니다. pg_stat_statements 설치에 대한 자세한 내용은 limitless_stat_statements를 참조하세요.

pg_available_extensions 및 pg_available_extension_versions 함수를 사용하여 Aurora PostgreSQL Limitless Database에서 지원되는 확장을 찾을 수 있습니다.

다음 DDL은 확장에 지원됩니다.

CREATE EXTENSION

PostgreSQL에서와 같이 확장을 만들 수 있습니다.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]
```

자세한 내용은 PostgreSQL 설명서에서 CREATE EXTENSION을 참조하세요.

ALTER EXTENSION

다음 DDL이 지원됩니다.

```
ALTER EXTENSION name UPDATE [ TO new_version ]

ALTER EXTENSION name SET SCHEMA new_schema
```

자세한 내용은 PostgreSQL 설명서에서 ALTER EXTENSION을 참조하세요.

DROP EXTENSION

PostgreSQL에서와 같이 확장을 삭제할 수 있습니다.

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

자세한 내용은 PostgreSQL 설명서에서 DROP EXTENSION을 참조하세요.

다음 DDL은 확장에 지원되지 않습니다.

ALTER EXTENSION

확장에서 멤버 객체를 추가하거나 삭제할 수 없습니다.

```
ALTER EXTENSION name ADD member_object

ALTER EXTENSION name DROP member_object
```

Aurora PostgreSQL Limitless Database의 pg_buffercache 차이점

Aurora PostgreSQL Limitless Database에서 pg_buffercache 확장을 설치하고 `pg_buffercache` 뷰를 사용하면 현재 연결된 노드인 라우터에서 오는 버퍼 관련 정보만 수신합니다. 마찬가지로 `pg_buffercache_summary` 또는 `pg_buffercache_usage_counts` 함수를 사용하면 연결된 노드의 정보만 제공됩니다.

노드가 많을 수 있으며 문제를 효과적으로 진단하기 위해 원하는 노드에서 버퍼 정보에 액세스해야 할 수 있습니다. 따라서 Limitless Database는 다음 함수를 제공합니다.

- `rds_aurora.limitless_pg_buffercache(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_summary(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_usage_counts(subcluster_id)`

라우터든 샤드든 원하는 노드의 하위 클러스터 ID를 입력하면 해당 노드와 관련된 버퍼 정보에 쉽게 액세스할 수 있습니다. 이러한 함수는 무한 데이터베이스에 `pg_buffercache` 확장을 설치할 때 직접 사용할 수 있습니다.

> **Note**
>
> Aurora PostgreSQL Limitless Database는 `pg_buffercache` 확장 버전 1.4 이상에서 이러한 함수를 지원합니다.

limitless_pg_buffercache 뷰에 표시된 열은 pg_buffercache 뷰의 열과 약간 다릅니다.

- bufferid - pg_buffercache에서도 동일합니다.
- relname - pg_buffercache에서와 같이 파일 노드 번호를 표시하는 대신 limitless_pg_buffercache는 현재 데이터베이스 또는 공유 시스템 카탈로그에서 사용 가능한 경우 연결된 relname을, 사용 가능하지 않은 경우 NULL을 표시합니다.
- parent_relname - pg_buffercache에 없는 이 새 열에는 relname 열의 값이 파티셔닝된 테이블(샤딩된 테이블의 경우)을 나타내는 경우 상위 relname이 표시됩니다. 그렇지 않으면 NULL이 표시됩니다.
- spcname – pg_buffercache에서와 같이 테이블스페이스 객체 식별자(OID)가 표시되는 대신 limitless_pg_buffercache에서는 테이블스페이스 이름이 표시됩니다.
- datname - pg_buffercache에서와 같이 데이터베이스 OID가 표시되는 대신 limitless_pg_buffercache에서는 데이터베이스 이름이 표시됩니다.
- relforknumber - pg_buffercache에서도 동일합니다.
- relblocknumber - pg_buffercache에서도 동일합니다.
- isdirty - pg_buffercache에서도 동일합니다.
- usagecount - pg_buffercache에서도 동일합니다.
- pinning_backends - pg_buffercache에서도 동일합니다.

limitless_pg_buffercache_summary 및 limitless_pg_buffercache_usage_counts 뷰의 열은 각각 일반 pg_buffercache_summary 및 pg_buffercache_usage_counts 뷰의 열과 동일합니다.

이러한 함수를 사용하면 Limitless Database 환경의 모든 노드에서 자세한 버퍼 캐시 정보에 액세스하여 데이터베이스 시스템을 보다 효과적으로 진단하고 관리할 수 있습니다.

Aurora PostgreSQL Limitless Database의 pgstattuple 차이점

Aurora PostgreSQL에서 pgstattuple 확장은 현재 외부 테이블, 파티셔닝된 테이블 또는 파티셔닝된 인덱스를 지원하지 않습니다. 그러나 Aurora PostgreSQL Limitless Database에서는 사용자가 만든 객체가 이러한 지원되지 않는 유형 중 하나인 경우가 많습니다. 일반 테이블 및 인덱스(예: 카탈로그 테이블 및 해당 인덱스)가 있지만 대부분의 객체는 외부 노드에 상주하여 라우터의 외부 객체가 됩니다.

이 확장은 튜플 수준에서 통계를 얻는 데 중요하며, 이는 팽창 제거 및 진단 정보 수집과 같은 작업에 매우 중요합니다. 따라서 Aurora PostgreSQL Limitless Database는 무한 데이터베이스에서 pgstattuple 확장을 지원합니다.

Aurora PostgreSQL Limitless Database는 `rds_aurora` 스키마에 다음 함수를 포함합니다.

튜플 수준 통계 함수

rds_aurora.limitless_pgstattuple(*relation_name*)

- 용도: 표준 테이블 및 해당 인덱스에 대한 튜플 수준 통계 추출
- 입력: `relation_name`(텍스트) – 관계의 이름
- 출력: Aurora PostgreSQL의 `pgstattuple` 함수에서 반환된 열과 일치하는 열

rds_aurora.limitless_pgstattuple(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블, 카탈로그 테이블 및 해당 인덱스에 대한 튜플 수준 통계 추출
- 입력:
  - `relation_name`(텍스트) – 관계의 이름
  - `subcluster_id`(텍스트) - 통계를 추출할 노드의 하위 클러스터 ID
- 출력:
  - 참조 및 카탈로그 테이블(인덱스 포함)의 경우 열은 Aurora PostgreSQL의 열과 일치합니다.
  - 샤딩된 테이블의 경우 통계는 지정된 하위 클러스터에 있는 샤딩된 테이블의 파티션만 나타냅니다.

인덱스 통계 함수

rds_aurora.limitless_pgstatindex(*relation_name*)

- 용도: 표준 테이블의 B트리 인덱스에 대한 통계 추출
- 입력: `relation_name`(텍스트) – B트리 인덱스의 이름
- 출력: `root_block_no`를 제외한 모든 열이 반환됩니다. 반환된 열은 Aurora PostgreSQL의 `pgstatindex` 함수와 일치합니다.

rds_aurora.limitless_pgstatindex(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블 및 카탈로그 테이블의 B트리 인덱스에 대한 통계 추출
- 입력:
  - `relation_name`(텍스트) - B트리 인덱스의 이름
  - `subcluster_id`(텍스트) - 통계를 추출할 노드의 하위 클러스터 ID
- 출력:
  - 참조 및 카탈로그 테이블 인덱스의 경우 모든 열(`root_block_no` 제외)이 반환됩니다. 반환된 열은 Aurora PostgreSQL과 일치합니다.

  - 샤딩된 테이블의 경우 통계는 지정된 하위 클러스터에 있는 샤딩된 테이블 인덱스의 파티션만 나타냅니다. `tree_level` 열에는 요청된 하위 클러스터의 모든 테이블 조각에 대한 평균이 표시됩니다.

rds_aurora.limitless_pgstatginindex(*relation_name*)

- 용도: 표준 테이블의 Generalized Inverted Indexes(GIN)에 대한 통계 추출
- 입력: `relation_name`(텍스트) – GIN의 이름
- 출력: Aurora PostgreSQL의 `pgstatginindex` 함수에서 반환된 열과 일치하는 열

rds_aurora.limitless_pgstatginindex(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블 및 카탈로그 테이블의 GIN 인덱스에 대한 통계 추출
- 입력:
  - `relation_name`(텍스트) - 인덱스의 이름
  - `subcluster_id`(텍스트) - 통계를 추출할 노드의 하위 클러스터 ID
- 출력:
  - 참조 및 카탈로그 테이블 GIN 인덱스의 경우 열은 Aurora PostgreSQL의 열과 일치합니다.
  - 샤딩된 테이블의 경우 통계는 지정된 하위 클러스터에 있는 샤딩된 테이블 인덱스의 파티션만 나타냅니다.

rds_aurora.limitless_pgstathashindex(*relation_name*)

- 용도: 표준 테이블의 해시 인덱스에 대한 통계 추출
- 입력: `relation_name` (텍스트) – 해시 인덱스의 이름
- 출력: Aurora PostgreSQL의 `pgstathashindex` 함수에서 반환된 열과 일치하는 열

rds_aurora.limitless_pgstathashindex(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블 및 카탈로그 테이블의 해시 인덱스에 대한 통계 추출
- 입력:
  - `relation_name`(텍스트) - 인덱스의 이름
  - `subcluster_id`(텍스트) - 통계를 추출할 노드의 하위 클러스터 ID
- 출력:
  - 참조 및 카탈로그 테이블 해시 인덱스의 경우 열은 Aurora PostgreSQL의 열과 일치합니다.
  - 샤딩된 테이블의 경우 통계는 지정된 하위 클러스터에 있는 샤딩된 테이블 인덱스의 파티션만 나타냅니다.

페이지 수 함수

rds_aurora.limitless_pg_relpages(*relation_name*)

- 용도: 표준 테이블 및 해당 인덱스의 페이지 수 추출
- 입력: `relation_name`(텍스트) – 관계의 이름
- 출력: 지정된 관계의 페이지 수

rds_aurora.limitless_pg_relpages(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블 및 카탈로그 테이블(인덱스 포함)의 페이지 수 추출
- 입력:
  - `relation_name`(텍스트) – 관계의 이름
  - `subcluster_id`(텍스트) - 페이지 수를 추출할 노드의 하위 클러스터 ID
- 출력: 샤딩된 테이블의 경우 페이지 수는 지정된 하위 클러스터에 있는 모든 테이블 조각의 페이지 수 합계입니다.

대략적인 튜플 수준 통계 함수

rds_aurora.limitless_pgstattuple_approx(*relation_name*)

- 용도: 표준 테이블 및 해당 인덱스에 대한 대략적인 튜플 수준 통계 추출
- 입력: `relation_name`(텍스트) – 관계의 이름
- 출력: Aurora PostgreSQL의 pgstattuple_approx 함수에서 반환된 열과 일치하는 열

rds_aurora.limitless_pgstattuple_approx(*relation_name*, *subcluster_id*)

- 용도: 참조 테이블, 샤딩된 테이블 및 카탈로그 테이블(인덱스 포함)에 대한 대략적인 튜플 수준 통계 추출
- 입력:
  - `relation_name`(텍스트) – 관계의 이름
  - `subcluster_id`(텍스트) - 통계를 추출할 노드의 하위 클러스터 ID
- 출력:
  - 참조 및 카탈로그 테이블(인덱스 포함)의 경우 열은 Aurora PostgreSQL의 열과 일치합니다.
  - 샤딩된 테이블의 경우 통계는 지정된 하위 클러스터에 있는 샤딩된 테이블의 파티션만 나타냅니다.

> **Note**
>
> 현재 Aurora PostgreSQL Limitless Database는 구체화된 뷰, TOAST 테이블 또는 임시 테이블에서 `pgstattuple` 확장을 지원하지 않습니다.
> Aurora PostgreSQL은 다른 형식을 지원하지만 Aurora PostgreSQL Limitless Database에서는 입력을 텍스트로 제공해야 합니다.

외래 키

외래 키(`FOREIGN KEY`) 제약은 몇 가지 제한 사항과 함께 지원됩니다.

- `FOREIGN KEY`를 사용한 `CREATE TABLE`은 표준 테이블에만 지원됩니다. `FOREIGN KEY`를 사용하여 샤딩된 테이블 또는 참조 테이블을 만들려면 먼저 외래 키 제약 없이 테이블을 만듭니다. 그런 다음, 다음 문을 사용하여 수정합니다.

```
ALTER TABLE ADD CONSTRAINT;
```

- 테이블에 외래 키 제약이 있는 경우 표준 테이블을 샤딩된 테이블 또는 참조 테이블로 변환하는 것은 지원되지 않습니다. 제약을 삭제한 다음 변환 후 추가하세요.
- 외래 키 제약에 대한 테이블 유형에는 다음 제한 사항이 적용됩니다.
  - 표준 테이블에는 다른 표준 테이블에 대한 외래 키 제약이 있을 수 있습니다.
  - 상위 테이블과 하위 테이블이 공동 배치되고 외래 키가 샤드 키의 슈퍼 집합인 경우 샤딩된 테이블에 외래 키 제약이 있을 수 있습니다.
  - 샤딩된 테이블에는 참조 테이블에 대한 외래 키 제약이 있을 수 있습니다.
  - 참조 테이블에는 다른 참조 테이블에 대한 외래 키 제약이 있을 수 있습니다.

주제

- 외래 키 옵션
- 예시

외래 키 옵션

일부 DDL 옵션의 경우 Aurora PostgreSQL Limitless Database에서 외래 키가 지원됩니다. 다음 표에는 Aurora PostgreSQL Limitless Database 테이블 간에 지원되거나 지원되지 않는 옵션이 나열되어 있습니다.

DDL 옵션	참조에서 참조로	샤딩됨에서 샤딩됨으로(공동 배치됨)	참조에서 샤딩됨으로	표준에서 표준으로
`DEFERRABLE`	예	예	예	예

DDL 옵션	참조에서 참조로	샤딩됨에서 샤딩됨으로(공동 배치됨)	참조에서 샤딩됨으로	표준에서 표준으로
INITIALLY DEFERRED	예	예	예	예
INITIALLY IMMEDIATE	예	예	예	예
MATCH FULL	예	예	예	예
MATCH PARTIAL	아니요	아니요	아니요	아니요
MATCH SIMPLE	예	예	예	예
NOT DEFERRABLE	예	예	예	예
NOT VALID	예	아니요	아니요	예
ON DELETE CASCADE	예	예	예	예
ON DELETE NO ACTION	예	예	예	예
ON DELETE RESTRICT	예	예	예	예
ON DELETE SET DEFAULT	아니요	아니요	아니요	아니요
ON DELETE SET NULL	예	아니요	아니요	예
ON UPDATE CASCADE	아니요	아니요	아니요	예

DDL 옵션	참조에서 참조로	샤딩됨에서 샤딩됨으로(공동 배치됨)	참조에서 샤딩됨으로	표준에서 표준으로
ON UPDATE NO ACTION	예	예	예	예
ON UPDATE RESTRICT	예	예	예	예
ON UPDATE SET DEFAULT	아니요	아니요	아니요	아니요
ON UPDATE SET NULL	예	아니요	아니요	예

예시

- 표준에서 표준으로:

```
set rds_aurora.limitless_create_table_mode='standard';

CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

SELECT constraint_name, table_name, constraint_type
FROM information_schema.table_constraints WHERE constraint_type='FOREIGN KEY';

 constraint_name         | table_name  | constraint_type
-------------------------+-------------+-----------------
 orders_product_no_fkey  | orders      | FOREIGN KEY
```

```
(1 row)
```

- 샤딩됨에서 샤딩됨으로(공동 배치됨):

```
set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"product_no"}';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
set rds_aurora.limitless_create_table_collocate_with='products';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- 참조에서 샤딩됨으로:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- 참조에서 참조로:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

함수

함수는 Aurora PostgreSQL Limitless Database에서 지원됩니다.

함수에 지원되는 DDL은 다음과 같습니다.

CREATE FUNCTION

Aurora PostgreSQL에서와 마찬가지로 함수를 만들 수 있지만, 함수를 대체하는 동안 휘발성을 변경할 수는 없습니다.

자세한 내용은 PostgreSQL 설명서에서 CREATE FUNCTION을 참조하십시오.

ALTER FUNCTION

Aurora PostgreSQL에서와 마찬가지로 함수를 변경할 수 있지만, 휘발성 변경은 불가능합니다.

자세한 내용은 PostgreSQL 설명서에서 ALTER FUNCTION을 참조하세요.

DROP FUNCTION

Aurora PostgreSQL에서와 마찬가지로 함수를 삭제할 수 있습니다.

```
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
 [, ...]
    [ CASCADE | RESTRICT ]
```

자세한 내용은 PostgreSQL 설명서에서 DROP FUNCTION을 참조하세요.

주제

- 함수 배포
- 함수 휘발성

함수 배포

함수의 모든 문이 단일 샤드를 대상으로 하는 경우 전체 함수를 대상 샤드로 푸시하는 것이 좋습니다. 그러면 결과가 라우터 자체에서 함수를 풀지 않고 라우터로 다시 전파됩니다. 함수 및 저장 프로시저 푸시다운 기능은 데이터 소스, 즉 샤드에 더 가까이에서 함수 또는 저장 프로시저를 실행하려는 고객에게 유용합니다.

함수를 배포하려면 먼저 함수를 만든 다음 `rds_aurora.limitless_distribute_function` 프로시저를 직접 호출하여 함수를 배포합니다. 이 함수는 다음 구문을 사용합니다.

```
SELECT rds_aurora.limitless_distribute_function('function_prototype',
 ARRAY['shard_key'], 'collocating_table');
```

함수에 다음과 같은 파라미터가 있습니다.

- *function_prototype* - 배포할 함수입니다. 입력 인수만 언급하고 출력 인수는 언급하지 않습니다.

  OUT 파라미터로 정의된 인수가 있는 경우, function_prototype의 인수에 해당 유형을 포함하지 마세요.
- ARRAY['*shard_key*'] - 함수의 샤드 키로 식별되는 함수 인수 목록입니다.
- *collocating_table* - 대상 샤드의 데이터 범위를 포함하는 샤딩된 테이블입니다.

실행을 위해 이 함수를 푸시다운할 샤드를 식별하기 위해 시스템은 ARRAY['*shard_key*'] 인수를 가져와 해시하고 이 해시 값이 포함된 범위를 호스팅하는 *collocating_table*에서 샤드를 찾습니다.

제한 사항

함수 또는 프로시저를 배포하면 해당 샤드의 샤드 키 범위에 의해 경계가 지정된 데이터만 처리합니다. 함수 또는 프로시저가 다른 샤드의 데이터에 액세스하려고 하는 경우 배포된 함수 또는 프로시저에서 반환되는 결과는 배포되지 않은 것의 결과와 다릅니다.

예를 들어, 여러 샤드를 터치하는 쿼리가 포함된 함수를 만든 다음 rds_aurora.limitless_distribute_function 프로시저를 직접 호출하여 배포합니다. 샤드 키에 대한 인수를 제공하여 이 함수를 간접 호출하면 해당 함수를 실행한 결과가 해당 샤드에 있는 값에 의해 경계가 지정될 수 있습니다. 이러한 결과는 함수를 배포하지 않고 생성된 결과와 다릅니다.

예시

다음과 같이 customer_id 샤드 키가 포함된 샤딩된 테이블 customers가 있는 func 함수를 생각해 보세요.

```
postgres_limitless=> CREATE OR REPLACE FUNCTION func(c_id integer, sc integer)
 RETURNS int
    language SQL
    volatile
    AS $$
    UPDATE customers SET score = sc WHERE customer_id = c_id RETURNING score;
    $$;
```

이제 이 함수를 배포합니다.

```
SELECT rds_aurora.limitless_distribute_function('func(integer, integer)',
 ARRAY['c_id'], 'customers');
```

다음은 예시 쿼리 계획입니다.

```
EXPLAIN(costs false, verbose true) SELECT func(27+1,10);

                   QUERY PLAN
 --------------------------------------------------
  Foreign Scan
    Output: (func((27 + 1), 10))
    Remote SQL:  SELECT func((27 + 1), 10) AS func
  Single Shard Optimized
 (4 rows)
```

```
EXPLAIN(costs false, verbose true)
 SELECT * FROM customers,func(customer_id, score) WHERE customer_id=10 AND score=27;

                         QUERY PLAN
 ---------------------------------------------------------------------
  Foreign Scan
    Output: customer_id, name, score, func
    Remote SQL:  SELECT customers.customer_id,
      customers.name,
      customers.score,
      func.func
     FROM public.customers,
      LATERAL func(customers.customer_id, customers.score) func(func)
    WHERE ((customers.customer_id = 10) AND (customers.score = 27))
  Single Shard Optimized
 (10 rows)
```

다음 예시에서는 IN 및 OUT 파라미터가 인수인 프로시저를 보여줍니다.

```
CREATE OR REPLACE FUNCTION get_data(OUT id INTEGER, IN arg_id INT)
    AS $$
    BEGIN
        SELECT customer_id,
        INTO id
        FROM customer
```

```
        WHERE customer_id = arg_id;
    END;
    $$ LANGUAGE plpgsql;
```

다음 예시에서는 IN 파라미터만 사용하여 프로시저를 배포합니다.

```
EXPLAIN(costs false, verbose true) SELECT * FROM get_data(1);

             QUERY PLAN
 -----------------------------------
  Foreign Scan
    Output: id
    Remote SQL:  SELECT customer_id
     FROM get_data(1) get_data(id)
  Single Shard Optimized
 (6 rows)
```

함수 휘발성

pg_proc 뷰에서 provolatile 값을 확인하여 함수가 변경 불가능한지, 안정적인지, 휘발성인지 확인할 수 있습니다. provolatile 값은 함수의 결과가 입력 인수에만 의존하는지 아니면 외부 요인의 영향을 받는지를 나타냅니다.

값은 다음 중 하나입니다.

- i - 동일한 입력에 대해 항상 동일한 결과를 제공하는 변경 불가능한 함수
- s - 스캔 내에서 결과(고정 입력의 경우)가 변경되지 않는 안정적인 함수
- v - 결과가 언제든지 변경될 수 있는 휘발성 함수 또한 부작용이 있는 함수에 v를 사용하여 이 함수에 대한 직접 호출이 최적화로 인해 제거되지 않도록 합니다.

다음 예시에서는 휘발성 함수를 보여줍니다.

```
SELECT proname, provolatile FROM pg_proc WHERE proname='pg_sleep';

 proname  | provolatile
----------+-------------
 pg_sleep | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='uuid_generate_v4';
```

```
     proname      | provolatile
------------------+-------------
 uuid_generate_v4 | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='nextval';

 proname | provolatile
---------+-------------
 nextval | v
(1 row)
```

Aurora PostgreSQL Limitless Database에서는 기존 함수의 휘발성 변경이 지원되지 않습니다. 이는 다음 예시와 같이 `ALTER FUNCTION` 및 `CREATE OR REPLACE FUNCTION` 명령 모두에 적용됩니다.

```
-- Create an immutable function
CREATE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$IMMUTABLE;

-- Altering the volatility throws an error
ALTER FUNCTION immutable_func1 STABLE;

-- Replacing the function with altered volatility throws an error
CREATE OR REPLACE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$VOLATILE;
```

함수에 올바른 휘발성을 할당하는 것이 좋습니다. 예를 들어 함수가 여러 테이블에서 `SELECT`를 사용하거나 데이터베이스 객체를 참조하는 경우 `IMMUTABLE`로 설정하지 마세요. 테이블 내용이 변경되면 변경 불가능성이 중단됩니다.

Aurora PostgreSQL은 변경 불가능한 함수 내부에 `SELECT`를 허용하지만 결과가 올바르지 않을 수 있습니다. Aurora PostgreSQL Limitless Database는 오류와 잘못된 결과를 모두 반환할 수 있습니다. 함수 휘발성에 대한 자세한 내용은 PostgreSQL 설명서에서 [Function volatility categories]를 참조하세요.

시퀀스

명명된 시퀀스는 오름차순 또는 내림차순으로 고유 번호를 생성하는 데이터베이스 객체입니다. `CREATE SEQUENCE`는 새 시퀀스 번호 생성기를 만듭니다. 시퀀스 값은 고유하도록 보장됩니다.

Aurora PostgreSQL Limitless Database에서 명명된 시퀀스를 만들면 배포된 시퀀스 객체가 만들어집니다. 그런 다음 Aurora PostgreSQL Limitless Database는 모든 Distributed Transaction Routers(라우터)에 중첩되지 않는 시퀀스 값 청크를 배포합니다. 청크는 라우터에서 로컬 시퀀스 객체로 표시되므로 `nextval` 및 `currval`과 같은 시퀀스 작업은 로컬에서 실행됩니다. 라우터는 독립적으로 작동하며 필요한 경우 배포된 시퀀스에서 새 청크를 요청합니다.

시퀀스에 대한 자세한 내용은 PostgreSQL 설명서에서 [CREATE SEQUENCE](#)를 참조하세요.

주제

새 청크 요청

`rds_aurora.limitless_sequence_chunk_size` 파라미터를 사용하여 라우터에 할당된 청크의 크기를 구성합니다. 기본값은 `250000`입니다. 각 라우터는 처음에 활성 청크와 예약된 청크라는 두 개의 청크를 소유합니다. 활성 청크는 로컬 시퀀스 객체(`minvalue` 및 `maxvalue` 설정)를 구성하는 데 사용되며 예약된 청크는 내부 카탈로그 테이블에 저장됩니다. 활성 청크가 최솟값 또는 최댓값에 도달하면 예약된 청크로 바뀝니다. 이를 위해 `ALTER SEQUENCE`는 내부적으로 사용되며, 즉 `AccessExclusiveLock`이 획득됩니다.

백그라운드 작업자는 라우터 노드에서 10초마다 실행하여 사용된 예약된 청크의 시퀀스를 스캔합니다. 사용된 청크가 발견되면 작업자는 배포된 시퀀스에서 새 청크를 요청합니다. 백그라운드 작업자가 새 청크를 요청할 시간이 있을 만큼 청크 크기를 충분히 크게 설정해야 합니다. 사용자 세션의 컨텍스트에서는 원격 요청이 절대 발생하지 않으므로 새 시퀀스를 직접 요청할 수 없습니다.

제한 사항

Aurora PostgreSQL Limitless Database의 시퀀스에는 다음 제한 사항이 적용됩니다.

- pg_sequence 카탈로그, pg_sequences 함수 및 SELECT * FROM *sequence_name* 문은 모두 배포된 상태가 아닌 로컬 시퀀스 상태만 표시합니다.
- 시퀀스 값은 고유하도록 보장되고 세션 내에서 변화가 없도록 보장됩니다. 그러나 다른 세션이 다른 라우터에 연결된 경우 다른 세션에서 nextval 문이 실행되면 순서가 잘못될 수 있습니다.
- 시퀀스 크기(사용 가능한 값 수)가 모든 라우터에 배포될 수 있을 만큼 충분히 큰지 확인하세요. chunk_size를 구성하려면 rds_aurora.limitless_sequence_chunk_size 파라미터를 사용합니다. (각 라우터에는 두 개의 청크가 있습니다.)
- CACHE 옵션이 지원되지만 캐시는 chunk_size보다 작아야 합니다.

지원되지 않는 옵션

Aurora PostgreSQL Limitless Database의 시퀀스에는 다음 옵션이 지원되지 않습니다.

시퀀스 조작 함수

setval 함수는 지원되지 않습니다. 자세한 내용은 PostgreSQL 설명서에서 Sequence Manipulation Functions을 참조하세요.

CREATE ANY SEQUENCE

다음 옵션은 지원되지 않습니다.

```
CREATE [{ TEMPORARY | TEMP} | UNLOGGED] SEQUENCE
    [[ NO ] CYCLE]
```

자세한 내용은 PostgreSQL 설명서에서 CREATE SEQUENCE를 참조하세요.

ALTER SEQUENCE

다음 옵션은 지원되지 않습니다.

```
ALTER SEQUENCE
    [[ NO ] CYCLE]
```

자세한 내용은 PostgreSQL 설명서에서 ALTER SEQUENCE를 참조하세요.

ALTER TABLE

ALTER TABLE 명령은 시퀀스에서 지원되지 않습니다.

예시

CREATE/DROP SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
       1
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16960 |       20 |        1 |            1 |  10000 |      1 |        1 | f
(1 row)

% connect to another router
postgres_limitless=> SELECT nextval('s');

 nextval
---------
   10001
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16959 |       20 |    10001 |            1 |  20000 |  10001 |        1 | f
(1 row)

postgres_limitless=> DROP SEQUENCE s;
DROP SEQUENCE
```

ALTER SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> ALTER SEQUENCE s RESTART 500;
ALTER SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
     500
(1 row)

postgres_limitless=> SELECT currval('s');

 currval
---------
     500
(1 row)
```

시퀀스 조작 함수

```
postgres=# CREATE TABLE t(a bigint primary key, b bigint);
CREATE TABLE

postgres=# CREATE SEQUENCE s minvalue 0 START 0;
CREATE SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));
INSERT 0 1

postgres=# SELECT * FROM t;

 a | b
---+---
 0 | 0
 1 | 1
```

```
(2 rows)

postgres=# ALTER SEQUENCE s RESTART 10000;
ALTER SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# SELECT * FROM t;

   a   |   b
-------+-------
     0 |     0
     1 |     1
 10000 | 10000
(3 rows)
```

시퀀스 뷰

Aurora PostgreSQL Limitless Database는 시퀀스에 대해 다음 뷰를 제공합니다.

rds_aurora.limitless_distributed_sequence

이 뷰는 배포된 시퀀스 상태 및 구성을 보여줍니다. minvalue, maxvalue, start, inc 및 cache 열은 pg_sequences 뷰와 동일한 의미를 가지며 시퀀스가 만들어질 때 사용된 옵션을 보여줍니다. lastval 열에는 배포된 시퀀스 객체에서 가장 최근에 할당되거나 예약된 값이 표시됩니다. 라우터가 시퀀스 청크를 로컬로 유지하기 때문에 값이 이미 사용되었다는 의미는 아닙니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_distributed_sequence WHERE
 sequence_name='test_serial_b_seq';

 schema_name |    sequence_name    | lastval | minvalue |  maxvalue   | start | inc |
 cache
-------------+-------------------+---------+----------+------------+-------+-----
+-------
 public      | test_serial_b_seq | 1250000 |        1 | 2147483647 |     1 |   1 |
   1
(1 row)
```

rds_aurora.limitless_sequence_metadata

이 뷰는 배포된 시퀀스 메타데이터를 표시하고 클러스터 노드에서 시퀀스 메타데이터를 집계합니다. 이는 다음 열을 사용합니다.

- subcluster_id - 청크를 소유한 클러스터 노드 ID입니다.
- 활성 청크 - 사용 중인 시퀀스의 청크입니다(active_minvalue, active_maxvalue).
- 예약된 청크 - 다음에 사용할 로컬 청크입니다(reserved_minvalue, reserved_maxvalue).
- local_last_value - 로컬 시퀀스에서 마지막으로 관찰된 값입니다.
- chunk_size - 만들 때 구성된 청크의 크기입니다.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_sequence_metadata WHERE
 sequence_name='test_serial_b_seq' order by subcluster_id;

 subcluster_id |    sequence_name    | schema_name | active_minvalue | active_maxvalue
 | reserved_minvalue | reserved_maxvalue | chunk_size | chunk_state |
 local_last_value
---------------+-------------------+-------------+-----------------
+-----------------+-------------------+-------------------+------------
+-------------+------------------
```

```
 1              | test_serial_b_seq | public      |          500001 |          750000
 |          1000001 |           1250000 |     250000 |           1 |
 550010
 2              | test_serial_b_seq | public      |          250001 |          500000
 |           750001 |           1000000 |     250000 |           1 |

(2 rows)
```

시퀀스 문제 해결

시퀀스에서 다음과 같은 문제가 발생할 수 있습니다.

청크 크기가 충분히 크지 않음

청크 크기가 충분히 크지 않고 트랜잭션 속도가 높으면 활성 청크가 소진되기 전에 백그라운드 작업자가 새 청크를 요청할 시간이 충분하지 않을 수 있습니다. 이로 인해 경합이 발생하고 `LIMITLESS:AuroraLimitlessSequenceReplace`, `LWLock:LockManager`, `Lockrelation`, `LWlock:bufferscontent` 등의 대기 이벤트가 발생할 수 있습니다.

`rds_aurora.limitless_sequence_chunk_size` 파라미터의 값을 늘립니다.

시퀀스 캐시가 너무 높게 설정됨

PostgreSQL에서는 세션 수준에서 시퀀스 캐싱이 수행됩니다. 각 세션은 시퀀스 객체에 대한 한 번의 액세스 중에 연속 시퀀스 값을 할당하고 그에 따라 시퀀스 객체의 `last_value`를 늘립니다. 그런 다음 해당 세션 내에서 다음번에 `nextval`을 사용하면 시퀀스 객체를 터치하지 않고 미리 할당된 값을 반환하기만 합니다.

세션 내에서 할당되었지만 사용되지 않은 모든 숫자는 세션이 종료될 때 손실되어 시퀀스에서 '구멍'이 됩니다. 이로 인해 sequence_chunk가 빠르게 소비되어 경합이 발생하고 `LIMITLESS:AuroraLimitlessSequenceReplace`, `LWLock:LockManager`, `Lockrelation`, `LWlock:bufferscontent` 등의 대기 이벤트가 발생할 수 있습니다.

시퀀스 캐시 설정을 줄이세요.

다음 그림은 시퀀스 문제로 인한 대기 이벤트를 보여줍니다.


Database load Info
Viewed as Absolute
Sliced by Waits
Bar Line Table
Show estimated vCPU
Load is measured in average active sessions (AAS). The vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance.
Average active sessions (AAS)
2,000
1,500
1,000
500
0
05:42:55 05:43 05:43:05 05:43:10 05:43:15 05:43:20 05:43:25 05:43:30 05:43:35 05:43:40 05:43:45 05:43:50 05:43:55 05:44 05:44:05 05:44:10 05:44:15 05:44:20
IPC:ProcArrayGroupUpdate
LIMITLESS:LimitlessConnect
LIMITLESS:LimitlessExecute
IO:XactSync
LIMITLESS:LimitlessModify
Lock:relation
LWLock:LockManager
LIMITLESS:AuroraLimitlessSequenceRe
LWLock:BufferContent
CPU
Dimensions
Load distribution
Top instances
Top waits
Top SQL
Top hosts
Top users
Top session types
Top applications
Top databases
Top SQL (25)
Find SQL statements
1 2 3
Load by waits (AAS)
SQL statements
441.03
select nextval(?) as order_id
2.50
INSERT INTO ec_sample.orderdetails (customer_id,order_id,orderline_id,product_id...
1.39
INSERT INTO ec_sample.orders (customer_id,order_id,order_date,order_status,updat...

지원되는/지원되지 않는 데이터 조작 언어(DML) 및 쿼리 처리 SQL 명령

다음 표에는 Aurora PostgreSQL Limitless Database에서 지원되는/지원되지 않는 DML 명령과 제한 사항 또는 자세한 정보에 대한 참조가 나와 있습니다.

Command	지원?	제한 사항 또는 추가 정보
ABORT	예	없음
ANALYZE	예	ANALYZE
BEGIN	예	없음
CALL	예	없음
CHECKPOINT	예	없음
CLOSE	예	없음
CLUSTER	예	CLUSTER
COMMIT	예	없음
COMMIT PREPARED	No	해당 사항 없음
COPY	예	없음
DEALLOCATE	예	없음
DECLARE	예	없음
DELETE	예	없음
DISCARD	예	없음
DO	예	없음
END	예	없음
EXECUTE	예	없음
EXPLAIN	예	EXPLAIN

Command	지원?	제한 사항 또는 추가 정보
FETCH	예	없음
IMPORT FOREIGN SCHEMA	No	해당 사항 없음
INSERT	예	INSERT
LISTEN	No	해당 사항 없음
LOCK	예	없음
MERGE	No	해당 사항 없음
MOVE	예	없음
NOTIFY	No	해당 사항 없음
OPEN	예	없음
PREPARE	예	없음
PREPARE TRANSACTION	No	해당 사항 없음
REFRESH MATERIALIZED VIEW	No	해당 사항 없음
REINDEX	No	해당 사항 없음
RELEASE SAVEPOINT	예	없음
ROLLBACK	예	없음
ROLLBACK PREPARED	No	해당 사항 없음
ROLLBACK TO SAVEPOINT	예	없음
SAVEPOINT	예	없음
SELECT	예	없음
SELECT INTO	예	없음

Command	지원?	제한 사항 또는 추가 정보
SET	예	없음
START TRANSACTION	예	없음
UNLISTEN	No	없음
UPDATE	예	UPDATE
UPDATE … WHERE CURRENT OF	No	해당 사항 없음
VACUUM	예	VACUUM
VALUES	예	없음

Aurora PostgreSQL Limitless Database에 대한 DML 제한 사항 및 기타 정보

다음 주제에서는 Aurora PostgreSQL Limitless Database의 DML 및 쿼리 처리 SQL 명령에 대한 제한 사항을 설명하거나 자세한 정보를 제공합니다.

주제

ANALYZE

ANALYZE 명령은 데이터베이스의 테이블 내용에 대한 통계를 수집합니다. 이후 쿼리 플래너는 이러한 통계를 사용하여 쿼리에 대한 가장 효율적인 실행 계획을 결정하는 데 도움을 줍니다. 자세한 내용은 PostgreSQL 설명서의 ANALYZE를 참조하세요.

Aurora PostgreSQL Limitless Database에서 ANALYZE 명령은 실행 시 모든 라우터 및 샤드에 대한 테이블 통계를 수집합니다.

ANALYZE 실행 중 모든 라우터에 대한 통계 계산을 방지하기 위해 테이블 통계는 라우터 중 하나에 대해 계산된 다음 피어 라우터에 복사됩니다.

CLUSTER

CLUSTER 명령은 인덱스를 기반으로 테이블을 물리적으로 재정렬합니다. 인덱스가 테이블에 이미 정의되어 있어야 합니다. Aurora PostgreSQL Limitless Database에서 클러스터링은 각 샤드에 있는 인덱스의 일부와 같은 위치에 있습니다(로컬).

자세한 내용은 PostgreSQL 설명서의 CLUSTER를 참조하세요.

EXPLAIN

다음 파라미터를 사용하여 EXPLAIN 명령의 출력을 구성합니다.

- rds_aurora.limitless_explain_options - EXPLAIN 출력에 포함할 항목입니다. 기본값은 single_shard_optimization입니다. 계획이 단일 샤드에 최적화되어 있는지가 표시되지만 샤드 계획은 포함되지 않습니다.

이 예시에서는 EXPLAIN 출력에 샤드의 계획이 표시되지 않습니다.

```
postgres_limitless=> EXPLAIN SELECT * FROM employees where id =25;

                      QUERY PLAN
------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
 Single Shard Optimized
(2 rows)
```

이제 shard_plans 및 single_shard_optimization을 포함하도록 rds_aurora.limitless_explain_options을 설정합니다. 라우터와 샤드 모두에서 문 실행 계획을 볼 수 있습니다. 또한 enable_seqscan 파라미터를 비활성화하여 인덱스 스캔이 샤드 계층에 사용되도록 합니다.

```
postgres_limitless=> SET rds_aurora.limitless_explain_options = shard_plans,
 single_shard_optimization;
SET

postgres_limitless=> SET enable_seqscan = OFF;
SET

postgres_limitless=> EXPLAIN SELECT * FROM employees WHERE id = 25;

                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Remote Plans from Shard postgres_s4:
         Index Scan using employees_ts00287_id_idx on employees_ts00287
 employees_fs00003  (cost=0.14..8.16 rows=1 width=15)
           Index Cond: (id = 25)
 Single Shard Optimized
(5 rows)
```

EXPLAIN 명령에 대한 자세한 내용은 PostgreSQL 설명서의 EXPLAIN을 참조하세요.

INSERT

대부분의 INSERT 명령은 Aurora PostgreSQL Limitless Database에서 지원됩니다.

PostgreSQL에는 명시적 UPSERT 명령이 없지만 INSERT ... ON CONFLICT 문을 지원합니다.

충돌 작업에 하위 쿼리 또는 변경 가능한 함수가 있는 경우 INSERT ... ON CONFLICT는 지원되지 않습니다.

```
-- RANDOM is a mutable function.
INSERT INTO sharded_table VALUES (1, 100) ON CONFLICT (id) DO UPDATE SET other_id =
 RANDOM();

ERROR: Aurora Limitless Tables doesn't support pushdown-unsafe functions with DO UPDATE
 clauses.
```

INSERT 명령에 대한 자세한 내용은 PostgreSQL 설명서의 INSERT를 참조하세요.

UPDATE

샤드 키 업데이트는 지원되지 않습니다. 예를 들어 customer_id 샤드 키가 포함된 customers라는 샤딩된 테이블이 있습니다. 다음 DML 문은 오류를 일으킵니다.

```
postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_id =1;
ERROR:  Shard key column update is not supported

postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_name='abc';
ERROR:  Shard key column update is not supported
```

샤드 키를 업데이트하려면 먼저 샤드 키를 사용하여 행에 DELETE 작업을 수행한 다음 업데이트된 샤드 키 값을 사용하여 INSERT 작업으로 새 행을 삽입해야 합니다.

UPDATE 명령에 대한 자세한 내용은 PostgreSQL 설명서의 Updating data를 참조하세요.

VACUUM

샤딩된 테이블과 참조 테이블 모두에 vacuum 작업을 수행할 수 있습니다. 다음 VACUUM 함수는 Aurora PostgreSQL Limitless Database에서 완전히 지원됩니다.

- VACUUM

- ANALYZE
- DISABLE_PAGE_SKIPPING
- FREEZE
- FULL
- INDEX_CLEANUP
- PARALLEL
- PROCESS_TOAST
- TRUNCATE
- 상세 표시

Aurora PostgreSQL Limitless Database에서 VACUUM에는 다음과 같은 제한 사항이 있습니다.

- pg_visibility_map 확장은 지원되지 않습니다.
- pg_stat_all_indexes 뷰를 사용하여 미사용 인덱스를 확인하는 것은 지원되지 않습니다.
- pg_stat_user_indexes, pg_class 및 pg_stats에 대한 통합 뷰는 구현되지 않습니다.

VACUUM 명령에 대한 자세한 내용은 PostgreSQL 설명서의 VACUUM을 참조하세요. Aurora PostgreSQL Limitless Database에서 vacuum 작업이 작동하는 방법에 대한 자세한 내용은 Vacuum 작업을 통한 스토리지 공간 회수 섹션을 참조하세요.

Aurora PostgreSQL Limitless Database의 변수

다음 변수를 사용하여 Aurora PostgreSQL Limitless Database를 구성할 수 있습니다.

rds_aurora.limitless_active_shard_key

데이터베이스를 쿼리하는 동안 단일 샤드 키를 설정하므로 모든 SELECT 및 DML 쿼리가 샤드 키와 함께 상수 조건자로 추가됩니다. 자세한 내용은 활성 샤드 키 설정 단원을 참조하십시오.

rds_aurora.limitless_create_table_collocate_with

이 변수를 특정 테이블 이름으로 설정하여 새로 만든 테이블을 해당 테이블과 공동 배치합니다. 자세한 내용은 변수를 사용하여 무한 테이블 만들기 단원을 참조하십시오.

rds_aurora.limitless_create_table_mode

테이블 만들기 모드를 설정합니다. 자세한 내용은 변수를 사용하여 무한 테이블 만들기 단원을 참조하십시오.

rds_aurora.limitless_create_table_shard_key

이 변수를 샤드 키로 사용할 열 이름 배열로 설정합니다. 자세한 내용은 변수를 사용하여 무한 테이블 만들기 단원을 참조하십시오.

rds_aurora.limitless_explain_options

EXPLAIN 출력에 포함할 항목입니다. 자세한 내용은 EXPLAIN 단원을 참조하십시오.

Aurora PostgreSQL Limitless Database의 DB 클러스터 파라미터

다음 DB 클러스터 파라미터를 사용하여 Aurora PostgreSQL Limitless Database를 구성할 수 있습니다.

rds_aurora.limitless_adaptive_fetch_size

배치 미리 가져오기를 개선합니다. true로 설정하면 이 파라미터는 미리 가져오기에서 가져오기 크기 자체 조정(적응)을 허용합니다. false로 설정하면 가져오기 크기가 일정합니다.

rds_aurora.limitless_auto_scale_options

DB 샤드 그룹에서 라우터를 추가하거나 샤드를 분할하는 데 사용할 수 있는 옵션을 설정합니다. 값은 add_router, split_shard 또는 둘 다일 수 있습니다.

자세한 내용은 DB 샤드 그룹에 라우터 추가 및 DB 샤드 그룹에서 샤드 분할 단원을 참조하세요.

rds_aurora.limitless_distributed_deadlock_timeout

분산 교착 상태가 있는지 확인하기 전에 잠금을 기다리는 데 걸리는 시간(밀리초)입니다. 기본값은 1000(1초)입니다.

자세한 내용은 Aurora PostgreSQL Limitless Database의 분산 교착 단원을 참조하십시오.

rds_aurora.limitless_enable_auto_scale

DB 샤드 그룹에서 라우터 추가 및 샤드 분할을 활성화합니다.

자세한 내용은 DB 샤드 그룹에 라우터 추가 및 DB 샤드 그룹에서 샤드 분할 단원을 참조하세요.

rds_aurora.limitless_finalize_split_shard_mode

시스템 시작한 샤드 분할을 마무리하는 방법을 결정합니다. 자세한 내용은 DB 샤드 그룹에서 샤드 분할 단원을 참조하십시오.

rds_aurora.limitless_maximum_adaptive_fetch_size

적응형 가져오기 크기의 상한을 설정합니다. 범위는 1~INT_MAX입니다. 기본값은 1000입니다.

Amazon Aurora Global Database 사용

Amazon Aurora Global Database 기능을 사용하면 여러 AWS 리전에 걸쳐 있는 여러 Aurora DB 클러스터를 설정합니다. Aurora는 기본 DB 클러스터의 모든 변경 사항을 하나 이상의 보조 클러스터와 자동으로 동기화합니다. Aurora 글로벌 데이터베이스는 하나의 리전에 프라이머리 DB 클러스터를 그리고 서로 다른 리전에 최대 10개의 보조 DB 클러스터를 포함합니다. 이 다중 리전 구성은 전체 AWS 리전에 영향을 미칠 수 있는 드물게 발생하는 중단으로부터 빠르게 복구합니다. 여러 지리적 위치에 모든 데이터의 전체 복사본을 보유하면 전 세계에 널리 분산된 위치에서 연결되는 애플리케이션의 읽기 작업도 지연 시간이 짧게 처리할 수 있습니다.

주제

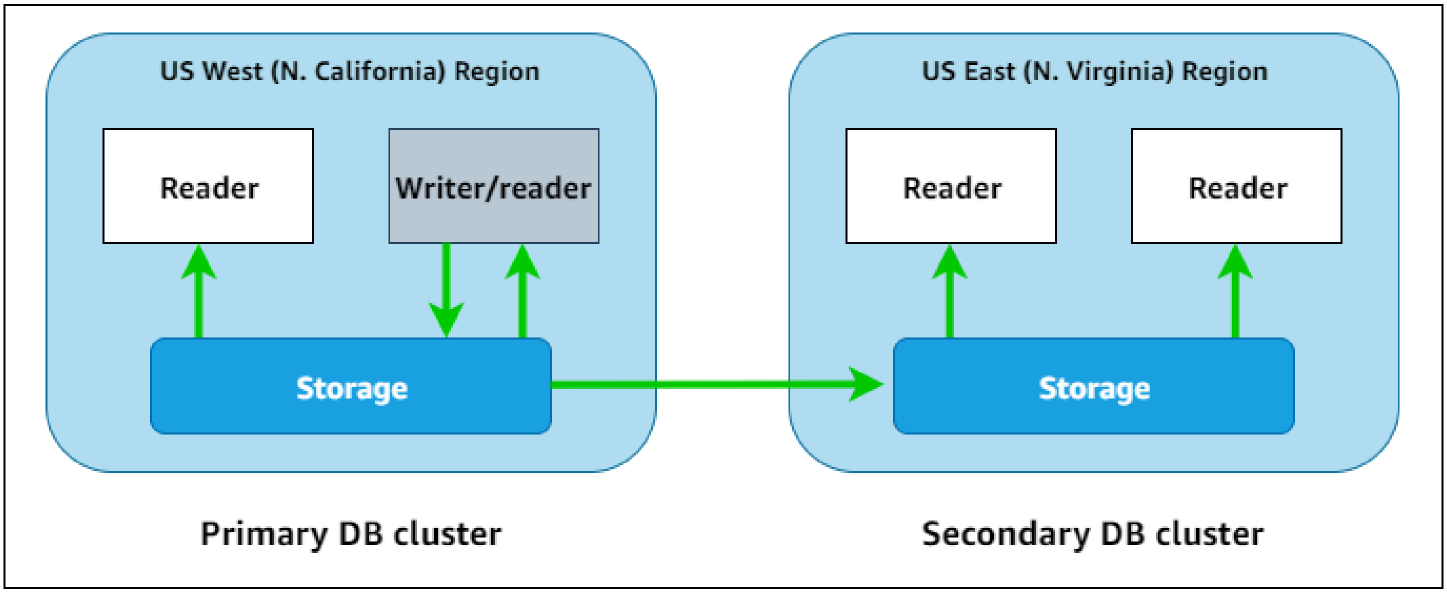
Amazon Aurora Global Database 개요

Amazon Aurora Global Database 기능을 사용하면 여러 AWS 리전에 걸쳐 있는 단일 Aurora 데이터베이스를 사용하여 전역으로 분산된 애플리케이션을 실행할 수 있습니다.

Aurora 글로벌 데이터베이스는 데이터가 써지는 하나의 프라이머리 AWS 리전 및 최대 10개의 읽기 전용 보조 AWS 리전으로 구성됩니다. 쓰기 연산을 기본 AWS 리전의 기본 DB 클러스터에서 시작합니다. 이렇게 하는 가장 편리한 방법은 Aurora Global Database 라이터 엔드포인트에 연결하는 것입

니다. 이 엔드포인트는 다른 AWS 리전으로 전환 또는 장애 조치 후에도 항상 기본 DB 클러스터를 가리킵니다. 쓰기 작업 후 Aurora는 일반적으로 1초 미만의 대기 시간으로 전용 인프라를 사용하여 보조 AWS 리전에 데이터를 복제합니다.

다음 다이어그램에서 두개의 AWS 리전에 걸쳐 있는 Aurora Global Database의 예를 볼 수 있습니다.



하나 이상의 Aurora 리더 인스턴스를 추가하여 읽기 전용 워크로드를 처리함으로써 보조 클러스터를 독립적으로 확장할 수도 있습니다. Aurora Serverless v2를 리더 인스턴스에 사용하여 더 세분화되고 유연한 조정을 수행할 수 있습니다.

기본 클러스터만 쓰기 작업을 수행합니다. 쓰기 작업을 수행하는 클라이언트는 항상 기본 클러스터의 라이터 DB 인스턴스를 가리키는 Aurora Global Database 라이터 엔드포인트에 연결됩니다. 다이어그램에 표시된 대로 Aurora는 빠른 저 오버헤드 복제에 데이터베이스 엔진이 아닌 클러스터 스토리지 볼륨을 사용합니다. 자세한 내용은 Amazon Aurora 스토리지 개요를 참조하세요.

Aurora Global Database는 전 세계 설치 공간을 갖춘 애플리케이션 용으로 설계되었습니다. 읽기 전용 보조 DB 클러스터는 여러 AWS 리전에서 애플리케이션 사용자와 근접하게 읽기 작업 최적화를 지원합니다. 쓰기 전달 기능을 사용하면 보조 클러스터가 기본 클러스터로 쓰기 요청을 보내도록 글로벌 데이터베이스를 구성할 수도 있습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 섹션을 참조하세요.

Aurora Global Database는 2가지 시나리오(Aurora Global Database 전환 및 Aurora Global Database 장애 조치)에 따라 기본 DB 클러스터의 리전을 변경하는 2가지 작업을 지원합니다.

- 리전 순환과 같이 계획된 운영 절차의 경우 전환 메커니즘(이전 명칭: '계획된 관리형 장애 조치')을 사용하세요. 이 기능을 사용하면 정상적인 Aurora Global Database의 기본 클러스터를 데이터 손실

없이 보조 리전 중 하나로 재배치할 수 있습니다. 자세한 내용은 Amazon Aurora Global Database에서 전환 수행를 참조하세요.

- 기본 리전에서 중단된 후 Aurora Global Database를 복구하려면 장애 조치 메커니즘을 사용하세요. 이 기능을 사용하면 기본 DB 클러스터에서 다른 리전으로 장애 조치를 수행할 수 있습니다(리전 간 장애 조치). 자세한 내용은 Aurora Global Database에서 계획된 관리형 장애 조치 수행를 참조하세요.

Amazon Aurora Global Database의 장점

Aurora Global Database를 사용하면 다음과 같은 이점을 얻을 수 있습니다.

- 로컬 대기 시간으로 글로벌 읽기 – 전 세계에 지사를 두고 있는 경우, Aurora Global Database를 사용하여 기본 AWS 리전에서 정보의 주요 소스를 최신 상태로 유지할 수 있습니다. 다른 리전의 사무실은 로컬 지연 시간을 사용하여 해당 리전의 정보에 액세스할 수 있습니다.
- 확장 가능한 보조 Aurora DB 클러스터 – AWS 리전에 읽기 전용 인스턴스를 추가하여 보조 클러스터를 확장할 수 있습니다. 세컨더리 클러스터는 읽기 전용이므로 단일 Aurora 클러스터에 대하여 일반적인 제한인 15개가 아닌 최대 16개의 읽기 전용 DB 인스턴스를 지원할 수 있습니다.
- 기본 DB클러스터에서 보조 Aurora DB 클러스터로의 빠른 복제 – Aurora Global Database에서 수행되는 복제는 기본 DB 클러스터의 성능에 거의 영향을 미치지 않습니다. DB 인스턴스의 리소스는 애플리케이션 읽기/쓰기 워크로드 처리 전용입니다.
- 리전 전체의 운영 중단으로부터 복구 – 보조 클러스터를 사용하면 기존 복제 솔루션보다 적은 데이터 손실(더 낮은 RPO)로 새로운 기본 AWS 리전에서 Aurora Global Database를 보다 신속하게(더 낮은 RTO) 가용 상태로 만들 수 있습니다.

리전 및 버전 사용 가능 여부

기능에 관한 가용성 및 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. Aurora Global Database의 버전 및 리전 가용성에 대한 자세한 정보는 Aurora Global Database를 지원하는 리전 및 DB 엔진 섹션을 참조하세요.

Amazon Aurora Global Database에 적용되는 제한 사항

현재 다음 제한이 Aurora Global Database에 적용됩니다.

- Aurora Global Database는 특정 AWS 리전에서 사용할 수 있으며 특정 Aurora MySQL 및 Aurora PostgreSQL 버전에 사용할 수 있습니다. 자세한 내용은 Aurora Global Database를 지원하는 리전 및 DB 엔진 섹션을 참조하세요.
- Aurora Global Database에는 지원되는 Aurora DB 인스턴스 클래스, 최대 AWS 리전 수 등에 대한 구성 관련 특정 요구 사항이 있습니다. 자세한 내용은 Amazon Aurora Global Database의 구성 요구 사항 섹션을 참조하세요.
- MySQL 5.7과 호환되는 Aurora MySQL의 경우 Aurora Global Database 전환에는 버전 2.09.1 이상의 마이너 버전이 필요합니다.
- 기본 및 보조 DB 클러스터에 있는 메이저 및 마이너 엔진 버전이 동일한 경우에만 Aurora Global Database에서 관리형 리전 간 전환 또는 장애 조치를 수행할 수 있습니다. 엔진 및 엔진 버전에 따라 패치 수준이 동일해야 하거나 패치 수준이 다를 수 있습니다. 패치 수준이 다른 기본 클러스터와 보조 클러스터 간에 이러한 작업을 허용하는 엔진 및 엔진 버전 목록은 관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성 섹션을 참조하세요. 엔진 버전에 동일한 패치 수준이 필요한 경우 Aurora Global Database에서 수동 장애 조치 수행에 나온 단계에 따라 수동으로 장애 조치를 수행할 수 있습니다.
- Aurora Global Database는 현재 다음 Aurora 기능을 지원하지 않습니다.
  - Aurora Serverless v1
  - Aurora 내 역추적
- RDS 프록시 기능을 글로벌 데이터베이스에서 사용할 때의 제한 사항은 글로벌 데이터베이스에서 RDS 프록시의 제한 사항 섹션을 참조하세요.
- 자동 마이너 버전 업그레이드는 Aurora MySQL 및 글로벌 데이터베이스의 부분인 Aurora PostgreSQL 클러스터에는 적용되지 않습니다. 글로벌 데이터베이스 클러스터의 일부인 DB 인스턴스에 대해 이 설정을 지정할 수 있지만 이 설정은 아무런 영향을 미치지 않습니다.
- Aurora Global Database는 현재 DB 데이터베이스 클러스터에 대해 Aurora Auto Scaling을 지원하지 않습니다.
- Aurora MySQL 5.7을 실행하는 Aurora Global Database에서 데이터베이스 활동 스트림(DAS)을 사용하려면 엔진 버전이 버전 2.08 이상이어야 합니다. DAS에 대한 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 섹션을 참조하세요.
- 현재 다음 제한이 Aurora Global Database 업그레이드에 적용됩니다.
  - 해당 Aurora 전역 데이터베이스의 메이저 버전 업그레이드를 수행하는 동안에는 전역 데이터베이스 클러스터에 대한 사용자 지정 파라미터 그룹을 적용할 수 없습니다. 전역 클러스터의 각 리전에 사용자 지정 파라미터 그룹을 생성한 다음 업그레이드 후 리전 클러스터에 수동으로 적용합니다.
  - Aurora MySQL 기반 Aurora Global Database 사용 시 `lower_case_table_names` 파라미터가 활성화 되어 있는 경우 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수

없습니다. 사용할 수 있는 메서드에 대한 자세한 내용은 메이저 버전 업그레이드 섹션을 참조하세요.

  - Aurora Global Database 사용 시 목표 복구 시점(RPO) 기능이 켜져 있는 경우 Aurora PostgreSQL DB 엔진의 메이저 버전 업그레이드를 수행할 수 없습니다. RPO 기능에 대한 자세한 내용은 Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO 관리 섹션을 참조하세요.
  - Aurora Global Database 사용 시 표준 프로세스를 사용하여 Aurora MySQL 버전 3.01 또는 3.02에서 3.03 이상으로 마이너 버전 업그레이드를 수행할 수 없습니다. 사용할 프로세스에 대한 자세한 내용은 엔진 버전을 수정하여 Aurora MySQL 업그레이드 섹션을 참조하세요.

  Aurora Global Database 업그레이드에 대한 자세한 내용은 Amazon Aurora Global Database 업그레이드 섹션을 참조하세요.

- 글로벌 데이터베이스에서 Aurora DB 클러스터를 개별적으로 중지하거나 시작할 수는 없습니다. 자세한 내용은 Amazon Aurora DB 클러스터 중지 및 시작를 참조하세요.
- 보조 Aurora DB 클러스터에 연결된 Aurora 리더 DB 인스턴스는 특정 상황에서 다시 시작될 수 있습니다. 기본 AWS 리전의 라이터 DB 인스턴스에 재시작 또는 장애 조치가 발생할 경우 보조 리전의 리더 DB 인스턴스도 다시 시작됩니다. 모든 리더 DB 인스턴스가 기본 DB 클러스터의 라이터 인스턴스와 다시 동기화될 때까지 보조 클러스터를 사용할 수 없습니다. 재부팅 시 또는 장애 조치 중에 기본 클러스터의 동작은 단일 비글로벌 DB 클러스터의 동작과 동일합니다. 자세한 내용은 Amazon Aurora를 사용한 복제 섹션을 참조하세요.

  기본 DB 클러스터를 변경하기 전에 전역 데이터베이스에 대한 영향을 이해해야 합니다. 자세한 내용은 계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구를 참조하세요.

- Aurora Global Database에서는 현재 Amazon Aurora가 DB 클러스터의 AWS KMS 키에 대한 액세스 권한을 상실할 때 `inaccessible-encryption-credentials-recoverable` 상태를 지원하지 않습니다. 이 경우 암호화된 DB 클러스터는 바로 최종 `inaccessible-encryption-credentials` 상태가 됩니다. 이러한 상태에 대한 자세한 내용은 DB 클러스터 상태 보기 섹션을 참조하세요.
- Secrets Manager는 Aurora Global Database를 지원하지 않습니다. 글로벌 데이터베이스에 리전을 추가할 때는 먼저 DB 인스턴스에 대한 Secrets Manager 통합을 꺼야 합니다.
- Aurora Global Database를 사용하는 Aurora PostgreSQL 기반 DB 클러스터에는 다음과 같은 제한 사항이 있습니다:
  - Aurora 글로벌 데이터베이스의 일부인 Aurora PostgreSQL 보조 DB 클러스터에는 클러스터 캐시 관리가 지원되지 않습니다.
  - 글로벌 데이터베이스의 기본 DB 클러스터가 Amazon RDS PostgreSQL 인스턴스의 복제본을 기반으로 하는 경우 보조 클러스터를 생성할 수 없습니다. AWS Management Console, AWS CLI,

또는 `CreateDBCluster` API 작업을 사용하여 해당 클러스터에서 보조 클러스터를 생성하지 마십시오. 이렇게 하면 시간이 초과되며 세컨더리 클러스터가 생성되지 않습니다.

기본 DB 엔진과 동일한 버전의 DB 엔진을 사용하여 글로벌 데이터베이스에 대한 보조 Aurora DB 클러스터를 생성하는 것이 좋습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 생성 섹션을 참조하세요.

Amazon Aurora Global Database 시작하기

Aurora Global Database를 시작하려면 먼저 어떤 Aurora DB 엔진을 사용할지, 어떤 AWS 리전에서 사용할지 결정합니다. 특정 AWS 리전의 Aurora MySQL 및 Aurora PostgreSQL 데이터베이스 엔진의 특정 버전에 한하여 Aurora Global Database를 지원합니다. 완전한 목록은 Aurora Global Database를 지원하는 리전 및 DB 엔진 단원을 참조하세요.

다음 방법 중 하나로 Aurora Global Database를 만들 수 있습니다.

- 새 Aurora DB 클러스터 및 Aurora DB 인스턴스로 새 Aurora Global Database 생성 – Amazon Aurora 글로벌 데이터베이스 생성의 단계를 수행하여 이 작업을 수행할 수 있습니다. 기본 Aurora DB 클러스터를 생성한 후 Amazon Aurora Global Database에 AWS 리전 추가의 단계에 따라 보조 AWS 리전을 추가합니다.
- Aurora Global Database 기능을 지원하는 기존 Aurora DB 클러스터를 사용하고 여기에 AWS 리전 추가 – 기존 Aurora DB 클러스터가 글로벌 호환되는 DB 엔진 버전을 사용하는 경우에만 이 작업을 수행할 수 있습니다.

  Aurora DB 클러스터를 선택할 때 AWS Management Console에서 [작업(Action)]에 대해 [리전 추가(Add region)]를 선택할 수 있는지 확인합니다. 선택 가능하면 해당 Aurora DB 클러스터를 Aurora 전역 클러스터에 사용할 수 있습니다. 자세한 내용은 Amazon Aurora Global Database에 AWS 리전 추가 섹션을 참조하세요.

Aurora Global Database를 만들기 전에 모든 구성 요구 사항을 이해하는 것이 좋습니다.

주제

- [Aurora 또는 Amazon RDS 스냅샷에서 Amazon Aurora Global Database 생성]

Amazon Aurora Global Database의 구성 요구 사항

글로벌 데이터베이스 작업을 시작하기 전에 사용할 클러스터 및 인스턴스 이름, AWS 리전, 인스턴스 수, 인스턴스 클래스를 계획하세요. 선택 항목이 다음 구성 요구 사항과 일치하는지 확인하세요.

Aurora Global Database는 두 개 이상의 AWS 리전에 걸쳐 있습니다. 기본 AWS 리전은 하나의 라이터 Aurora DB 인스턴스가 있는 Aurora DB 클러스터를 지원합니다. 보조 AWS 리전은 전체가 Aurora 복제본으로 구성된 읽기 전용 Aurora DB 클러스터를 실행합니다. 보조 AWS 리전이 한 개 이상 필요하지만, Aurora 글로벌 데이터베이스는 최대 10개의 보조 AWS 리전을 포함할 수 있습니다. 이 표에는 Aurora Global Database에서 허용되는 최대 Aurora DB 클러스터, Aurora DB 인스턴스 및 Aurora 복제본이 나열됩니다.

설명	기본 AWS 리전	보조 AWS 리전
Aurora DB 클러스터	1	10개(최대)
라이터 인스턴스	1	0
Aurora DB 클러스터당 읽기 전용 인스턴스 (Aurora 복제본)	15(최대)	16(최대)
읽기 전용 인스턴스(최대 허용, 지정된 보조 리전 수)	15 - s	s = 총 보조 AWS 리전 수

Aurora Global Database를 구성하는 Aurora DB 클러스터의 특정 요구 사항은 다음과 같습니다.

- DB 인스턴스 클래스 요구 사항 – Aurora Global Database에는 메모리 집약적 애플리케이션에 최적화된 DB 인스턴스 클래스가 필요합니다. 메모리 최적화 DB 인스턴스 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 유형] 섹션을 참조하세요. db.r5 이상의 인스턴스 클래스를 사용하는 것이 좋습니다.
- AWS 리전 요구 사항 – Aurora Global Database에는 하나의 AWS 리전에 기본 Aurora DB 클러스터가 필요하고 다른 리전에 하나 이상의 보조 Aurora DB 클러스터가 필요합니다. 보조(읽기 전용) Aurora DB 클러스터를 10개까지 만들 수 있으며 각각은 다른 리전에 있어야 합니다. 즉, Aurora Global Database의 두 Aurora DB 클러스터가 동일한 AWS 리전에 있을 수 없습니다.

Aurora DB 엔진, 엔진 버전 및 Aurora Global Database와 함께 사용할 수 있는 AWS 리전의 조합에 대한 자세한 내용은 Aurora Global Database를 지원하는 리전 및 DB 엔진 섹션을 참조하세요.

- 이름 설정 요구 사항 – 개별 Aurora DB 클러스터에 선택하는 이름은 모든 AWS 리전에서 고유해야 합니다. 다른 리전에 있더라도 다른 Aurora DB 클러스터에는 동일한 이름을 사용할 수 없습니다.
- Aurora Serverless v2의 용량 요구 사항 – Aurora Serverless v2의 글로벌 데이터베이스에서 프라이머리AWS 리전의 DB 클러스터의 최소 권장 용량은 8ACU입니다.

이 섹션의 절차를 따르기 전에 AWS 계정가 필요합니다. Amazon Aurora 작업을 위한 설정 작업을 완료합니다. 자세한 내용은 Amazon Aurora 환경 설정 섹션을 참조하세요. 또한 Aurora DB 클러스터를 생성하기 위한 다른 예비 단계도 완료해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

글로벌 데이터베이스를 설정할 준비가 되면 Amazon Aurora 글로벌 데이터베이스 생성에서 필요한 모든 리소스를 만드는 절차를 참조하세요. 기존 Aurora 클러스터를 프라이머리 클러스터로 사용하여 글로벌 데이터베이스를 생성하려면 Amazon Aurora Global Database에 AWS 리전 추가의 절차를 따라야 할 수도 있습니다.

Amazon Aurora 글로벌 데이터베이스 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora Global Database 및 관련 리소스를 만들려면 다음 단계를 따르세요.

> **Note**
>
> 글로벌 호환 가능한 Aurora 데이터베이스 엔진을 실행하는 기존 Aurora DB 클러스터가 있는 경우, 이 절차의 더 짧은 형식을 사용할 수 있습니다. 이 경우, 기존 DB 클러스터에 다른 AWS 리전을 추가하여 Aurora Global Database를 생성할 수 있습니다. 그렇게 하려면 Amazon Aurora Global Database에 AWS 리전 추가 단원을 참조하십시오.

콘솔

Aurora Global Database 생성 단계는 Aurora Global Database 기능을 지원하는 AWS 리전에 로그인하는 것으로 시작됩니다. 전체 목록은 Aurora Global Database를 지원하는 리전 및 DB 엔진 단원을 참조하세요.

다음 단계 중 하나는 Aurora DB 클러스터에 Amazon VPC 기반 Virtual Private Cloud(VPC)를 선택하는 것입니다. 사용자의 VPC를 사용하려면 VPC를 선택할 수 있도록 미리 생성해 두는 것이 좋습니다. 동

시에, 관련 서브넷을 생성하고 필요에 따라 서브넷 그룹 및 보안 그룹을 생성합니다. 자세한 방법은 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)을 참조하세요.

Aurora DB 클러스터 생성에 대한 일반적인 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하세요.

Aurora 글로벌 데이터베이스를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스 생성을 선택합니다. 데이터베이스 생성 페이지에서 다음을 수행합니다.
   - 데이터베이스 생성 방법으로 [표준 생성(Standard Create)]을 선택합니다. (손쉬운 생성을 선택하지 마세요.)
   - `Engine type`의 엔진 옵션 섹션에서 해당하는 엔진 유형인 Aurora(MySQL 호환) 또는 Aurora(PostgreSQL 호환)를 선택합니다.
3. 다음 절차의 단계를 사용하여 Aurora Global Database 만들기를 계속합니다.

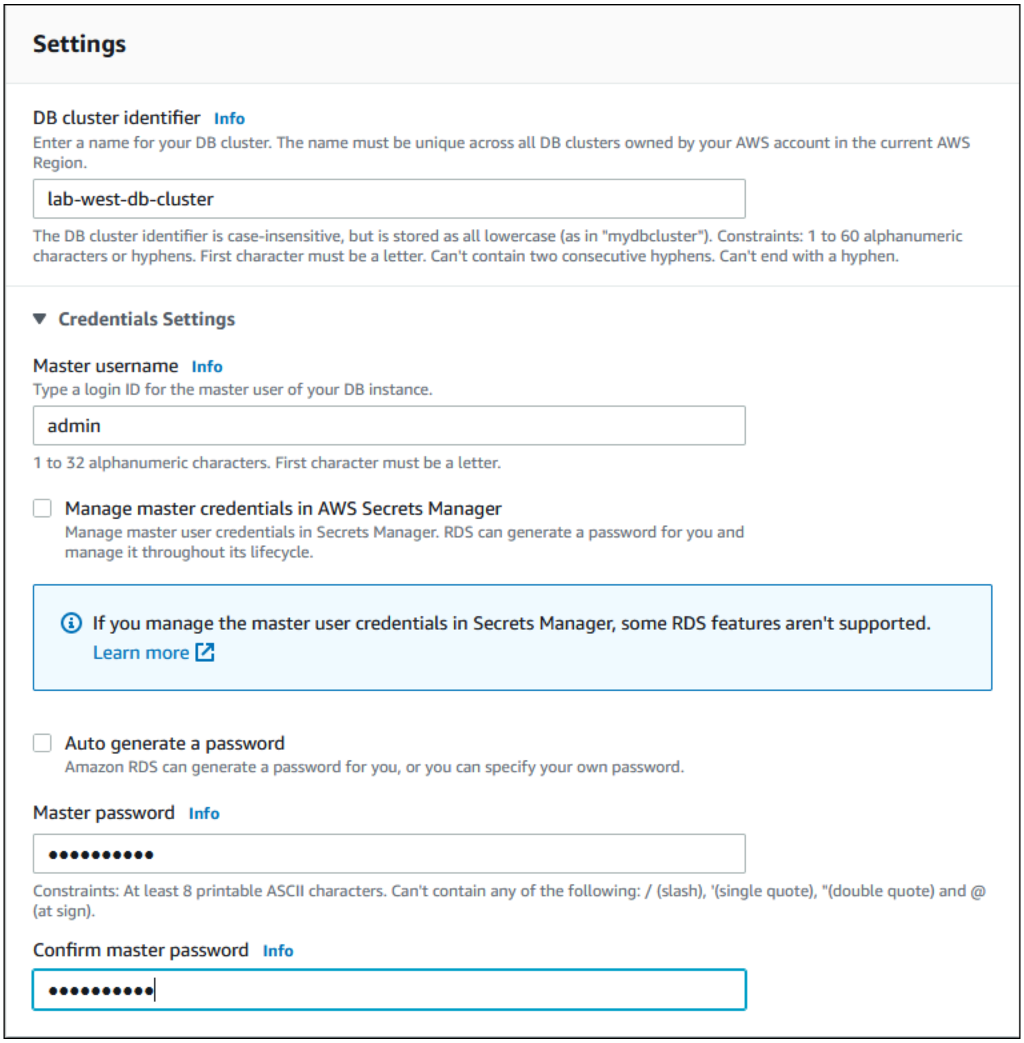
Aurora MySQL을 사용하여 글로벌 데이터베이스 생성

다음 단계는 Aurora MySQL의 모든 버전에 적용됩니다.

Aurora MySQL을 사용하여 Aurora Global Database를 생성하는 방법

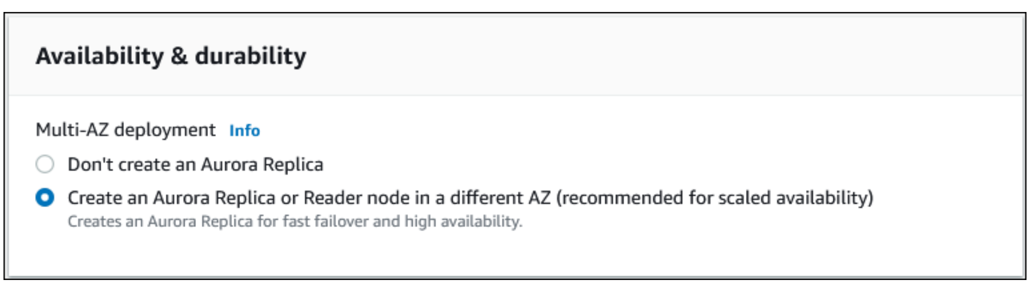
데이터베이스 생성(Create database) 페이지를 완료합니다.

1. Engine options(엔진 옵션)의 경우 다음을 선택합니다.
   - Engine version(엔진 버전)의 경우 Aurora Global Database에 사용할 Aurora MySQL의 버전을 선택합니다.
2. 템플릿의 경우, 프로덕션을 선택합니다. 또는 사용 사례에 적합한 경우 개발/테스트를 선택할 수 있습니다. 프로덕션 환경에서 개발/테스트를 사용하지 마십시오.
3. 설정에서 다음을 수행합니다.
   a. DB 클러스터 식별자에 대한 의미 있는 이름을 입력합니다. Aurora 글로벌 데이터베이스 생성을 마치면 이 이름은 기본 DB 클러스터를 식별합니다.
   b. DB 인스턴스의 `admin` 사용자 계정에 대한 고유한 암호를 입력하거나 사용자를 위해 Aurora를 생성합니다. 암호 자동 생성을 선택하면 암호를 복사할 수 있는 옵션이 표시됩니다.



4. [DB 인스턴스 클래스(DB instance class)]에서 `db.r5.large` 또는 다른 메모리 최적화 DB 인스턴스 클래스를 선택합니다. db.r5 이상의 인스턴스 클래스를 사용하는 것이 좋습니다.

5. 가용성 및 내구성의 경우 다른 가용 영역(AZ)에서 Aurora가 Aurora 복제본을 생성하도록 선택하는 것이 좋습니다. 지금 Aurora 복제본을 생성하지 않으면 나중에 복제본을 생성해야 합니다.



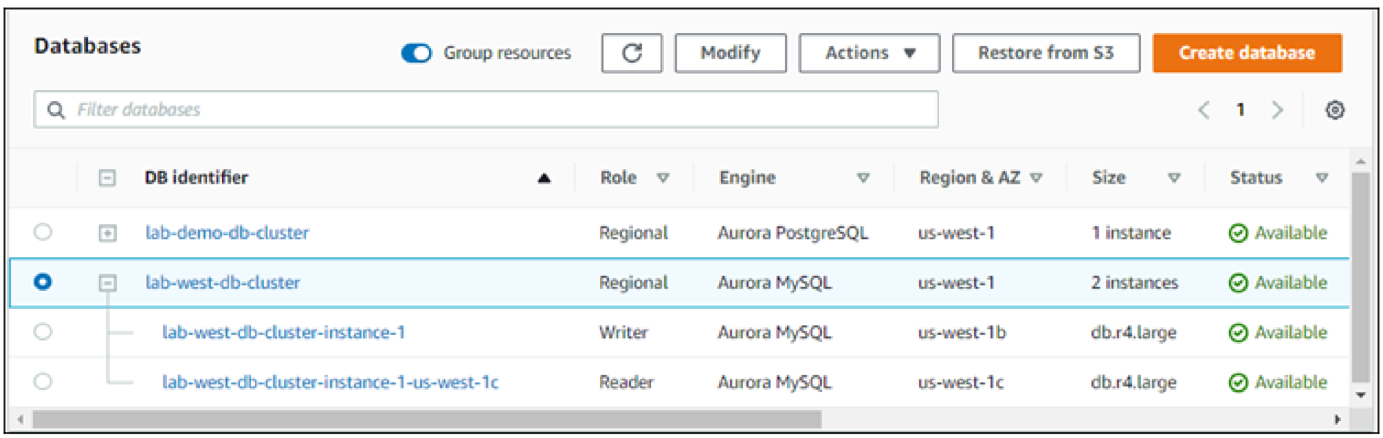
6. Connectivity(연결)의 경우 이 DB 인스턴스의 가상 네트워킹 환경을 정의하는 Amazon VPC을 기반으로 하는 Virtual Private Cloud(VPC)를 선택합니다. 기본값을 선택하여 이 작업을 단순화할 수 있습니다.

7. 데이터베이스 인증 설정을 완료합니다. 프로세스를 단순화하기 위해 지금 암호 인증을 선택하고 AWS Identity and Access Management(IAM)을 나중에 설정할 수 있습니다.
8. 추가 구성에서 다음을 수행합니다.

   a. 이 클러스터의 기본 Aurora DB 인스턴스를 생성하려면 초기 데이터베이스 이름에 이름을 입력합니다. Aurora 기본 DB 클러스터의 라이터 노드입니다.

      사용할 사용자 지정 파라미터 그룹이 없는 경우 DB 클러스터 파라미터 그룹 및 DB 파라미터 그룹에 대해 기본값을 선택된 상태로 둡니다.
   b. [역추적 사용(Enable backtrack)] 확인란 선택을 취소합니다(선택되어 있는 경우). Aurora Global Database는 역추적을 지원하지 않습니다. 그렇지 않으면 추가 구성에 다른 기본 설정을 적용할 수 있습니다.
9. 데이터베이스 생성을 선택합니다.

   Aurora DB 인스턴스, 이 Aurora 복제본 및 Aurora DB 클러스터 생성 프로세스를 완료하기 위해 Aurora에 몇 분 정도 소요될 수 있습니다. Aurora DB 클러스터를 Aurora Global Database에서 기본 DB 클러스터로 사용할 준비가 된 시기를 해당 상태로 알 수 있습니다. 이 경우, 해당 상태와 라이터 및 복제본 노드의 상태는 다음과 같이 사용 가능입니다.



기본 DB 클러스터를 사용할 수 있는 경우 보조 클러스터를 추가하여 Aurora Global Database를 생성합니다. 이렇게 하려면 Amazon Aurora Global Database에 AWS 리전 추가 섹션의 단계를 따르세요.

Aurora PostgreSQL을 사용하여 글로벌 데이터베이스 생성

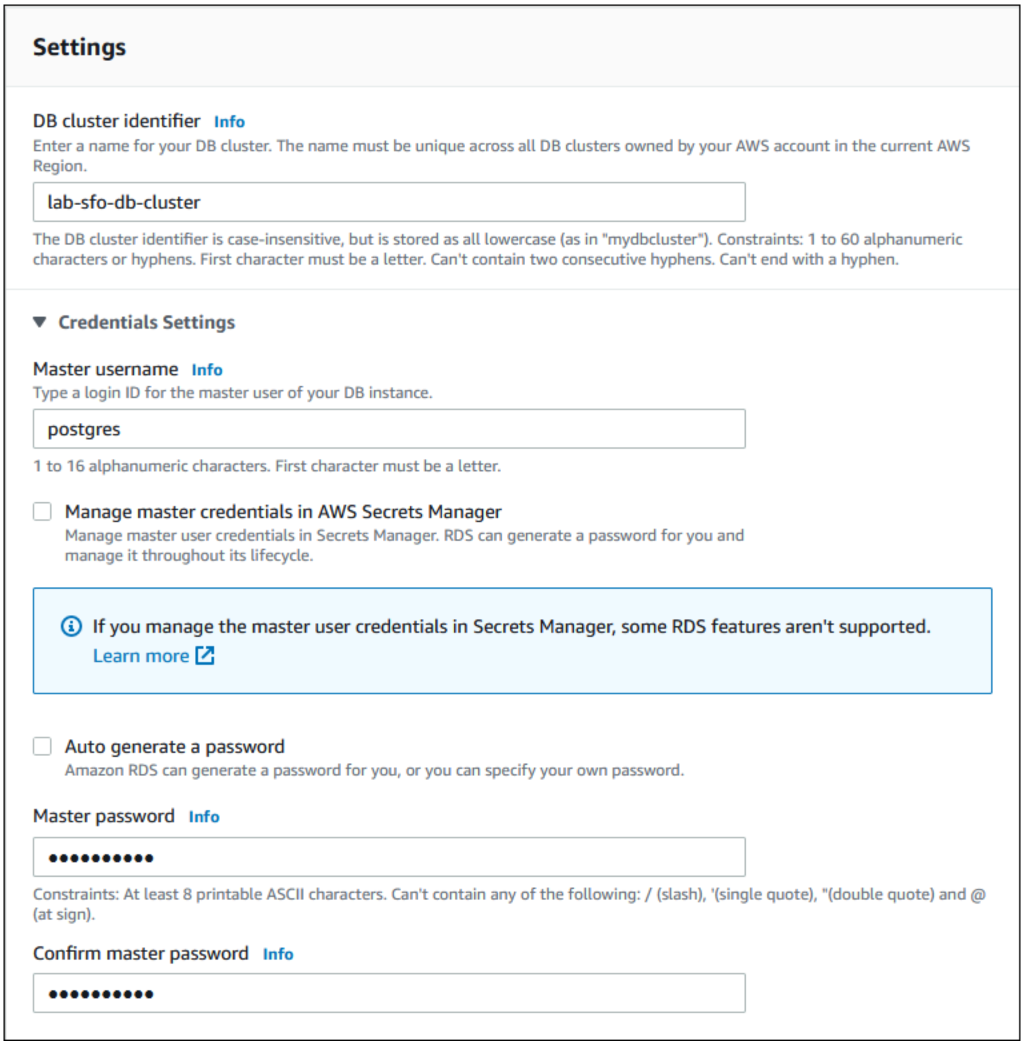
Aurora PostgreSQL을 사용하여 Aurora Global Database를 생성하는 방법

데이터베이스 생성(Create database) 페이지를 완료합니다.

1. Engine options(엔진 옵션)의 경우 다음을 선택합니다.

- Engine version(엔진 버전)의 경우 Aurora Global Database에 사용할 Aurora PostgreSQL의 버전을 선택합니다.

2. 템플릿의 경우, 프로덕션을 선택합니다. 또는 해당하는 경우 개발/테스트를 선택할 수 있습니다. 프로덕션 환경에서 개발/테스트를 사용하지 마십시오.
3. 설정에서 다음을 수행합니다.
   a. DB 클러스터 식별자에 대한 의미 있는 이름을 입력합니다. Aurora 글로벌 데이터베이스 생성을 마치면 이 이름은 기본 DB 클러스터를 식별합니다.
   b. DB 인스턴스의 사용자 계정에 대한 고유한 암호를 입력하거나 사용자를 위해 Aurora를 생성합니다. 암호 자동 생성을 선택하면 암호를 복사할 수 있는 옵션이 표시됩니다.



4. [DB 인스턴스 클래스(DB instance class)]에서 `db.r5.large` 또는 다른 메모리 최적화 DB 인스턴스 클래스를 선택합니다. db.r5 이상의 인스턴스 클래스를 사용하는 것이 좋습니다.

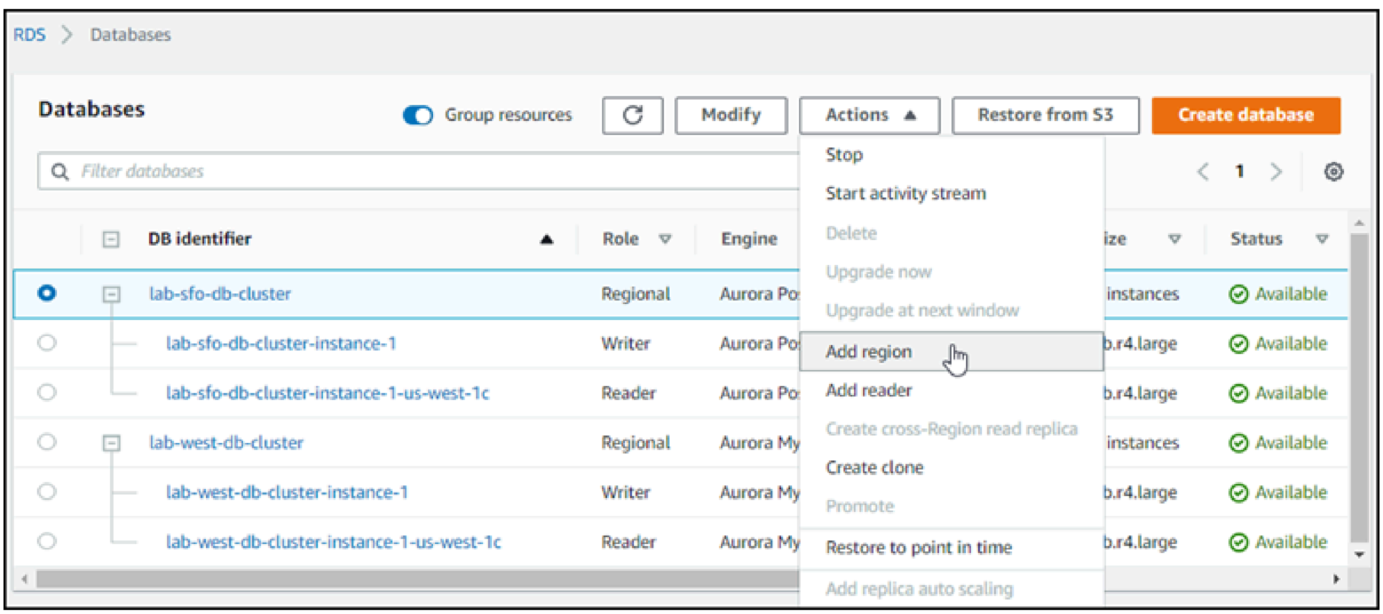
5. Availability & durability(가용성 및 내구성)의 경우 다른 AZ에서 Aurora 복제본을 생성하도록 Aurora를 선택하기를 권장합니다. 지금 Aurora 복제본을 생성하지 않으면 나중에 복제본을 생성해야 합니다.
6. 연결의 경우 이 DB 인스턴스의 가상 네트워킹 환경을 정의하는 Amazon VPC을 기반으로 하는 Virtual Private Cloud(VPC)를 선택합니다. 기본값을 선택하여 이 작업을 단순화할 수 있습니다.
7. (선택 사항) Database authentication(데이터베이스 인증) 설정을 완료합니다. 암호 인증이 항상 사용됩니다. 프로세스를 단순화하기 위해 이 섹션을 건너뛰고 나중에 IAM 또는 암호 및 Kerberos 인증을 설정할 수 있습니다.
8. 추가 구성에서 다음을 수행합니다.

    a. 이 클러스터의 기본 Aurora DB 인스턴스를 생성하려면 초기 데이터베이스 이름에 이름을 입력합니다. Aurora 기본 DB 클러스터의 라이터 노드입니다.

       사용할 사용자 지정 파라미터 그룹이 없는 경우 DB 클러스터 파라미터 그룹 및 DB 파라미터 그룹에 대해 기본값을 선택된 상태로 둡니다.

    b. 암호화, 로그 내보내기 등의 추가 구성에 대한 다른 모든 기본 설정을 적용합니다.
9. 데이터베이스 생성을 선택합니다.

   Aurora DB 인스턴스, 이 Aurora 복제본 및 Aurora DB 클러스터 생성 프로세스를 완료하기 위해 Aurora에 몇 분 정도 소요될 수 있습니다. 클러스터를 사용할 준비가 되면 Aurora DB 클러스터와 해당 라이터 및 복제본 노드의 상태가 [사용 가능(Available)]으로 표시됩니다. 이것은 보조 데이터베이스를 추가하면 Aurora 글로벌 데이터베이스의 기본 DB 클러스터가 됩니다.



기본 DB 클러스터가 사용할 수 있게 되면 Amazon Aurora Global Database에 AWS 리전 추가의 단계에 따라 보조 클러스터를 하나 이상 생성합니다.

AWS CLI

다음 절차의 AWS CLI 명령은 다음 작업을 수행합니다.

1. Aurora Global Database를 만들고 이름을 지정하며 사용할 Aurora 데이터베이스 엔진 유형을 지정합니다.
2. Aurora 글로벌 데이터베이스에 대한 Aurora DB 클러스터를 생성합니다.
3. 클러스터에 대한 Aurora DB 인스턴스를 생성합니다. 이것은 글로벌 데이터베이스의 기본 Aurora DB 클러스터입니다.
4. Aurora DB 클러스터용 두 번째 DB 인스턴스를 생성합니다. Aurora DB 클러스터를 완료하기 위한 리더입니다.
5. Amazon Aurora Global Database에 AWS 리전 추가의 다음 단계에 따라 다른 리전에서 두 번째 Aurora DB 클러스터를 생성한 다음 Aurora 글로벌 데이터베이스에 추가합니다.

Aurora 데이터베이스 엔진에 대한 절차를 따릅니다.

Aurora MySQL을 사용하여 글로벌 데이터베이스 생성

Aurora MySQL을 사용하여 Aurora Global Database를 생성하는 방법

1. `create-global-cluster` CLI 명령을 사용하여 AWS 리전, Aurora 데이터베이스 엔진 및 버전의 이름을 전달합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-global-cluster --region primary_region \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql \
    --engine-version version # optional
```

   Windows의 경우:

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora-mysql ^
    --engine-version version # optional
```

이러면 이름 (식별자) 및 Aurora 데이터베이스 엔진이 포함된 "빈" Aurora 글로벌 데이터베이스가 생성됩니다. Aurora 글로벌 데이터베이스를 사용할 수 있기까지 몇 분 정도 걸릴 수 있습니다. 다음 단계로 넘어가기 전에 `describe-global-clusters` CLI 명령을 사용하여 사용할 수 있는지 확인합니다.

```
aws rds describe-global-clusters --region primary_region --global-cluster-
identifier global_database_id
```

Aurora 글로벌 데이터베이스를 사용할 수 있는 경우 기본 Aurora DB 클러스터를 생성할 수 있습니다.

2. 기본 Aurora DB 클러스터를 생성하려면 `create-db-cluster` CLI 명령을 사용합니다. `--global-cluster-identifier` 파라미터를 사용하여 Aurora 글로벌 데이터베이스의 이름을 포함합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   Windows의 경우:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   `describe-db-clusters` AWS CLI 명령을 사용하여 Aurora DB 클러스터가 준비되었는지 확인합니다. 특정 Aurora DB 클러스터를 골라 내려면 `--db-cluster-identifier` 파라미터를 사용

합니다. 또는 명령에 Aurora DB 클러스터 이름을 생략하여 지정된 리전의 모든 Aurora DB 클러스터에 대한 세부 정보를 얻을 수 있습니다.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

응답에 클러스터에 대한 `"Status": "available"`이 표시되면 사용할 준비가 된 것입니다.

3. 기본 Aurora DB 클러스터에 대한 DB 인스턴스를 생성합니다. 이렇게 하려면 [`create-db-instance`] CLI 명령을 사용합니다. 명령에 Aurora DB 클러스터의 이름을 지정하고 인스턴스에 대한 구성 세부 정보를 지정합니다. Aurora DB 클러스터에서 파라미터를 가져오므로 명령에서 `--master-username` 및 `--master-user-password` 파라미터를 전달할 필요가 없습니다.

   `--db-instance-class`의 경우 메모리 최적화 클래스의 인스턴스 클래스(예: `db.r5.large`)만 사용할 수 있습니다. db.r5 이상의 인스턴스 클래스를 사용하는 것이 좋습니다. 이러한 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 유형] 섹션을 참조하세요.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-mysql \
  --engine-version version \
  --region primary_region
```

   Windows의 경우:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-mysql ^
  --engine-version version ^
  --region primary_region
```

   `create-db-instance` 작업을 완료하는 데 다소 시간이 걸립니다. 계속하기 전에 상태를 확인하여 Aurora DB 인스턴스를 사용할 수 있는지 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

명령이 `available` 상태를 반환하면 기본 DB 클러스터에 대해 다른 Aurora DB 인스턴스를 생성할 수 있습니다. Aurora DB 클러스터의 읽기 전용 인스턴스 (Aurora 복제본) 입니다.

4. 클러스터에 대한 다른 Aurora DB 인스턴스를 생성하려면 [`create-db-instance`] CLI 명령을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-mysql
```

   Windows의 경우:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

DB 인스턴스를 사용할 수 있는 경우 라이터 노드에서 복제본으로의 복제가 시작됩니다. 계속하기 전에 `describe-db-instances` CLI 명령으로 DB 인스턴스를 사용할 수 있는지 확인합니다.

이때 라이터 DB 인스턴스와 Aurora 복제본이 포함된 기본 Aurora DB 클러스터가 있는 Aurora 글로벌 데이터베이스가 있습니다. 이제 다른 리전에 읽기 전용 Aurora DB 클러스터를 추가하여 Aurora 글로벌 데이터베이스를 완료할 수 있습니다. 이렇게 하려면 Amazon Aurora Global Database에 AWS 리전 추가 단원의 절차를 따르십시오.

Aurora PostgreSQL을 사용하여 글로벌 데이터베이스 생성

다음 명령을 사용하여 Aurora Global Database에 대한 Aurora 객체를 생성할 때 각 객체를 사용할 수 있게 되려면 몇 분 정도 걸릴 수 있습니다. 지정된 명령을 완료한 후 특정 Aurora 객체의 상태를 확인하여 해당 상태를 사용할 수 있는지 확인하는 것이 좋습니다.

이렇게 하려면 `describe-global-clusters` CLI 명령을 사용합니다.

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

Aurora PostgreSQL을 사용하여 Aurora 글로벌 데이터베이스를 생성하려면

1. create-global-cluster CLI 명령을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-postgresql \
       --engine-version version # optional
   ```

   Windows의 경우:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-postgresql ^
       --engine-version version # optional
   ```

   Aurora 글로벌 데이터베이스를 사용할 수 있는 경우 기본 Aurora DB 클러스터를 생성할 수 있습니다.

2. 기본 Aurora DB 클러스터를 생성하려면 create-db-cluster CLI 명령을 사용합니다. `--global-cluster-identifier` 파라미터를 사용하여 Aurora 글로벌 데이터베이스의 이름을 포함합니다.

   대상 LinuxmacOS, 또는Unix:

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
     --master-username userid \
     --master-user-password password \
     --engine aurora-postgresql \
     --engine-version version \
     --global-cluster-identifier global_database_id
   ```

   Windows의 경우:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

Aurora DB 클러스터가 준비되었는지 확인합니다. 다음 명령의 응답에 Aurora DB 클러스터에 대한 `"Status": "available"`가 표시되면 계속할 수 있습니다.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. 기본 Aurora DB 클러스터에 대한 DB 인스턴스를 생성합니다. 이렇게 하려면 [`create-db-instance`] CLI 명령을 사용합니다.

   Aurora DB 클러스터의 이름을 `--db-cluster-identifier` 파라미터와 함께 전달

   Aurora DB 클러스터에서 파라미터를 가져오므로 명령에서 `--master-username` 및 `--master-user-password` 파라미터를 전달할 필요가 없습니다.

   `--db-instance-class`의 경우 메모리 최적화 클래스의 인스턴스 클래스(예: `db.r5.large`)만 사용할 수 있습니다. db.r5 이상의 인스턴스 클래스를 사용하는 것이 좋습니다. 이러한 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 유형] 섹션을 참조하세요.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

   Windows의 경우:

```
aws rds create-db-instance ^
```

```
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --region primary_region
```

4. 계속하기 전에 Aurora DB 인스턴스의 상태를 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

   응답에 Aurora DB 인스턴스 상태가 `available`로 표시되면 기본 DB 클러스터에 대해 다른 Aurora DB 인스턴스를 생성할 수 있습니다.

5. Aurora DB 클러스터용 Aurora 복제본을 생성하려면 [`create-db-instance`](#) CLI 명령을 사용합니다.

   대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

   Windows의 경우:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

DB 인스턴스를 사용할 수 있는 경우 라이터 노드에서 복제본으로의 복제가 시작됩니다. 계속하기 전에 [`describe-db-instances`](#) CLI 명령으로 DB 인스턴스를 사용할 수 있는지 확인합니다.

Aurora 글로벌 데이터베이스는 존재하지만, 라이터 DB 인스턴스와 Aurora 복제본으로 구성된 Aurora DB 클러스터가 있는 기본 리전만 있습니다. 이제 다른 리전에 읽기 전용 Aurora DB 클러스터를 추가하여 Aurora 글로벌 데이터베이스를 완료할 수 있습니다. 이렇게 하려면 [Amazon Aurora Global Database에 AWS 리전 추가](#) 단원의 절차를 따르십시오.

RDS API

RDS API를 사용하여 Aurora Global Database를 생성하려면 CreateGlobalCluster 작업을 실행합니다.

Amazon Aurora Global Database에 AWS 리전 추가

다음 절차를 사용하여 기존 글로벌 데이터베이스에 세컨더리 클러스터를 추가할 수 있습니다. 이 절차에 따라 첫 번째 보조 AWS 리전을 추가하여 독립형 Aurora DB 클러스터에서 글로벌 데이터베이스를 생성할 수도 있습니다.

Aurora Global Database에는 기본 Aurora DB 클러스터와 다른 AWS 리전에서 최소 한 개의 보조 Aurora DB 클러스터가 필요합니다. Aurora 글로벌 데이터베이스에 최대 10개의 보조 DB 클러스터를 연결할 수 있습니다. 각각의 새 세컨더리 DB 클러스터에 대해 다음 절차를 반복합니다. Aurora Global Database에 추가하는 각 보조 DB 클러스터에 대해 기본 DB 클러스터에 허용되는 Aurora 복제본 수를 하나씩 줄입니다.

예를 들어 Aurora 글로벌 데이터베이스에 10개의 보조 리전이 있는 경우 프라이머리 DB 클러스터에는 15개가 아닌 5개의 Aurora 복제본만 있을 수 있습니다. 자세한 내용은 Amazon Aurora Global Database의 구성 요구 사항 섹션을 참조하세요.

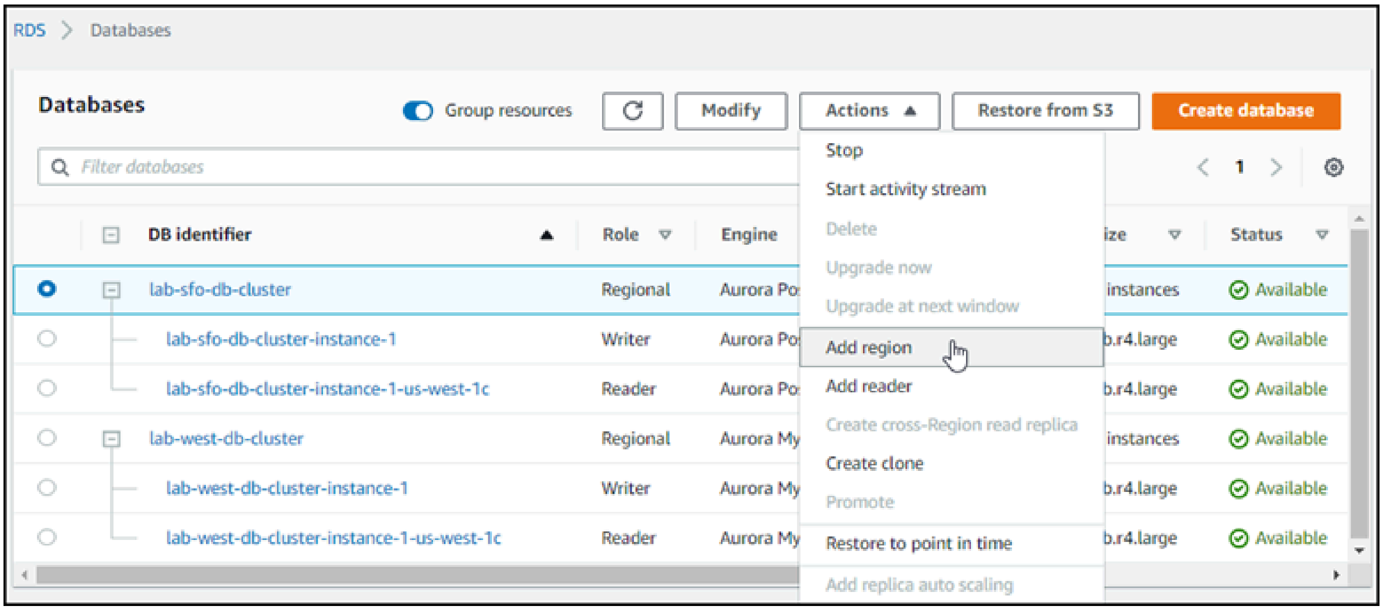
프라이머리 DB 클러스터의 Aurora 복제본(읽기 인스턴스) 수에 따라 추가할 수 있는 세컨더리 DB 클러스터 수가 결정됩니다. 프라이머리 DB 클러스터의 총 리더 인스턴스 수와 세컨더리 클러스터 수는 15를 초과할 수 없습니다. 예를 들어 프라이머리 DB 클러스터에 14개의 리더 인스턴스와 1개의 세컨더리 클러스터가 있는 경우 글로벌 데이터베이스에 다른 세컨더리 클러스터를 추가할 수 없습니다.

> **Note**
>
> Aurora MySQL 버전 3의 경우 보조 클러스터를 만들 때 `lower_case_table_names` 값이 기본 클러스터 값과 일치하는지 확인하세요. 이 설정은 서버가 식별자 대소문자 구분을 처리하는 방법에 영향을 주는 데이터베이스 파라미터입니다. 데이터베이스 파라미터에 대한 자세한 내용은 Amazon Aurora의 파라미터 그룹 내용을 참조하세요.
> 보조 클러스터를 생성할 때는 기본 클러스터와 보조 클러스터에 동일한 DB 엔진 버전을 사용하는 것이 좋습니다. 필요한 경우 기본 클러스터를 보조 클러스터와 동일한 버전으로 업그레이드합니다. 자세한 내용은 관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성 섹션을 참조하세요.

콘솔

Aurora Global Database에 AWS 리전 추가

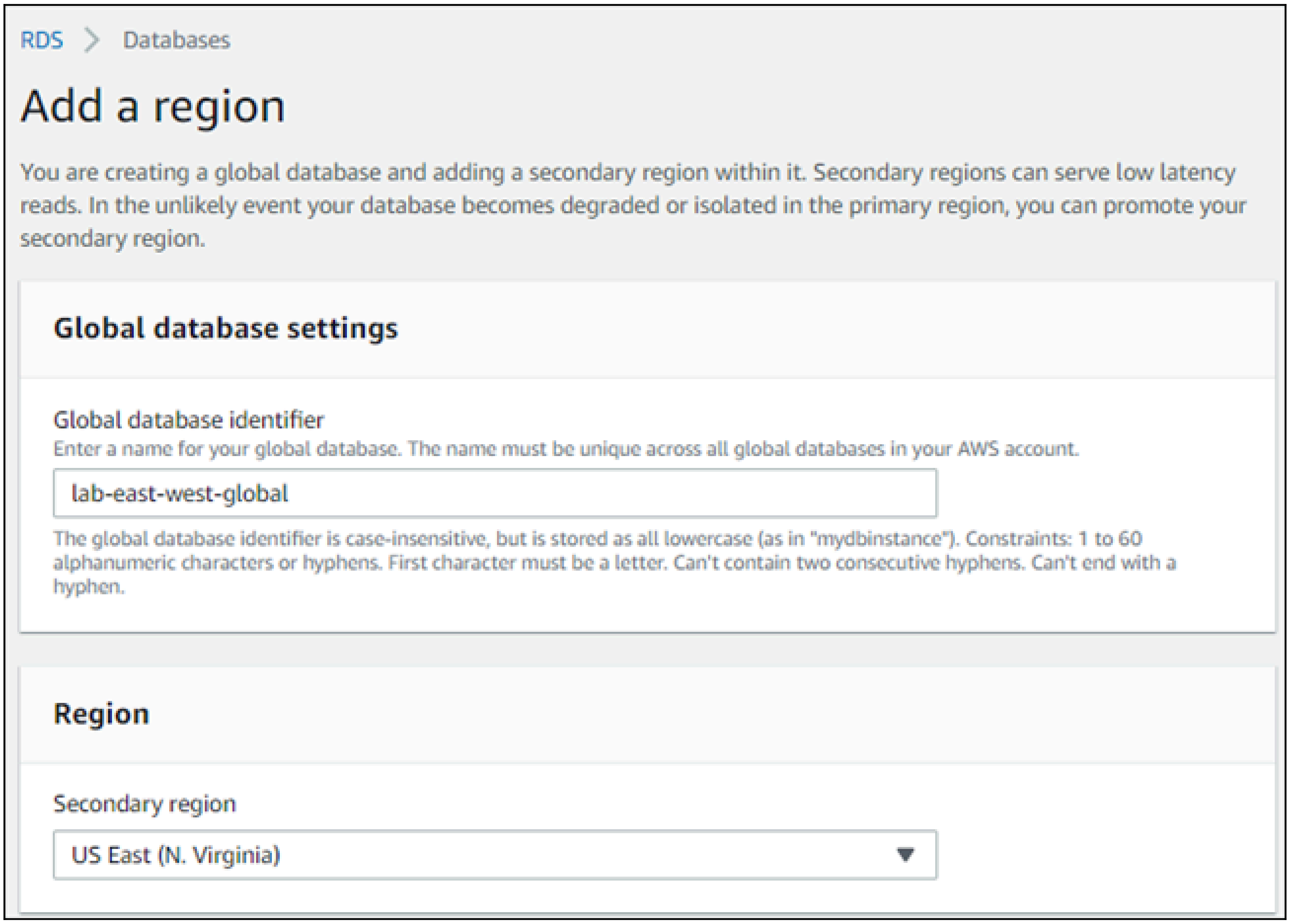
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 탐색 창에서 데이터베이스를 선택합니다.
3. 보조 Aurora DB 클러스터가 필요한 Aurora Global Database를 선택합니다. 기본 Aurora DB 클러스터가 `Available`인지 확인합니다.
4. 작업에서 AWS 리전 추가를 선택합니다.



5. 리전 추가 페이지에서 보조 AWS 리전을 선택합니다.

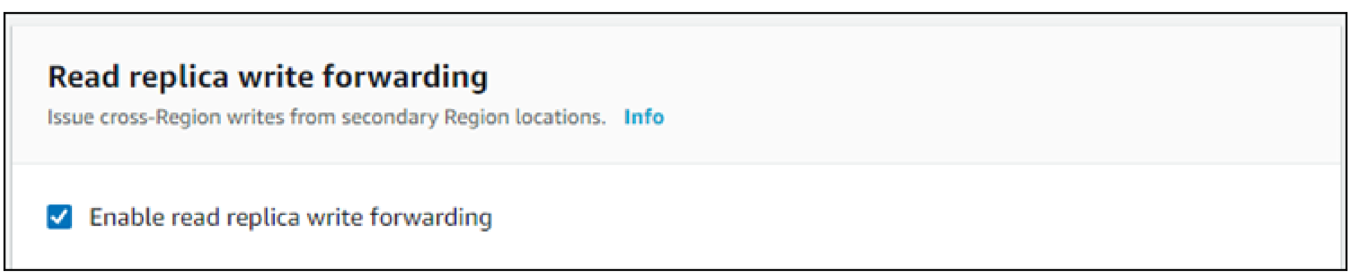
   동일한 Aurora Global Database에 대해 보조 Aurora DB 클러스터가 이미 있는 AWS 리전을 선택할 수 없습니다. 또한 이는 기본 Aurora DB 클러스터와 동일한 리전이 될 수 없습니다.

> **Note**
>
> Babelfish for Aurora PostgreSQL 글로벌 데이터베이스는 Babelfish 기본 설정을 제어하는 파라미터가 해당 리전에서 설정된 경우에만 보조 리전에서 작동합니다. 자세한 내용은 Babelfish용 DB 클러스터 파라미터 그룹 설정 섹션을 참조하세요.



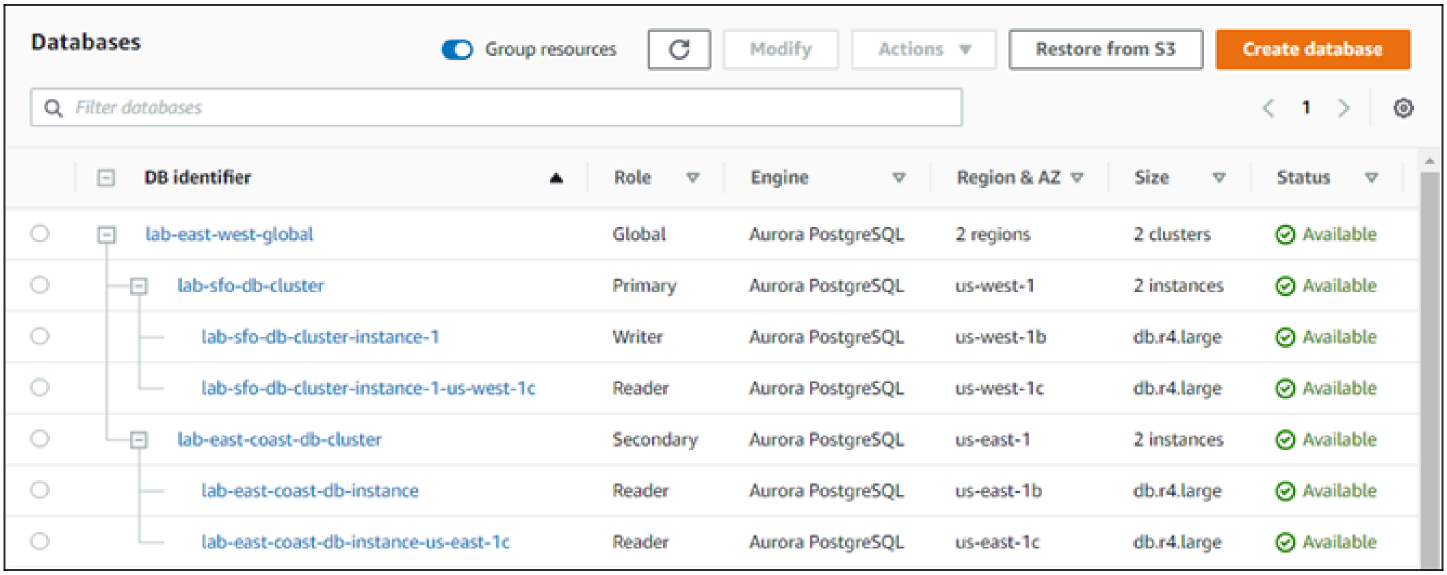
6. 새 AWS 리전에서 보조 Aurora 클러스터에 대한 나머지 필드를 완료합니다. 이 옵션은 Aurora DB 클러스터 인스턴스와 동일한 구성 옵션입니다. 단, Aurora MySQL–기반 Aurora Global Database에 대해 다음 옵션만 적용됩니다.

   • 읽기 복제본 쓰기 전달 활성화 – 이 옵션 설정을 사용하면 Aurora Global Database의 보조 DB 클러스터가 쓰기 작업을 기본 클러스터로 전달할 수 있습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 섹션을 참조하세요.

7. AWS 리전 추가를 선택합니다.

Aurora Global Database에 리전 추가를 완료한 다음 스크린샷에서 보듯이 AWS Management Console 의 [데이터베이스(Databases)] 목록에서 확인할 수 있습니다.



AWS CLI

Aurora Global Database에 보조 AWS 리전을 추가하는 방법

CLI를 사용하여 글로벌 데이터베이스에 세컨더리 클러스터를 추가하려면 글로벌 클러스터 컨테이너 객체가 이미 있어야 합니다. `create-global-cluster` 명령을 아직 실행하지 않았다면 [Amazon Aurora 글로벌 데이터베이스 생성](#)의 CLI 절차를 참조하세요.

1. [`create-db-cluster`](#) CLI 명령을 Aurora Global Database의 이름(`--global-cluster-identifier`)과 함께 사용합니다. 기타 명령 파라미터에서 다음을 수행합니다.
2. `--region`의 경우 Aurora 기본 리전과 다른 AWS 리전을 선택합니다.
3. `--engine` 및 `--engine-version` 파라미터의 구체적인 값을 선택합니다. 이러한 값은 Aurora Global Database의 기본 Aurora DB 클러스터에 대한 값과 같습니다.
4. 암호화된 클러스터의 경우 기본 AWS 리전을 암호화에 대한 `--source-region`로 지정합니다.

다음 예제에서는 새 Aurora DB 클러스터를 생성하여 Aurora Global Database에 읽기 전용 보조 Aurora DB 클러스터로 연결합니다. 마지막 단계에서 Aurora DB 인스턴스가 새 Aurora DB 클러스터에 추가됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region secondary_region \
```

```
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql | aurora-postgresql \
    --engine-version version

aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql | aurora-postgresql
```

Windows의 경우:

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql | aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql | aurora-postgresql
```

RDS API

RDS API를 사용하여 새 AWS 리전을 Aurora Global Database에 추가하려면 CreateDBCluster 작업을 실행합니다. `GlobalClusterIdentifier` 파라미터를 사용하여 기존 글로벌 데이터베이스의 식별자를 지정합니다.

보조 리전에 헤드리스 Aurora DB 클러스터 생성

Aurora Global Database는 기본 클러스터와 다른 AWS 리전에 하나 이상의 보조 Aurora DB 클러스터가 필요하지만, 보조 클러스터에 headless 구성을 사용할 수 있습니다. 헤드리스 세컨더리 Aurora DB 클러스터는 DB 인스턴스가 없는 클러스터입니다. 이 유형의 구성은 Aurora Global Database에 대한 비용을 줄일 수 있습니다. Aurora DB 클러스터에서는 컴퓨팅과 스토리지가 분리되어 있습니다. DB 인

스턴스가 없으면 컴퓨팅 요금이 청구되지 않고 스토리지 요금만 청구됩니다. 올바르게 설정되면, 헤드리스 보조 스토리지 볼륨이 기본 Aurora DB 클러스터와 동기화 상태를 유지합니다.

Aurora Global Database를 생성할 때 보통 때처럼 보조 클러스터를 추가합니다. 글로벌 데이터베이스의 모든 클러스터를 생성하는 경우 Amazon Aurora 글로벌 데이터베이스 생성의 절차를 따르세요. 프라이머리 클러스터로 사용할 DB 클러스터가 이미 있는 경우 Amazon Aurora Global Database에 AWS 리전 추가의 절차를 따르세요.

프라이머리 Aurora DB 클러스터가 세컨더리 DB 클러스터로 복제되기 시작한 후에는 세컨더리 Aurora DB 클러스터에서 Aurora 읽기 전용 DB 인스턴스를 삭제합니다. 이 세컨더리 클러스터에는 더 이상 DB 인스턴스가 없기 때문에 “헤드리스”로 간주됩니다. 세컨더리 클러스터에 DB 인스턴스가 없더라도 Aurora는 스토리지 볼륨을 프라이머리 Aurora DB 클러스터와 동기화합니다.

> **Warning**
>
> Aurora PostgreSQL을 사용하여 보조 AWS 리전에 헤드리스 클러스터를 생성하려면 AWS CLI 또는 RDS API를 사용하여 보조 AWS 리전을 추가합니다. 세컨더리 클러스터에 대한 리더 DB 인스턴스를 생성하는 단계를 건너뜁니다. 현재 헤드리스 클러스터 생성은 RDS 콘솔에서 지원되지 않습니다. 사용할 CLI 및 API 프로시저는 Amazon Aurora Global Database에 AWS 리전 추가 섹션을 참조하세요.
> 글로벌 데이터베이스에서 13.4, 12.8 또는 11.13 미만의 Aurora PostgreSQL 엔진 버전을 사용하는 경우 보조 리전에 리더 DB 인스턴스를 생성한 후 삭제하면 기본 리전의 라이터 DB 인스턴스에서 Aurora PostgreSQL 진공 문제가 발생할 수 있습니다. 이 문제가 발생하면 세컨더리 리전의 리더 DB 인스턴스를 삭제한 후 프라이머리 리전의 라이터 DB 인스턴스를 다시 시작합니다.

Aurora Global Database에 헤드리스 보조 Aurora DB 클러스터를 추가하는 방법

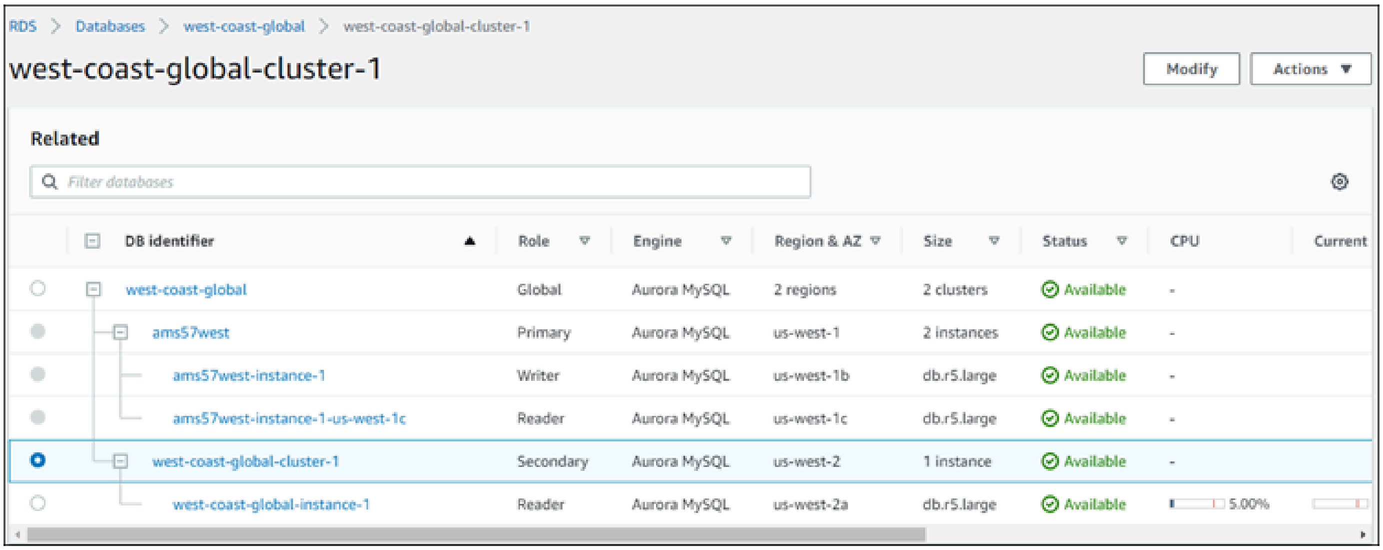
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 탐색 창에서 데이터베이스를 선택합니다.
3. 보조 Aurora DB 클러스터가 필요한 Aurora 글로벌 데이터베이스를 선택합니다. 기본 Aurora DB 클러스터가 `Available`인지 확인합니다.
4. 작업에서 AWS 리전 추가를 선택합니다.
5. 리전 추가 페이지에서 보조 AWS 리전을 선택합니다.

동일한 Aurora Global Database에 대해 보조 Aurora DB 클러스터가 이미 있는 AWS 리전을 선택할 수 없습니다. 또한 이는 기본 Aurora DB 클러스터와 동일한 리전이 될 수 없습니다.

6. 새 AWS 리전에서 보조 Aurora 클러스터에 대한 나머지 필드를 작성합니다. 이 옵션은 Aurora DB 클러스터 인스턴스와 동일한 구성 옵션입니다.

   Aurora MySQL–기반 Aurora Global Database의 경우, Enable read replica write forwarding(읽기 복제본 쓰기 전달 활성화) 옵션을 무시합니다. 리더 인스턴스를 삭제한 후에는 이 옵션을 사용할 수 없습니다.

7. AWS 리전 추가를 선택합니다. Aurora Global Database에 리전 추가를 완료한 다음 스크린샷에서 보듯이 AWS Management Console의 [데이터베이스(Databases)] 목록에서 확인할 수 있습니다.

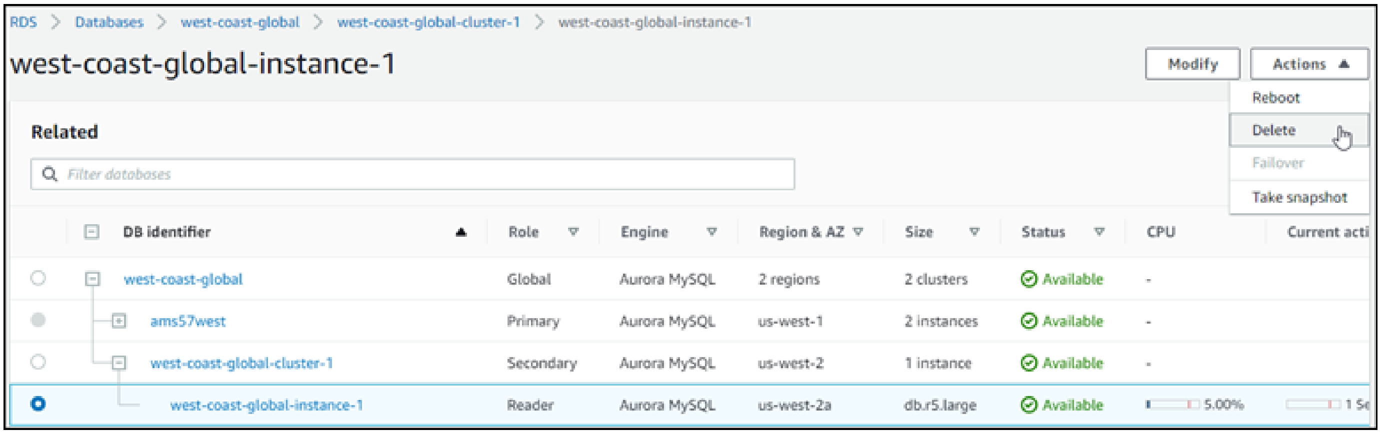


8. 계속하기 전에 AWS Management Console 또는 AWS CLI를 사용하여 보조 Aurora DB 클러스터 및 해당 리더 인스턴스의 상태를 확인합니다. 예:

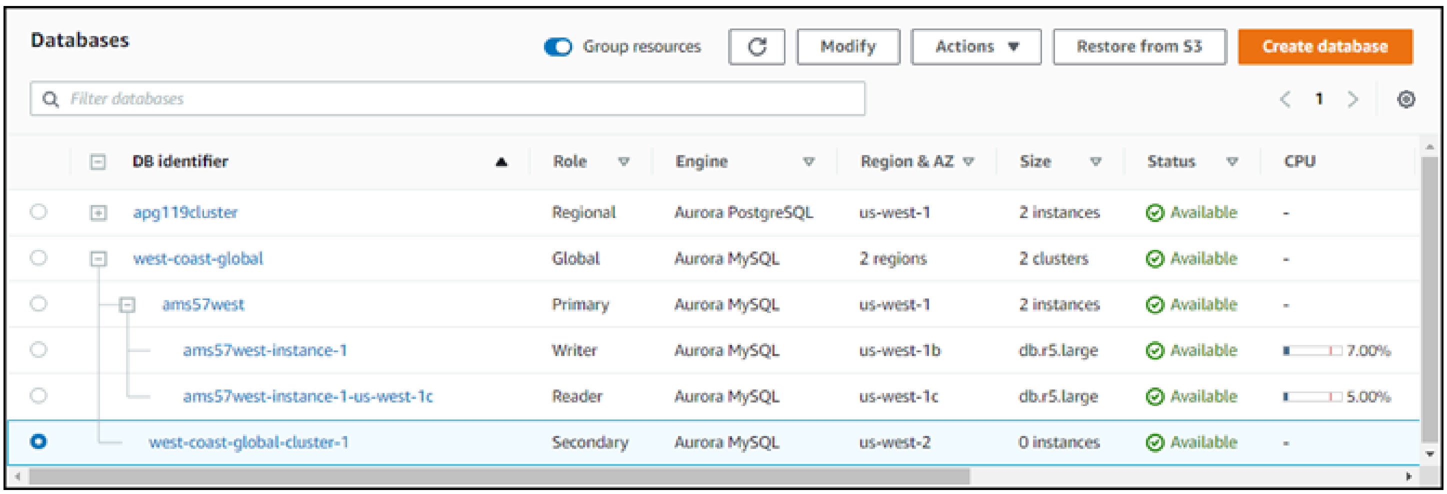
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
  '*[].[Status]' --output text
```

   새로 추가된 세컨더리 Aurora DB 클러스터의 상태가 `creating`에서 `available`로 변경되려면 몇 분 정도 걸릴 수 있습니다. Aurora DB 클러스터를 사용할 수 있게 되면 리더 인스턴스를 삭제할 수 있습니다.

9. 세컨더리 Aurora DB 클러스터에서 리더 인스턴스를 선택한 후 [삭제(Delete)]를 선택합니다.



리더 인스턴스를 삭제한 후 세컨더리 클러스터는 Aurora 글로벌 데이터베이스의 일부로 남습니다. 다음과 같이 연결된 인스턴스가 없습니다.



이러한 중단이 발생할 경우 이 헤드리스 보조 Aurora DB 클러스터를 사용하여 기본 AWS 리전의 계획되지 않은 중단으로부터 Amazon Aurora Global Database를 수동으로 복구할 수 있습니다.

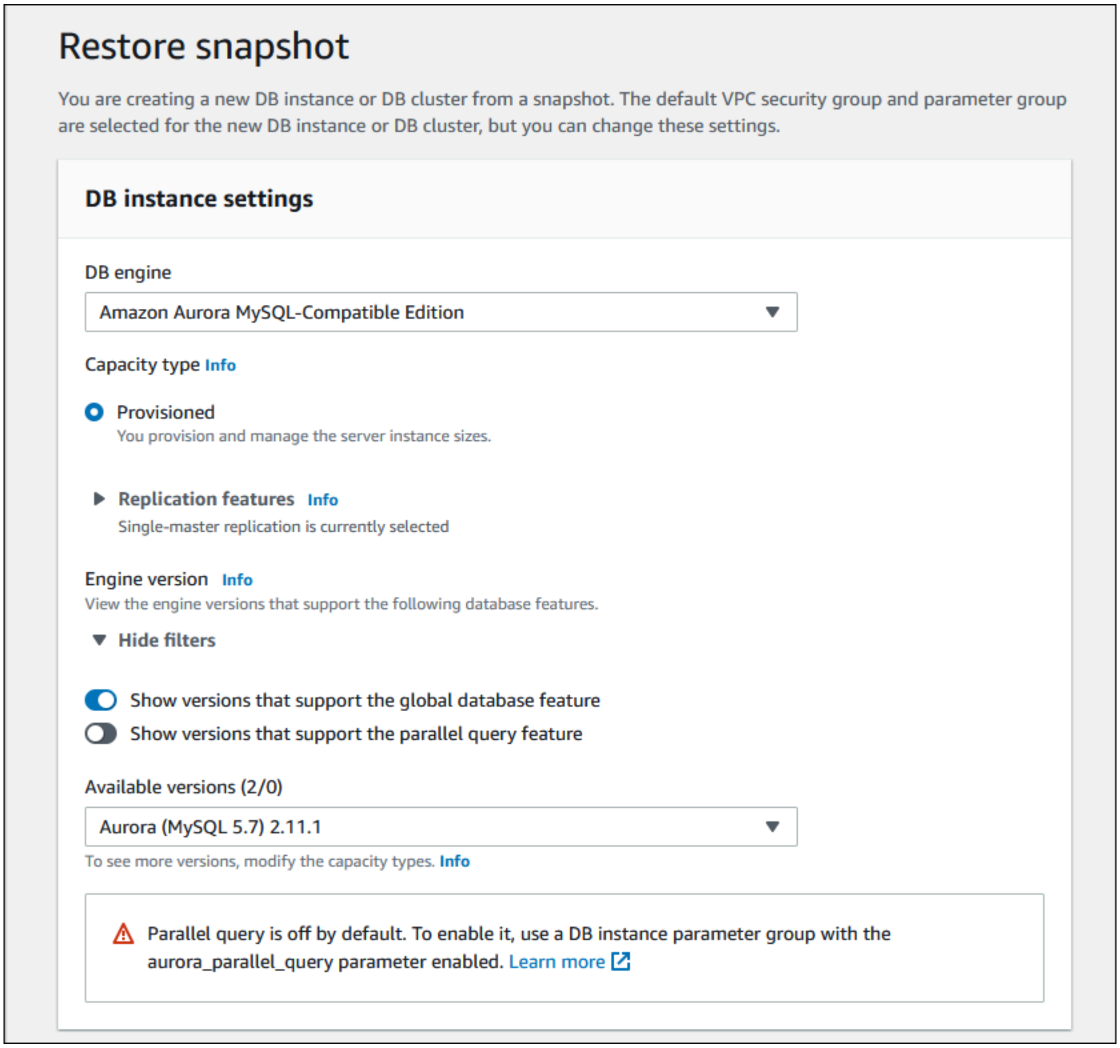
Aurora 또는 Amazon RDS 스냅샷에서 Amazon Aurora Global Database 생성

Aurora DB 클러스터의 스냅샷을 복원하거나 Amazon RDS DB 인스턴스에서 복원하여 Aurora 글로벌 데이터베이스의 시작점으로 사용할 수 있습니다. 스냅샷을 복원하고 새로운 Aurora 프로비저닝된 DB 클러스터를 동시에 생성합니다. 그런 다음 복원된 DB 클러스터에 다른 AWS 리전을 추가하여 Aurora Global Database로 전환합니다. 이 방법으로 스냅샷을 사용하여 생성하는 모든 Aurora DB 클러스터는 Aurora Global Database의 기본 클러스터가 됩니다.

사용하는 스냅샷은 DB 클러스터에서 `provisioned` 생성되거나 `serverless` Aurora DB 클러스터에서 생성될 수 있습니다.

복원 프로세스 중에 스냅샷과 동일한 DB 엔진 유형을 선택합니다. 예를 들어, Aurora PostgreSQL을 실행 중인 Aurora Serverless DB 클러스터에서 만든 스냅샷을 복원하려는 상황을 가정해 보겠습니다. 이 경우, 동일한 Aurora DB 엔진과 버전을 사용하여 Aurora PostgreSQL DB 클러스터를 생성합니다.

AWS 리전을 추가할 때 복원된 DB 클러스터는 Aurora Global Database에 대한 기본 클러스터 역할을 가정합니다. 이 기본 클러스터에 포함된 모든 데이터는 Aurora Global Database에 추가하는 보조 클러스터에 복제됩니다.



Amazon Aurora 글로벌 데이터베이스 관리

Aurora 글로벌 데이터베이스를 구성하는 개별 클러스터에서 대부분의 관리 작업을 수행할 수 있습니다. 콘솔의 데이터베이스 페이지에서 관련 리소스 그룹화를 선택하면, 연결된 전역 데이터베이스 아래

에 그룹화된 기본 클러스터와 보조 클러스터를 볼 수 있습니다. 글로벌 데이터베이스의 DB 클러스터가 실행 중인 AWS 리전, 해당 Aurora DB 엔진 및 버전, 해당 식별자를 찾으려면 구성(Configuration) 탭을 사용합니다.

리전 간 데이터베이스 장애 조치 프로세스는 단일 Aurora DB 클러스터가 아니라 Aurora 글로벌 데이터베이스에만 사용할 수 있습니다. 자세한 내용은 Amazon Aurora Global Database에서 전환 또는 장애 조치 사용을 참조하십시오.

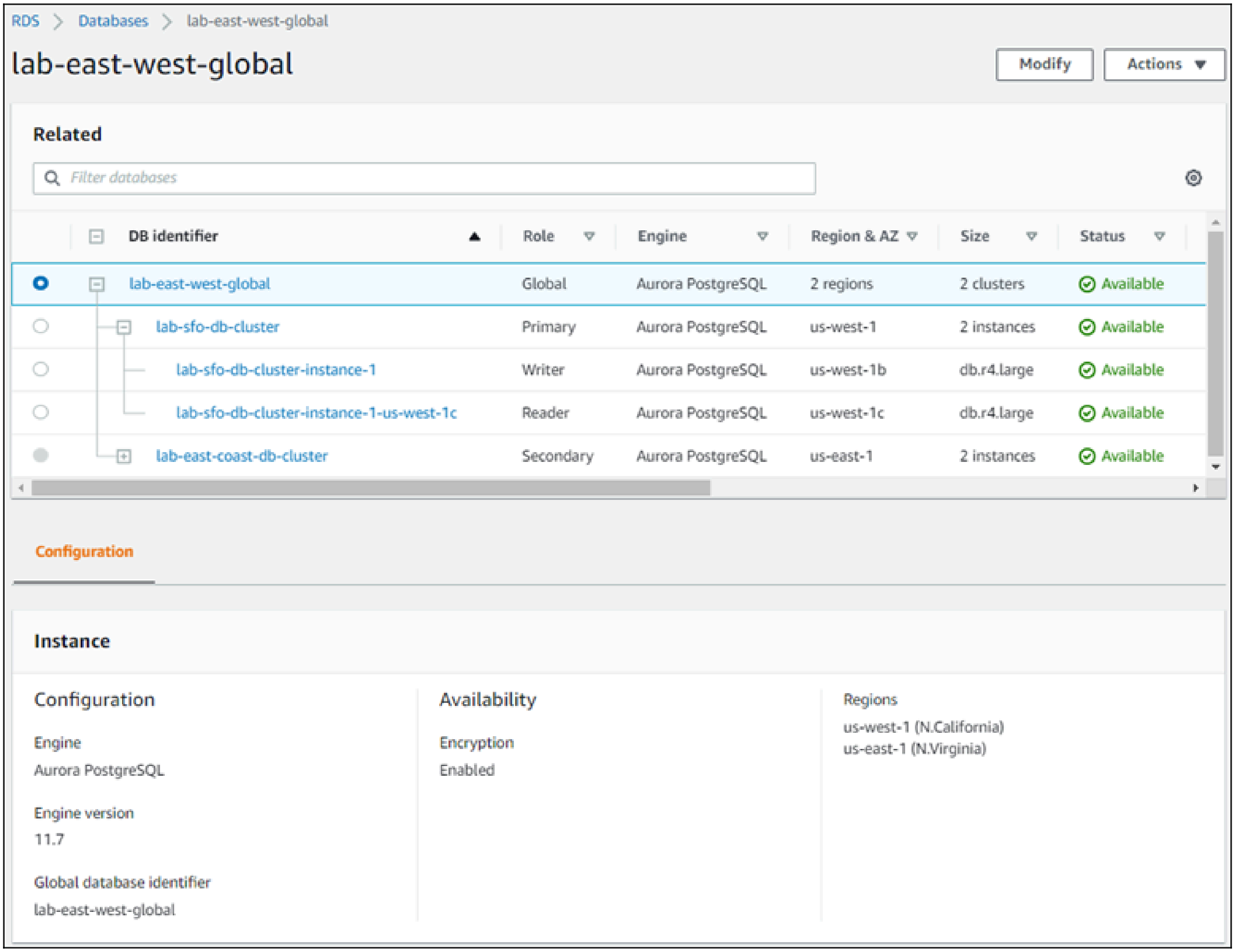
기본 리전에서 계획되지 않은 중단으로부터 Aurora 전역 데이터베이스를 복구하려면 계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구 단원을 참조하세요.

주제

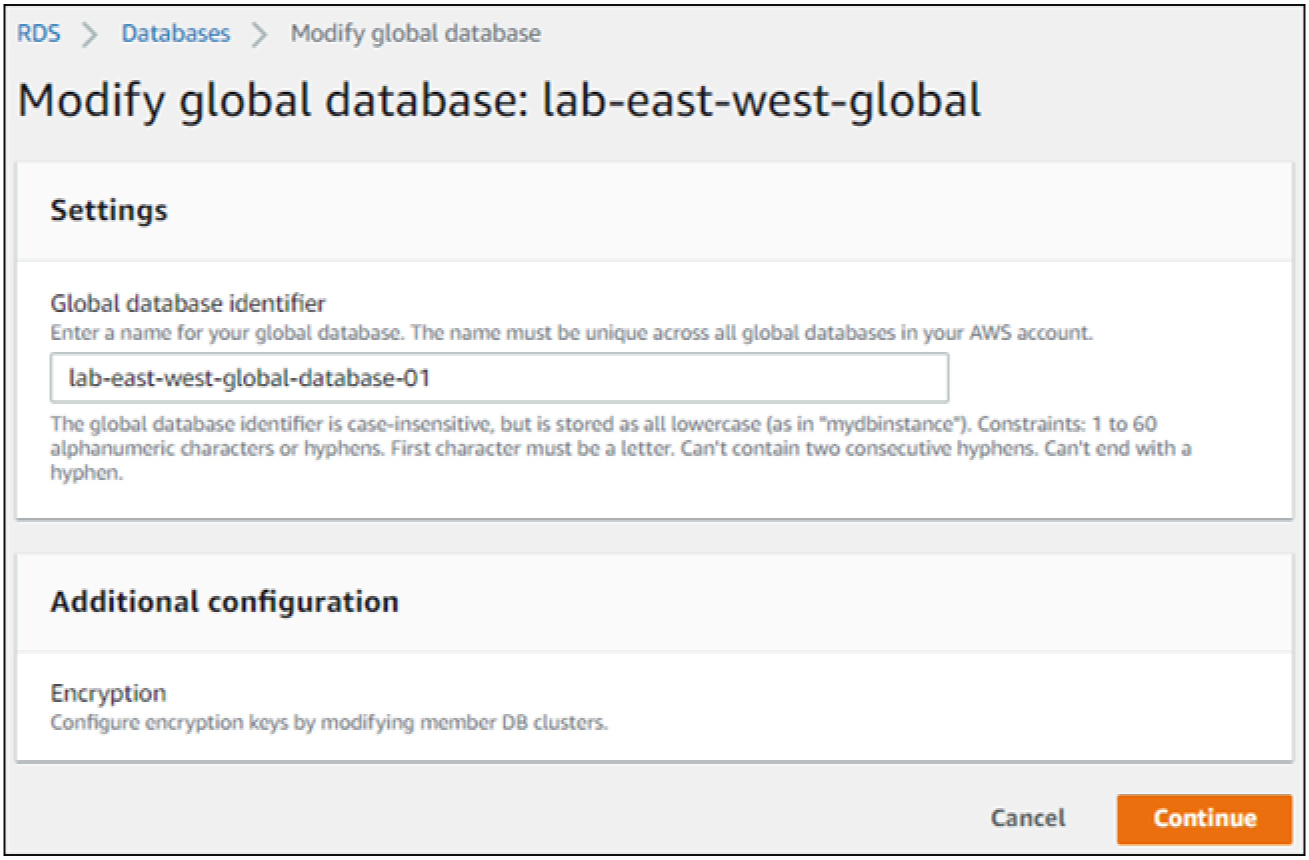
- Amazon Aurora 글로벌 데이터베이스 수정
- Aurora 글로벌 데이터베이스에 대한 파라미터 수정
- Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거
- Amazon Aurora 글로벌 데이터베이스 삭제
- Amazon Aurora Global Database 리소스 태그 지정

Amazon Aurora 글로벌 데이터베이스 수정

AWS Management Console의 [데이터베이스(Databases)] 페이지에 모든 Aurora Global Database가 나열되어 각각에 대한 기본 클러스터 및 보조 클러스터를 보여줍니다. Aurora 전역 데이터베이스에는 고유한 구성 설정이 있습니다. 특히, 다음 스크린샷과 같이 기본 및 보조 클러스터와 관련된 AWS 리전가 있습니다.



Aurora 글로벌 데이터베이스를 변경하면 다음 스크린 샷과 같이 변경 사항을 취소 할 수 있는 기회가 제공됩니다.



계속을 선택하면 변경 사항이 승인됩니다.

Aurora 글로벌 데이터베이스에 대한 파라미터 수정

Aurora 글로벌 데이터베이스 내에서 각 Aurora 클러스터에 대해 Aurora DB 클러스터 파라미터 그룹을 별도로 구성할 수 있습니다. 대부분의 파라미터는 다른 종류의 Aurora 클러스터와 동일하게 작동합니다. 전역 데이터베이스의 모든 클러스터 간에 설정을 일관성 있게 유지하는 것이 좋습니다. 이렇게 하면 보조 클러스터를 기본 클러스터로 승격하는 경우에 예상치 못한 동작 변경을 방지할 수 있습니다.

예를 들면 다른 클러스터가 기본 클러스터로 대신 사용되는 경우 일관되지 않은 동작을 방지하려면 시간대와 문자 세트에 대해 동일한 설정을 사용합니다.

`aurora_enable_repl_bin_log_filtering` 및 `aurora_enable_replica_log_compression` 구성 설정은 영향을 주지 않습니다.

Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거

여러 가지 이유로 Aurora 글로벌 데이터베이스에서 Aurora DB 클러스터를 제거할 수 있습니다. 예를 들어, 기본 클러스터가 성능이 저하되거나 격리된 경우 Aurora 글로벌 데이터베이스에서 Aurora DB 클러스터를 제거할 수 있습니다. 그런 다음, 프로비저닝된 독립형 Aurora DB 클러스터가 되어, 새로운

Aurora 전역 데이터베이스를 생성하는 데 사용할 수 있습니다. 자세한 내용은 계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구 단원을 참조하십시오.

더 이상 필요하지 않은 Aurora 전역 데이터베이스를 삭제하고자 하기 때문에 Aurora DB 클러스터를 제거하고 싶을 수도 있습니다. 연결된 모든 Aurora DB 클러스터를 제거(분리)하고 기본 클러스터를 마지막으로 남겨둘 때까지는 Aurora 전역 데이터베이스를 삭제할 수 없습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 삭제 섹션을 참조하세요.

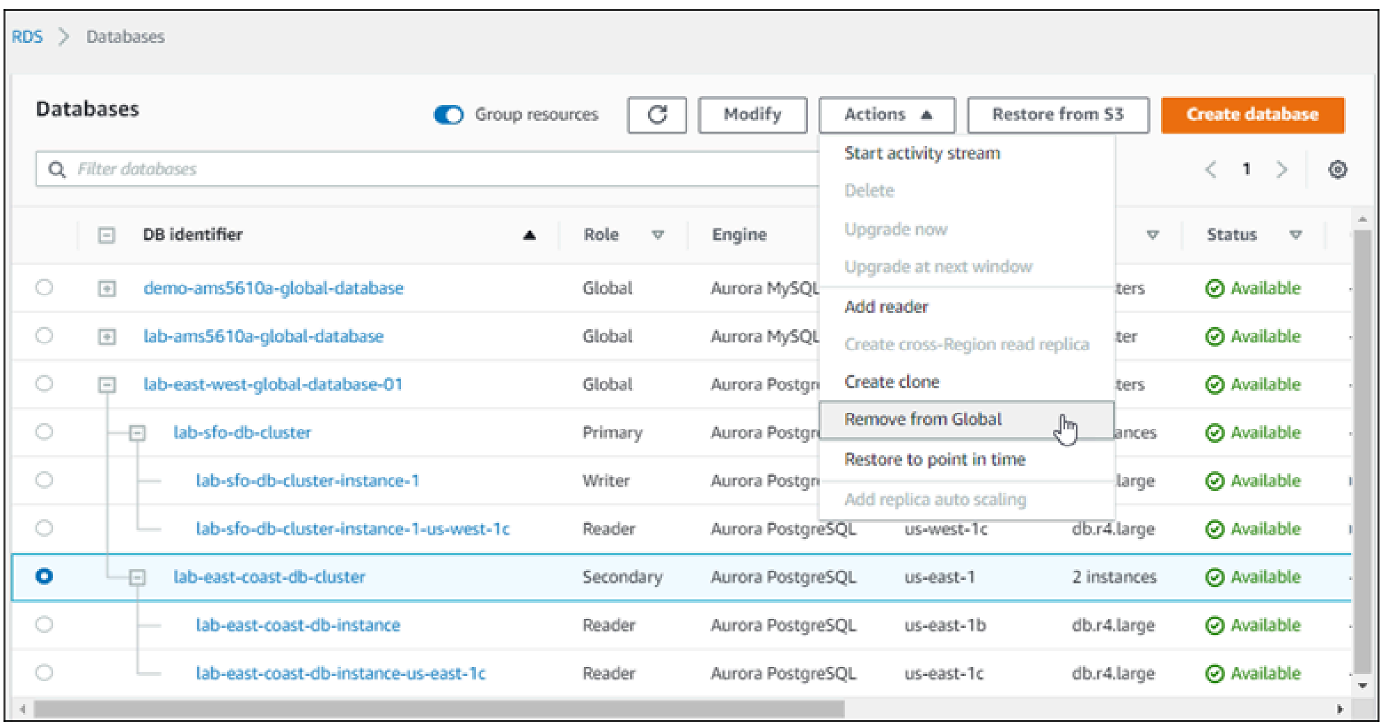
Aurora DB 클러스터가 Aurora 전역 데이터베이스에서 분리되면, 더 이상 기본 클러스터와 동기화되지 않습니다. 완전한 읽기/쓰기 기능을 갖춘 프로비저닝된 독립형 Aurora DB 클러스터가 됩니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora Global Database에서 Aurora DB 클러스터를 제거할 수 있습니다.

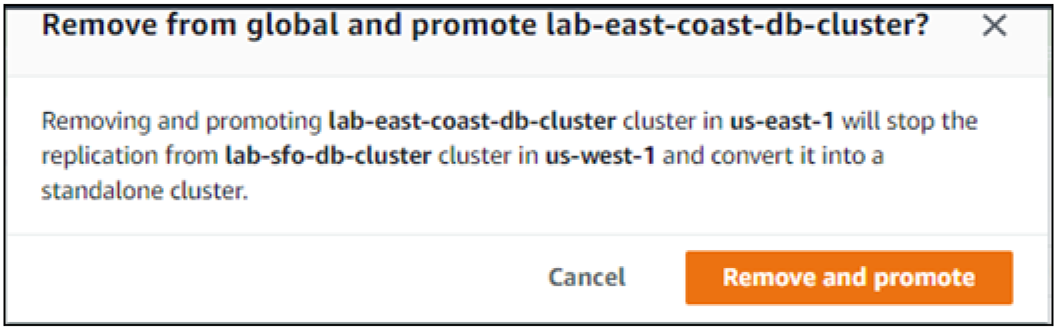
콘솔

Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스 페이지에서 클러스터를 선택합니다.
3. 작업에 대해 Remove from Global(글로벌에서 제거)을 선택합니다.

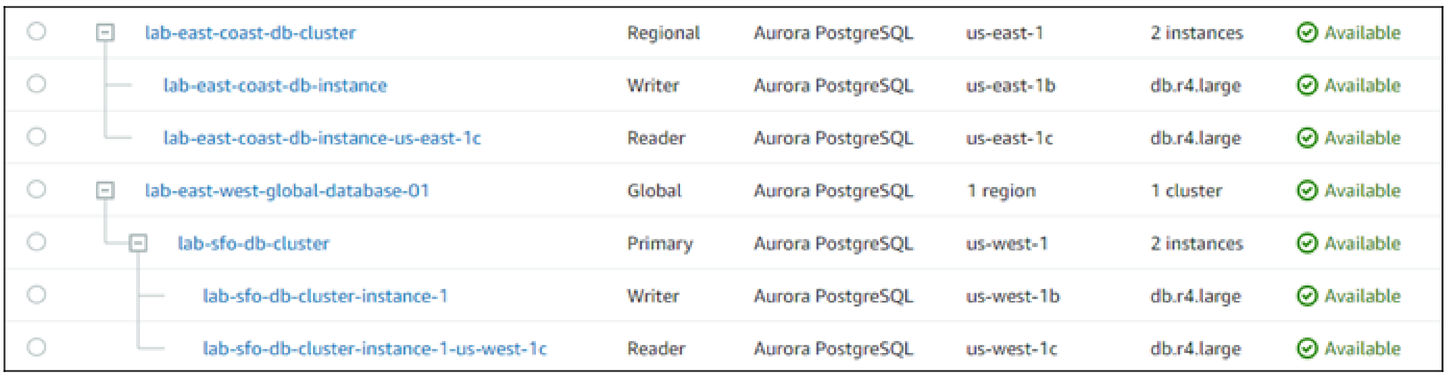


보조 데이터베이스를 Aurora 글로벌 데이터베이스에서 분리할지 확인하는 메시지가 표시됩니다.



4. 글로벌 데이터베이스에서 클러스터를 제거하려면 제거 및 승격을 선택합니다.

Aurora DB 클러스터는 더 이상 Aurora 글로벌 데이터베이스에서 보조 클러스터로 사용되지 않으며 더 이상 기본 DB 클러스터와 동기화되지 않습니다. 완전한 읽기/쓰기 기능을 갖춘 독립형 Aurora DB 클러스터입니다.



lab-east-coast-db-cluster	Regional	Aurora PostgreSQL	us-east-1	2 instances	Available
lab-east-coast-db-instance	Writer	Aurora PostgreSQL	us-east-1b	db.r4.large	Available
lab-east-coast-db-instance-us-east-1c	Reader	Aurora PostgreSQL	us-east-1c	db.r4.large	Available
lab-east-west-global-database-01	Global	Aurora PostgreSQL	1 region	1 cluster	Available
lab-sfo-db-cluster	Primary	Aurora PostgreSQL	us-west-1	2 instances	Available
lab-sfo-db-cluster-instance-1	Writer	Aurora PostgreSQL	us-west-1b	db.r4.large	Available
lab-sfo-db-cluster-instance-1-us-west-1c	Reader	Aurora PostgreSQL	us-west-1c	db.r4.large	Available

보조 클러스터를 모두 제거하거나 삭제한 후 동일한 방식으로 기본 클러스터를 제거할 수 있습니다. 모든 보조 클러스터를 제거해야 기본 Aurora DB 클러스터를 Aurora 글로벌 데이터베이스에서 분리 (제거) 할 수 있습니다.

Aurora 전역 데이터베이스는 0 리전 및 AZ가 있는 데이터베이스 목록에 남아 있을 수 있습니다. 이 Aurora 글로벌 데이터베이스를 더 이상 사용하지 않을 경우 삭제할 수 있습니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 삭제 섹션을 참조하세요.

AWS CLI

Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하려면 remove-from-global-cluster CLI 명령 및 다음 파라미터를 실행합니다.

- `--global-cluster-identifier` – Aurora 글로벌 데이터베이스의 이름 (식별자) 입니다.
- `--db-cluster-identifier` – Aurora 글로벌 데이터베이스에서 제거할 각 Aurora DB 클러스터의 이름입니다. 기본 DB 클러스터를 제거하기 전에 모든 보조 Aurora DB 클러스터를 제거합니다.

다음 명령은 Aurora 글로벌 데이터베이스에서 보조 클러스터를 제거한 후 기본 클러스터를 제거합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

Aurora Global Database의 각 보조 AWS 리전에 대해 `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`* 명령을 반복합니다.

Windows의 경우:

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

Aurora Global Database의 각 보조 `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`*에 대해 AWS 리전 명령을 반복합니다.

RDS API

RDS API를 사용하여 Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하려면 RemoveFromGlobalCluster 작업을 실행합니다.

Amazon Aurora 글로벌 데이터베이스 삭제

Aurora 글로벌 데이터베이스는 일반적으로 비즈니스에 중요한 데이터를 포함하므로, 글로벌 데이터베이스와 이 데이터베이스에 연결된 클러스터를 한 번에 삭제할 수는 없습니다. Aurora 전역 데이터베이스를 삭제하려면 다음을 수행합니다.

- Aurora 글로벌 데이터베이스에서 모든 보조 DB 클러스터를 제거합니다. 각 클러스터는 독립형 Aurora DB 클러스터가 됩니다. 자세한 방법은 Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거 단원을 참조하십시오.
- 각 독립형 Aurora DB 클러스터에서 모든 Aurora 복제본을 삭제합니다.
- Aurora 글로벌 데이터베이스에서 보조 클러스터를 제거합니다. 이 클러스터는 독립형 Aurora DB 클러스터가 됩니다.
- Aurora 기본 DB 클러스터에서 먼저 모든 Aurora 복제본을 삭제한 다음 라이터 DB 인스턴스를 삭제합니다.

새로 독립 실행형 Aurora DB 클러스터에서 작성기 인스턴스를 삭제하면 일반적으로 Aurora DB 클러스터와 Aurora 글로벌 데이터베이스도 제거됩니다.

더 일반적인 내용은 Aurora DB 클러스터에서 DB 인스턴스 삭제 단원을 참조하십시오.

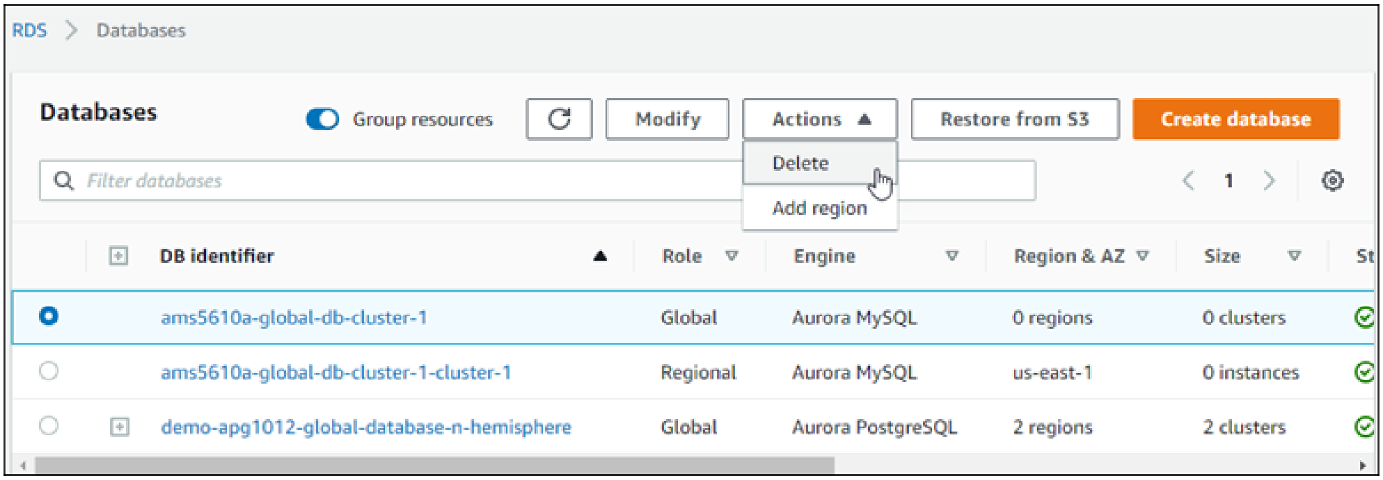
Aurora Global Database를 삭제하려면 AWS Management Console, AWS CLI 또는 RDS API를 사용할 수 있습니다.

콘솔

Aurora 글로벌 데이터베이스를 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 목록에서 삭제할 Aurora 글로벌 데이터베이스를 찾습니다.
3. Aurora 글로벌 데이터베이스에서 다른 모든 클러스터가 제거되었는지 확인합니다. Aurora 전역 데이터베이스에는 0 리전, AZ와 0 클러스터 크기가 표시되어야 합니다.

   Aurora 글로벌 데이터베이스에 Aurora DB 클러스터가 포함되어 있으면 삭제할 수 없습니다. 필요한 경우 Aurora 글로벌 데이터베이스에서 기본 및 보조 Aurora DB 클러스터를 분리합니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거 섹션을 참조하세요.
4. 목록에서 Aurora 글로벌 데이터베이스를 선택한 다음, 작업 메뉴에서 삭제를 선택합니다.



AWS CLI

Aurora Global Database를 삭제하려면 다음 예제와 같이 AWS 리전의 이름과 Aurora Global Database 식별자를 사용하여 [delete-global-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/delete-global-cluster.html) CLI 명령을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

Windows의 경우:

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS API

RDS API를 사용하여 Aurora 글로벌 데이터베이스의 일부로 포함된 클러스터를 삭제하려면 DeleteGlobalCluster API 작업을 실행합니다.

Amazon Aurora Global Database 리소스 태그 지정

Aurora 글로벌 데이터베이스 기능을 사용하면 RDS 태그를 글로벌 데이터베이스 내 다양한 수준의 리소스에 적용할 수 있습니다. AWS 또는 Aurora 리소스에서 태그를 사용하는 방법을 잘 모르는 경우 글로벌 데이터베이스 내에 태그를 적용하기 전에 Amazon Aurora 및 Amazon RDS 리소스에 태그 지정 섹션을 참조하세요.

> **Note**
>
> AWS는 태그 데이터를 비용 보고 메커니즘의 일부로 처리하기 때문에 태그 이름 또는 값에 민감한 데이터나 개인 식별 정보(PII)를 포함하지 마세요.

글로벌 데이터베이스 내에서 다음 종류의 리소스에 태그를 적용할 수 있습니다.

- 전체 글로벌 데이터베이스의 컨테이너 객체입니다. 이 리소스를 글로벌 클러스터라고 합니다.

  콘솔에서 AWS 리전 추가 작업을 수행하여 글로벌 클러스터를 생성한 후 글로벌 클러스터의 세부 정보 페이지를 사용하여 태그를 추가할 수 있습니다. 글로벌 클러스터 세부 정보 페이지의 태그 탭에서 태그 관리를 선택하여 태그 및 관련 값을 추가, 제거 또는 수정할 수 있습니다.

  AWS CLI 및 RDS API를 사용하면 글로벌 클러스터를 생성하는 동시에 글로벌 클러스터에 태그를 추가할 수 있습니다. 기존 글로벌 클러스터에 태그를 추가, 제거 또는 수정할 수도 있습니다.
- 기본 클러스터입니다. 여기에서 태그 작업에 독립 실행형 Aurora 클러스터와 동일한 절차를 사용합니다. 원본 Aurora 클러스터를 글로벌 데이터베이스로 전환하기 전에 태그를 설정할 수 있습니다. DB 클러스터 세부 정보 페이지 내의 태그 탭에서 태그 관리를 선택하여 태그 및 관련 값을 추가, 제거 또는 수정할 수도 있습니다.
- 모든 보조 클러스터입니다. 여기에서 태그 작업에 독립 실행형 Aurora 클러스터와 동일한 절차를 사용합니다. 콘솔에서 AWS 리전 추가 작업을 사용하여 보조 Aurora 클러스터를 생성하는 동시에 태그를 설정할 수 있습니다. DB 클러스터 세부 정보 페이지 내의 태그 탭에서 태그 관리를 선택하여 태그 및 관련 값을 추가, 제거 또는 수정할 수도 있습니다.
- 기본 또는 보조 클러스터 내의 개별 DB 인스턴스입니다. 여기에서 태그 작업에 Aurora 또는 RDS DB 인스턴스와 동일한 절차를 사용합니다. 콘솔의 리더 추가 작업을 사용하여 Aurora 클러스터에 새 DB 인스턴스를 추가하는 동시에 태그를 설정할 수 있습니다. DB 인스턴스 세부 정보 페이지 내의 태그 탭에서 태그 관리를 선택하여 태그 및 관련 값을 추가, 제거 또는 수정할 수도 있습니다.

다음은 글로벌 데이터베이스 내에서 할당할 수 있는 태그의 몇 가지 예입니다.

- 글로벌 클러스터에 태그를 추가하여 조직 내 소유자 및 연락처를 나타내는 익명화된 식별자 등 애플리케이션에 대한 전체 정보를 기록할 수 있습니다. 태그를 사용하여 글로벌 데이터베이스를 사용하는 애플리케이션의 속성을 나타낼 수 있습니다.
- 기본 클러스터와 보조 클러스터에 태그를 추가하여 AWS 리전 수준에서 애플리케이션의 비용을 추적할 수 있습니다. 해당 절차에 대한 자세한 내용은 [Amazon RDS에서 태그와 함께 AWS 결제가 작동하는 방식](#) 섹션을 참조하세요.

- Aurora 클러스터를 사용하여 특정 DB 인스턴스에 태그를 추가하여 특수 용도를 나타낼 수 있습니다. 예를 들어 기본 클러스터 내에서 보고서 생성에만 사용되는 장애 조치 우선 순위가 낮은 리더 인스턴스가 있을 수 있습니다. 태그는 이 특수 용도의 DB 인스턴스를 기본 클러스터 내에서 고가용성 전용인 다른 인스턴스와 구별할 수 있습니다.
- 글로벌 데이터베이스 리소스의 모든 수준에서 태그를 사용하여 IAM 정책을 통한 액세스를 제어할 수 있습니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 AWS 리소스에 대한 액세스 제어를 참조하세요.

> **Tip**
>
> AWS Management Console에서 글로벌 클러스터 컨테이너를 생성한 후 별도의 단계로 태그를 추가합니다. 글로벌 클러스터가 액세스 제어 태그 없이 존재하는 시간 간격을 피하려면, AWS CLI, RDS API 또는 AWS CloudFormation 템플릿을 통해 해당 리소스를 생성하여 `CreateGlobalCluster` 작업 중에 태그를 적용할 수 있습니다.

- 클러스터 수준에서 태그를 사용하거나 글로벌 클러스터에 태그를 사용하여 애플리케이션의 품질 보증 및 테스트에 대한 정보를 기록할 수 있습니다. 예를 들어 DB 클러스터에 태그를 지정하여 해당 클러스터로 마지막으로 전환을 수행한 시간을 기록할 수 있습니다. 글로벌 클러스터에 태그를 지정하여 전체 애플리케이션에 대한 마지막 재해 복구 훈련 시간을 기록할 수 있습니다.

글로벌 데이터베이스에 대한 태그 지정 AWS CLI 예제

다음 AWS CLI 예제에서는 글로벌 데이터베이스에서 모든 유형의 Aurora 리소스에 대한 태그를 지정하고 검사하는 방법을 보여줍니다.

`create-global-cluster` 명령을 사용하여 글로벌 클러스터 컨테이너의 태그를 지정할 수 있습니다. 다음 예제에서는 글로벌 클러스터를 생성하고 두 개의 태그를 할당합니다. 태그에는 `tag1` 및 `tag2` 키가 있습니다.

```
$  aws rds create-global-cluster --global-cluster-identifier my_global_cluster_id \
  --engine aurora-mysql --tags Key=tag1,Value=val1 Key=tag2,Value=val2
```

`describe-global-clusters` 명령을 사용하여 글로벌 클러스터 컨테이너에 태그를 나열할 수 있습니다. 태그 작업 시 일반적으로 이 명령을 먼저 실행하여 글로벌 클러스터의 Amazon 리소스 이름(ARN)을 검색합니다. ARN을 태그 작업을 위한 후속 명령의 파라미터로 사용합니다. 다음 명령은 `TagList` 속성의 태그 정보를 표시합니다. 또한 이후 예제에서 파라미터로 사용되는 ARN도 보여줍니다.

```
$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "GlobalClusterArn": "my_global_cluster_arn",
            ...
            "TagList": [
                {
                    "Value": "val1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                }
            ]
        }
    ]
}
```

add-tags-to-resource 명령을 사용하여 새 태그를 추가할 수 있습니다. 이 명령을 사용하면 식별자 대신 글로벌 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다. 기존 태그와 동일한 이름이 있는 태그를 추가하면 해당 태그의 값을 덮어씁니다. 태그 값에 공백 또는 특수 문자를 포함하는 경우 운영 체제 또는 명령 셸에 적합한 값을 따옴표로 표시합니다. 다음 예제에서는 이전 예제에서 글로벌 클러스터의 태그를 수정합니다. 원래 클러스터에는 tag1 및 tag2 키가 있는 태그가 있었습니다. 명령이 완료되면 글로벌 클러스터에 tag3 키가 있는 새 태그가 생기고 tag1 키가 있는 태그는 다른 값을 갖습니다.

```
$  aws rds add-tags-to-resource --resource-name my_global_cluster_arn \
  --tags Key=tag1,Value="new value for tag1" Key=tag3,Value="entirely new tag"

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
```

```
                    "Value": "new value for tag1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                },
                {
                    "Value": "entirely new tag",
                    "Key": "tag3"
                }
            ]
        }
    ]
}
```

remove-tags-from-resource 명령을 사용하여 글로벌 클러스터에서 태그를 삭제할 수 있습니다. 이 명령을 사용하면 태그 값 없이 태그 키 집합만 지정합니다. 다음 예제에서는 이전 예제에서 글로벌 클러스터의 태그를 수정합니다. 원래 클러스터에는 tag1, tag2 및 tag3 키가 있는 태그가 있었습니다. 명령이 완료되면 tag1 키가 있는 태그만 유지됩니다.

```
$  aws rds remove-tags-from-resource --resource-name my_global_cluster_arn --tag-keys
 tag2 tag3

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                }
            ]
        }
    ]
}
```

Amazon Aurora Global Database에 연결

각 Aurora Global Database에는 기본 DB 클러스터의 현재 라이터 인스턴스로 요청을 라우팅하기 위해 Aurora에서 자동으로 업데이트하는 라이터 엔드포인트가 함께 제공됩니다. 라이터 엔드포인트를 사용하면 관리형 Aurora Global Database 전환 및 장애 조치 기능을 사용하여 기본 리전의 위치를 변경한 후 연결 문자열을 수정할 필요가 없습니다. Aurora Global Database 전환 및 장애 조치와 함께 라이터 엔드포인트를 사용하는 방법에 대한 자세한 내용은 Amazon Aurora Global Database에서 전환 또는 장애 조치 사용 섹션을 참조하세요. RDS 프록시를 사용하여 Aurora Global Database에 연결하는 방법에 대한 자세한 내용은 Aurora Global Database에 RDS 프록시 사용을 참조하세요.

주제

- 애플리케이션 요구 사항을 충족하는 엔드포인트 선택
- Amazon Aurora Global Database의 엔드포인트 보기
- 글로벌 라이터 엔드포인트 사용 시 고려 사항

애플리케이션 요구 사항을 충족하는 엔드포인트 선택

Aurora Global Database에 연결하려면 데이터베이스와 요청을 라우팅하려는 AWS 리전에서 읽거나 쓸 필요가 있어야 합니다. 다음은 몇 가지 일반적인 사용 사례입니다.

- 라이터 인스턴스로 요청 라우팅: 데이터 조작 언어(DML) 및 데이터 정의 언어(DDL) 문을 실행해야 하거나 읽기와 쓰기 간에 강력한 일관성이 필요한 경우 Aurora Global Database 라이터 엔드포인트에 연결합니다. 이 엔드포인트는 요청을 글로벌 데이터베이스의 기본 클러스터에 있는 라이터 인스턴스로 라우팅합니다. 이 엔드포인트는 자동으로 업데이트되어 라이터 인스턴스로 요청을 라우팅하므로 글로벌 클러스터에서 라이터 위치를 변경할 때마다 애플리케이션을 업데이트할 필요가 없습니다. 글로벌 엔드포인트를 사용하여 리전 간 읽기/쓰기 요청을 라이터에게 보낼 수도 있습니다.

> **Note**
>
> Aurora Global Database 라이터 엔드포인트를 사용할 수 있기 전에 글로벌 데이터베이스를 설정한 경우 애플리케이션이 기본 클러스터의 클러스터 엔드포인트에 연결될 수 있습니다. 이 경우 대신 글로벌 라이터 엔드포인트를 사용하도록 연결 설정을 전환하는 것이 좋습니다. 이렇게 하면 Aurora Global Database 전환 또는 장애 조치 후에 매번 연결 설정을 변경할 필요가 없습니다.

> 라이터 엔드포인트 이름의 첫 번째 부분은 Aurora Global Database의 이름입니다. 따라서 Aurora Global Database의 이름을 바꾸면 라이터 엔드포인트 이름이 변경되고 이를 사용하는 모든 코드가 새 이름으로 업데이트되어야 합니다.

- 애플리케이션 리전에 더 가까운 읽기 규모 조정: 애플리케이션과 동일하거나 가까운 AWS 리전에서 읽기 전용 요청을 조정하려면 기본 또는 보조 Aurora 클러스터의 리더 엔드포인트에 연결합니다.
- 간헐적 리전 간 쓰기를 사용하여 읽기 규모 조정: 유지 관리 및 데이터 정리와 같은 간헐적 DML 문의 경우 쓰기 전달이 활성화된 보조 클러스터의 리더 엔드포인트에 연결합니다. 쓰기 전달을 사용하면 Aurora는 쓰기 문을 Aurora Global Database의 기본 리전에 있는 라이터에 자동으로 전달합니다. 쓰기 전달은 다음과 같은 이점을 제공합니다.
  - 리전 간 쓰기를 전송하기 위해 보조 클러스터와 기본 클러스터 간에 연결을 설정하는 번거로운 작업을 수행할 필요가 없습니다.
  - 애플리케이션에서 읽기 및 쓰기 요청을 분할할 필요가 없습니다.
  - 읽기 후 쓰기 요청의 일관성을 관리하기 위해 복잡한 로직을 개발할 필요가 없습니다.

  그러나 쓰기 전달을 사용하면 리전 간 장애 조치 또는 전환을 수행한 후 새로 승격된 기본 리전의 리더 엔드포인트에 연결하기 위해 애플리케이션 코드 또는 구성을 업데이트해야 합니다. 쓰기 요청을 처리하는 오버헤드를 확인하려면 쓰기 전달을 통해 수행되는 작업의 지연 시간을 모니터링하는 것이 좋습니다. 마지막으로 쓰기 전달은 데이터 정의 언어(DDL) 변경 또는 `SELECT FOR UPDATE` 문과 같은 특정 MySQL 또는 PostgreSQL 작업을 지원하지 않습니다.

  AWS 리전 간 쓰기 전달 사용에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 섹션을 참조하세요.

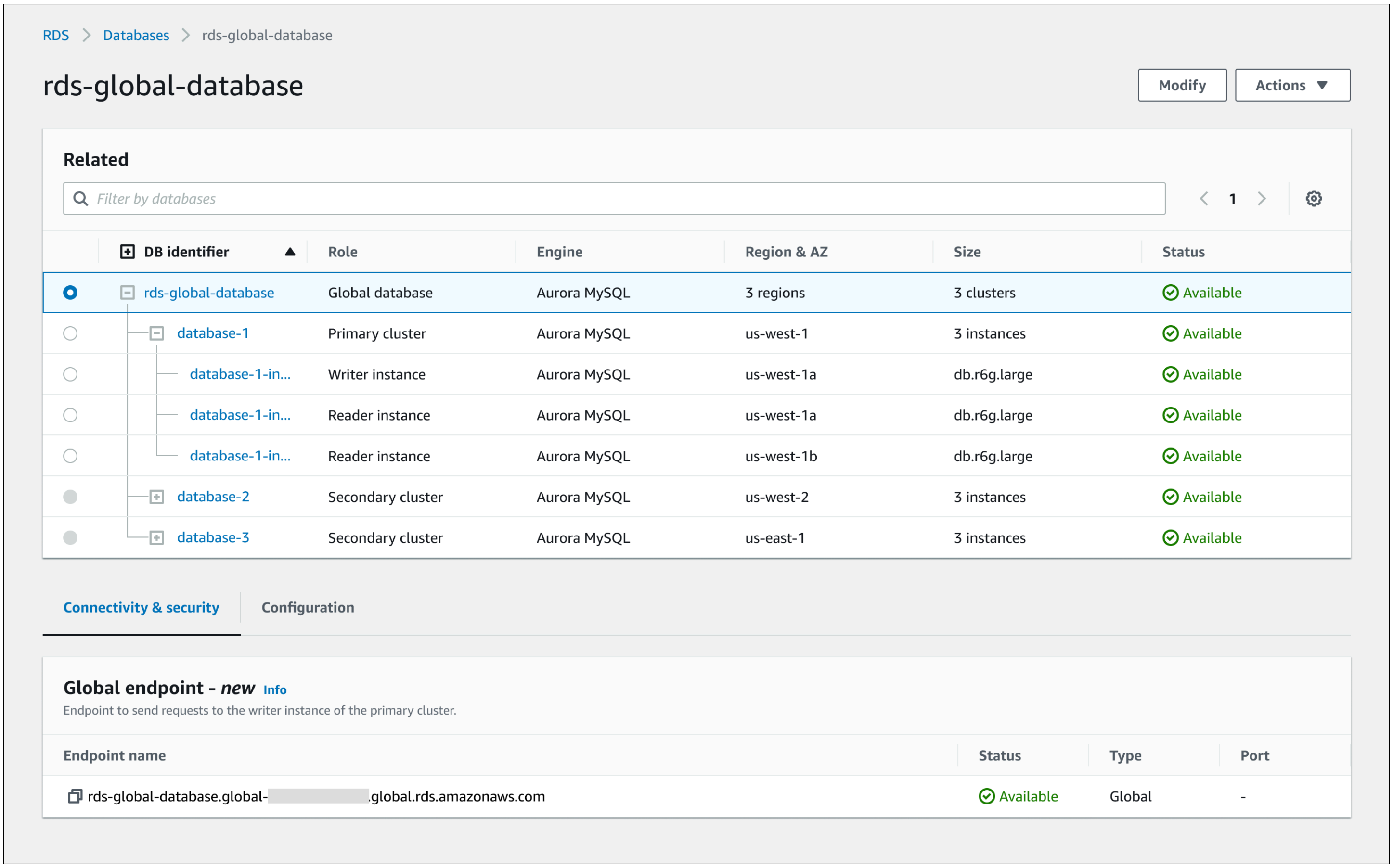
다양한 종류의 Aurora 엔드포인트에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 섹션을 참조하세요.

Amazon Aurora Global Database의 엔드포인트 보기

콘솔에서 Aurora Global Database를 볼 때 모든 관련 클러스터와 연결된 엔드포인트를 모두 볼 수 있습니다. 다음 그림은 기본 DB 클러스터의 세부 정보를 볼 때 표시되는 엔드포인트 유형의 예를 보여줍니다.

- 글로벌 라이터 - 항상 글로벌 데이터베이스 클러스터의 현재 라이터 DB 인스턴스를 가리키는 단일 읽기/쓰기 엔드포인트입니다.
- 라이터 - 글로벌 데이터베이스 클러스터의 기본 DB 클러스터에 대한 읽기/쓰기 요청을 위한 연결 엔드포인트입니다.

- 리더 - 글로벌 데이터베이스 클러스터의 기본 또는 보조 DB 클러스터에 대한 읽기 전용 요청을 위한 연결 엔드포인트입니다. 지연 시간을 최소화하려면 자신의 AWS 리전이나 가장 가까운 AWS 리전의 리더 엔드포인트를 선택합니다.



콘솔

글로벌 데이터베이스의 엔드포인트를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 목록에서 글로벌 데이터베이스 또는 엔드포인트를 보려는 기본 또는 보조 DB 클러스터를 선택합니다.
4. 연결 및 보안 탭을 선택하여 엔드포인트 세부 정보를 확인합니다. 표시되는 엔드포인트는 다음과 같이 선택한 클러스터 유형에 따라 달라집니다.

   - 글로벌 데이터베이스 - 글로벌 라이터 엔드포인트입니다.

- 기본 DB 클러스터 - 글로벌 라이터 엔드포인트와 기본 클러스터의 클러스터 엔드포인트 및 리더 엔드포인트입니다.
- 보조 DB 클러스터 - 보조 클러스터의 클러스터 엔드포인트 및 리더 엔드포인트입니다. 보조 클러스터에서 클러스터 엔드포인트는 쓰기 요청을 처리하지 않으므로 비활성 상태를 표시합니다. 읽기 쿼리에 대해서만 클러스터 엔드포인트에 연결할 수 있습니다.

AWS CLI

글로벌 클러스터의 라이터 엔드포인트를 보려면 다음 예제와 같이 AWS CLI describe-global-clusters 명령을 사용합니다.

```
aws rds describe-global-clusters --region aws_region
{
    "GlobalClusters": [
        {
            "GlobalClusterIdentifier": "global_cluster_id",
            "GlobalClusterResourceId": "cluster-unique_string",
            "GlobalClusterArn": "arn:aws:rds::123456789012:global-
cluster:global_cluster_id",
            "Status": "available",
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",
            "GlobalClusterMembers": [

              ...
            ],
            "Endpoint":
 "global_cluster_id.global-unique_string.global.rds.amazonaws.com"
        }
    ]
}
```

글로벌 클러스터의 멤버 DB 클러스터에 대한 클러스터 및 리더 엔드포인트를 보려면 다음 예제와 같이 AWS CLI describe-db-clusters 명령을 사용합니다. `Endpoint` 및 `ReaderEndpoint`에 대해 반환된 값은 각각 클러스터 및 리더 엔드포인트입니다.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
{
    "DBClusters": [
        {
```

```
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "az_1",
                "az_2",
                "az_3"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "db_cluster_id",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-08-01T18:21:11.301Z",
            "Endpoint":
 "db_cluster_id.cluster-unique_string.primary_region.rds.amazonaws.com",
            "ReaderEndpoint": "db_cluster_id.cluster-
ro-unique_string.primary_region.rds.amazonaws.com",
            "MultiAZ": false,
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",

            "ReadReplicaIdentifiers": [
                "arn:aws:rds:secondary_region:123456789012:cluster:db_cluster_id"
            ],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "db_instance_id",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                }
            ],

            ...
            "TagList": [],
            "GlobalWriteForwardingRequested": false
        }
    ]
}
```

RDS API

글로벌 클러스터의 라이터 엔드포인트를 보려면 RDS API DescribeGlobalClusters 작업을 사용합니다. 글로벌 클러스터의 멤버 DB 클러스터에 대한 클러스터 및 리더 엔드포인트를 보려면 RDS API DescribeDBClusters 작업을 사용합니다.

글로벌 라이터 엔드포인트 사용 시 고려 사항

다음 지침과 모범 사례에 따라 Aurora Global Database 라이터 엔드포인트를 효과적으로 사용할 수 있습니다.

- 리전 간 장애 조치 또는 전환 후 중단을 최소화하기 위해 애플리케이션 컴퓨팅과 기본 및 보조 AWS 리전 간에 VPC 연결을 설정할 수 있습니다. 예를 들어 기본 클러스터와 동일한 VPC에서 실행 중인 애플리케이션 또는 클라이언트 시스템이 있다고 가정해 보겠습니다. 보조 클러스터가 승격되면 전역 라이터 엔드포인트가 해당 클러스터를 가리키도록 자동으로 변경됩니다. 글로벌 라이터 엔드포인트를 사용하면 애플리케이션의 연결 설정을 변경할 수 없지만, 두 VPC 간에 네트워킹을 설정할 때까지 애플리케이션은 새로 승격된 기본 AWS 리전의 VPC에서 IP 주소에 액세스할 수 없습니다. 이 연결을 설정하기 위한 다양한 옵션을 평가하려면 Amazon VPC와 Amazon VPC 간 연결 옵션을 참조하세요.
- 글로벌 데이터베이스 장애 조치 또는 전환 후 글로벌 라이터 엔드포인트 업데이트는 도메인 이름 서비스(DNS) 캐싱 기간에 따라 시간이 오래 걸릴 수 있습니다. 자세한 내용은 Amazon Aurora MySQL Database Administrator's Handbook을 참조하세요. Aurora Global Database는 글로벌 라이터 엔드포인트에서 DNS 변경 사항이 표시되면 RDS 이벤트를 내보냅니다. 이벤트를 사용하여 이벤트가 생성된 후 기간이 지나도 DNS 캐시가 연장되지 않도록 전략을 수립할 수 있습니다. 자세한 내용은 DB 클러스터 이벤트 섹션을 참조하세요.
- Aurora Global Database는 데이터를 비동기적으로 복제합니다. 리전 간 장애 조치 방법을 사용하면 장애 조치가 시작되기 전에 선택한 보조에 복제되지 않은 일부 쓰기 트랜잭션 데이터가 발생할 수 있습니다. Aurora는 원래 기본 AWS 리전에서 쓰기를 차단하기 위해 최선의 노력을 기울일지라도 장애 조치는 브레인 분할 문제에 취약할 수 있습니다. 데이터 손실 및 브레인 분할 위험을 최소화하기 위한 고려 사항은 Aurora Global Database 라이터 엔드포인트에도 적용됩니다. 자세한 내용은 Aurora Global Database에서 계획된 관리형 장애 조치 수행 섹션을 참조하세요.

Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용

쓰기 전달을 사용하여 Aurora 글로벌 데이터베이스에서 실행되는 애플리케이션에 대해 관리해야 하는 엔드포인트 수를 줄일 수 있습니다. 쓰기 전달을 활성화하면 Aurora 글로벌 데이터베이스의 보조 클러

스터가 쓰기 작업을 수행하는 SQL 문을 기본 클러스터로 전달할 수 있습니다. 기본 클러스터는 원본을 업데이트한 다음 결과 변경 사항을 모든 보조 AWS 리전으로 다시 전파합니다.

쓰기 전달 구성을 사용하면 보조 AWS 리전에서 기본 리전으로 쓰기 작업을 전송하기 위한 자체 매커니즘을 구현할 필요가 없습니다. Aurora는 교차 지역 네트워킹 설정을 처리합니다. 또한 Aurora는 각 문에 필요한 모든 세션 및 트랜잭션 컨텍스트를 전송합니다. 데이터는 항상 기본 클러스터에서 먼저 변경된 다음 Aurora 글로벌 데이터베이스의 보조 클러스터로 다시 복제됩니다. 이러한 방식으로, 기본 클러스터는 항상 모든 데이터의 최신 복사본이 있는 진정한 원본입니다.

주제

Aurora MySQL 글로벌 데이터베이스에서 쓰기 전달 사용

주제

Aurora MySQL의 쓰기 전달을 사용할 수 있는 리전 및 버전

쓰기 전달은 Aurora MySQL 기반 글로벌 데이터베이스를 사용할 수 있는 모든 리전에서 Aurora MySQL 2.08.1 이상 버전에 지원됩니다.

Aurora MySQL 글로벌 데이터베이스의 버전 및 리전 가용성에 대한 자세한 정보는 Aurora MySQL을 사용하는 Aurora 전역 데이터베이스 섹션을 참조하세요.

Aurora MySQL에서 쓰기 전달 활성화

기본적으로 Aurora Global Database에 보조 클러스터를 추가할 때는 쓰기 전달이 활성화되지 않습니다.

AWS Management Console을 사용하여 쓰기 전달을 활성화하려면 글로벌 데이터베이스에 대해 리전을 추가할 때 읽기 전용 복제본 쓰기 전달에서 읽기 전용 복제본 쓰기 전달 켜기 확인란을 선택합니다. 기존 보조 클러스터의 경우 전역 쓰기 전달 켜기로 클러스터를 수정합니다. 쓰기 전달을 끄려면 리전을 추가하거나 보조 클러스터를 수정할 때 전역 쓰기 전달 켜기 확인란의 선택을 취소합니다.

AWS CLI를 사용하여 쓰기 전달을 활성화하려면 `--enable-global-write-forwarding` 옵션을 사용합니다. 이 옵션은 `create-db-cluster` 명령을 사용하여 새 보조 클러스터를 생성할 때 작동합니다. 이 옵션은 `modify-db-cluster` 명령을 사용하여 기존 보조 클러스터를 수정할 때도 작동합니다. 이렇게 하려면 글로벌 데이터베이스에서 쓰기 전달을 지원하는 Aurora 버전을 사용해야 합니다. 이러한 동일한 CLI 명령과 함께 `--no-enable-global-write-forwarding` 옵션을 사용하여 쓰기 전달을 끌 수 있습니다.

Amazon RDS API를 사용하여 쓰기 전달을 활성화하려면 `EnableGlobalWriteForwarding` 파라미터를 `true`로 설정합니다. 이 파라미터는 `CreateDBCluster` 작업을 사용하여 새 보조 클러스터를 생성할 때 작동합니다. 이 파라미터는 `ModifyDBCluster` 작업을 사용하여 기존 보조 클러스터를 수정할 때도 작동합니다. 이렇게 하려면 글로벌 데이터베이스에서 쓰기 전달을 지원하는 Aurora 버전을 사용해야 합니다. `EnableGlobalWriteForwarding` 파라미터를 `false`로 설정하여 쓰기 전달을 끌 수 있습니다.

> **Note**
>
> 데이터베이스 세션에서 쓰기 전달을 사용하기 위해 `aurora_replica_read_consistency` 구성 파라미터에 대한 설정을 지정합니다. 쓰기 전달 기능을 사용하는 모든 세션에서 이 작업을 수행합니다. 이 파라미터에 대한 자세한 정보는 [Aurora MySQL에서 쓰기 전달의 격리 및 일관성](#)를 참조하십시오.
> RDS 프록시 기능은 `aurora_replica_read_consistency` 변수에 대한 `SESSION` 값을 지원하지 않습니다. 이 값을 설정하면 예상하지 못한 동작이 일어날 수 있습니다.

다음 CLI 예제에서는 쓰기 전달이 활성화되거나 비활성화된 상태로 Aurora 글로벌 데이터베이스를 설정하는 방법을 보여 줍니다. 강조 표시된 항목은 Aurora 글로벌 데이터베이스에 대한 인프라를 설정할 때 지정하고 일관성을 유지해야 할 중요한 명령 및 옵션을 나타냅니다.

다음 예제에서는 쓰기 전달이 활성화된 상태로 Aurora 글로벌 데이터베이스, 기본 클러스터 및 보조 클러스터를 생성합니다. 사용자 이름, 암호, 기본 및 보조 AWS 리전을 직접 선택한 항목으로 대체합니다.

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
```

```
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

다음 예제는 이전 예제에서 계속됩니다. 쓰기 전달이 활성화되지 않은 상태로 보조 클러스터를 생성한 다음, 쓰기 전달을 활성화합니다. 이 예제가 완료된 후에는 글로벌 데이터베이스의 모든 보조 클러스터에 쓰기 전달이 활성화됩니다.

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

Aurora MySQL에서 보조 클러스터에 쓰기 전달이 활성화되어 있는지 확인

보조 클러스터에서 쓰기 전달을 사용할 수 있는지 여부를 확인하려면 클러스터에 `"GlobalWriteForwardingStatus": "enabled"` 특성이 있는지 여부를 확인할 수 있습니다.

AWS Management Console에서 클러스터의 세부 정보 페이지 구성 탭에서 전역 읽기 복제본 쓰기 전달이 활성화됨 상태인 것을 볼 수 있습니다.

모든 클러스터에 대한 글로벌 쓰기 전달 설정의 상태를 보려면 다음 AWS CLI 명령을 실행합니다.

보조 클러스터에는 쓰기 전달이 켜져 있는지 또는 꺼져 있는지를 나타내는 "enabled" 또는 "disabled" 값이 표시됩니다. null 값은 해당 클러스터에 쓰기 전달을 사용할 수 없음을 나타냅니다. 클러스터가 글로벌 데이터베이스의 일부가 아니거나, 보조 클러스터 대신 기본 클러스터입니다. 또한 쓰기 전달이 켜지거나 꺼지는 중인 경우 이 값은 "enabling" 또는 "disabling"일 수 있습니다.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu

[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

글로벌 쓰기 전달이 활성화된 모든 보조 클러스터를 찾으려면 다음 명령을 실행하십시오. 이 명령은 또한 클러스터의 판독기 엔드포인트를 반환합니다. Aurora Global Database에서 보조 클러스터에서 기본 클러스터로의 쓰기 전달을 사용할 때는 보조 클러스터의 읽기 장치 엔드포인트를 사용합니다.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
```

```
]
```

Aurora MySQL에서 쓰기 전달과 애플리케이션 및 SQL 호환성

다음과 같은 종류의 SQL 문을 쓰기 전달과 함께 사용할 수 있습니다.

- INSERT, DELETE 및 UPDATE와 같은 데이터 조작 언어(DML) 문입니다. 다음 설명과 같이, 쓰기 전달과 함께 사용할 수 있는 이러한 명령문의 속성에는 몇 가지 제한 사항이 있습니다.
- SELECT ... LOCK IN SHARE MODE 및 SELECT FOR UPDATE 문입니다.
- PREPARE 및 EXECUTE 문입니다.

특정 명령문은 쓰기 전달과 함께 글로벌 데이터베이스에서 사용할 때 허용되지 않거나 부실 결과를 생성할 수 있습니다. 따라서 EnableGlobalWriteForwarding 설정은 보조 클러스터에 대해 기본적으로 꺼져 있습니다. 이 설정을 켜기 전에, 애플리케이션 코드가 이러한 제한 사항의 영향을 받지 않는지 확인하십시오.

쓰기 전달에 사용하는 SQL 문에는 다음과 같은 제한 사항이 적용됩니다. 일부 경우에는 클러스터 수준에서 쓰기 전달이 활성화된 보조 클러스터에서 명령문을 사용할 수 있습니다. 이 접근 방식은 aurora_replica_read_consistency 구성 파라미터를 통해 세션 내에서 쓰기 전달을 켜지 않은 경우에 작동합니다. 쓰기 전달 때문에 명령문이 허용되지 않을 때 명령문을 사용하려고 시도하면 다음과 같은 형식의 오류 메시지가 나타납니다.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

데이터 정의 언어(DDL)

기본 클러스터에 연결하여 DDL 문을 실행합니다. 리더 DB 인스턴스에서 실행할 수 없습니다.

임시 테이블의 데이터를 사용하여 영구 테이블 업데이트

쓰기 전달이 활성화된 보조 클러스터에서 임시 테이블을 사용할 수 있습니다. 하지만 명령문이 임시 테이블을 참조하는 경우 DML 문을 사용하여 영구 테이블을 수정할 수 없습니다. 예를 들어 임시 테이블에서 데이터를 가져오는 INSERT ... SELECT 문을 사용할 수 없습니다. 임시 테이블은 보조 클러스터에 있으며 명령문이 기본 클러스터에서 실행될 때는 임시 테이블을 사용할 수 없습니다.

XA 트랜잭션

세션 내에서 쓰기 전달이 켜져 있으면 보조 클러스터에서 다음 문을 사용할 수 없습니다. 쓰기 전달이 활성화되지 않은 보조 클러스터나 aurora_replica_read_consistency 설정이 비어 있는 세션 내에서 이러한 명령문을 사용할 수 있습니다. 세션 내에서 쓰기 전달을 켜기 전에 코드에서 이러한 문을 사용하는지 확인하십시오.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

영구 테이블에 대한 LOAD 문

쓰기 전달이 활성화된 보조 클러스터에서는 다음 명령문을 사용할 수 없습니다.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

보조 클러스터의 임시 테이블로 데이터를 로드할 수 있습니다. 하지만 영구 테이블을 참조하는 모든 LOAD 문을 기본 클러스터에서만 실행해야 합니다.

플러그인 문

쓰기 전달이 활성화된 보조 클러스터에서는 다음 명령문을 사용할 수 없습니다.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT 문

세션 내에서 쓰기 전달이 켜져 있으면 보조 클러스터에서 다음 문을 사용할 수 없습니다. 쓰기 전달이 설정되어 있지 않은 보조 클러스터나 aurora_replica_read_consistency 설정이 비어 있는 세션 내에서 이러한 문을 사용할 수 있습니다. 세션 내에서 쓰기 전달을 켜기 전에 코드에서 이러한 문을 사용하는지 확인하십시오.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Aurora MySQL에서 쓰기 전달의 격리 및 일관성

쓰기 전달을 사용하는 세션에서는 REPEATABLE READ 격리 수준만 사용할 수 있습니다. 보조 READ COMMITTED 리전의 읽기 전용 클러스터에서도 AWS 격리 수준을 사용할 수 있지만, 쓰기 전달에는 해당 격리 수준이 작동하지 않습니다. REPEATABLE READ 및 READ COMMITTED 격리 수준에 대한 자세한 내용은 [Aurora MySQL 격리 수준](#) 단원을 참조하십시오.

보조 클러스터에서 읽기 일관성 정도를 제어할 수 있습니다. 읽기 일관성 수준은 기본 클러스터에서 일부 또는 모든 변경 사항이 복제되도록 각 읽기 작업 전에 보조 클러스터가 대기하는 정도를 결정합니다. 읽기 일관성 수준을 조정하여 세션에서 전달된 모든 쓰기 작업이 후속 쿼리 전에 보조 클러스터에 표시되도록 할 수 있습니다. 또한 이 설정을 사용하여 보조 클러스터의 쿼리에 항상 기본 클러스터의 최신 업데이트가 표시되게 할 수 있습니다. 이는 다른 세션이나 다른 클러스터에서 제출된 경우에도 마찬가지입니다. 애플리케이션에 대해 이러한 유형의 동작을 지정하려면 세션 수준 파라미터 aurora_replica_read_consistency의 값을 선택합니다.

> **Important**
>
> 쓰기를 전달할 세션에 대해 항상 aurora_replica_read_consistency 파라미터를 설정합니다. 그렇지 않으면 Aurora에서 해당 세션에 대한 쓰기 전달이 활성화되지 않습니다. 이 파라미터는 기본적으로 비어 있으므로, 이 파라미터를 사용할 때 특정 값을 선택하세요. aurora_replica_read_consistency 파라미터는 쓰기 전달이 활성화된 보조 클러스터에만 영향을 미칩니다.
> Aurora MySQL 버전 2 및 3.04보다 낮은 버전 3의 경우 세션 변수로 aurora_replica_read_consistency를 사용합니다. Aurora MySQL 버전 3.04 이상에서는 세션 변수 또는 DB 클러스터 파라미터로 aurora_replica_read_consistency를 사용할 수 있습니다.

aurora_replica_read_consistency 매개변수에 대해 EVENTUAL, SESSION 및 GLOBAL 값을 지정할 수 있습니다.

일관성 수준을 높이면 애플리케이션은 변경 사항이 AWS 리전 간에 전파될 때까지 더 오랜 시간 동안 대기합니다. 빠른 응답 시간과 쿼리가 실행되기 전에 다른 위치에서 변경한 내용을 완전히 사용할 수 있도록 보장하는 것 사이의 균형을 선택할 수 있습니다.

읽기 일관성을 EVENTUAL로 설정하면 쓰기 전달을 사용하는 보조 AWS 리전의 쿼리가 복제 지연으로 인해 약간 오래된 데이터를 표시할 수 있습니다. 동일한 세션에서 쓰기 작업의 결과는 쓰기 작업이 기본 리전에서 수행되어 현재 리전으로 복제될 때까지 표시되지 않습니다. 쿼리는 업데이트된 결과를 사

용할 수 있을 때까지 대기하지 않습니다. 따라서 쿼리는 명령문의 타이밍 및 복제 지연 양에 따라 이전 데이터 또는 업데이트된 데이터를 검색할 수 있습니다.

읽기 일관성을 SESSION으로 설정하면 쓰기 전달을 사용하는 보조 AWS 리전의 모든 쿼리는 해당 세션에서 수행한 모든 변경의 결과를 표시합니다. 변경 내용은 트랜잭션이 커밋되었는지 여부와 상관없이 표시됩니다. 필요한 경우 쿼리는 전달된 쓰기 작업의 결과가 현재 리전으로 복제될 때까지 대기합니다. 쿼리는 다른 리전이나 현재 리전 내의 다른 세션에서 수행한 쓰기 작업에서 업데이트된 결과가 나올 때까지 대기하지 않습니다.

읽기 일관성을 GLOBAL로 설정하면 보조 AWS 리전의 세션에는 해당 세션에서 변경한 내용이 표시됩니다. 또한 기본 AWS 리전 및 기타 보조 AWS 리전에서 커밋된 모든 변경 내용도 표시됩니다. 각 쿼리는 세션 지연 양에 따라 다른 기간 동안 대기할 수 있습니다. 쿼리가 시작된 시간을 기준으로, 보조 클러스터가 기본 클러스터에서 커밋된 모든 데이터로 최신 상태가 되면 쿼리가 진행됩니다.

쓰기 전달과 관련된 모든 파라미터에 대한 자세한 내용은 [Aurora MySQL에서 쓰기 전달에 대한 구성 파라미터](#) 단원을 참조하십시오.

쓰기 전달 사용의 예제

이 예시에서는 aurora_replica_read_consistency를 세션 변수로 사용합니다. Aurora MySQL 버전 3.04 이상에서는 세션 변수 또는 DB 클러스터 파라미터로 aurora_replica_read_consistency를 사용할 수 있습니다.

다음 예제에서, 기본 클러스터는 US East (N. Virginia) 리전에 있습니다. 보조 클러스터는 미국 동부 (오하이오) 리전에 있습니다. 이 예에서는 INSERT 문 다음에 SELECT 문을 실행하는 결과를 보여 줍니다. aurora_replica_read_consistency 설정의 값에 따라, 결과는 명령문의 타이밍에 따라 다를 수 있습니다. 일관성을 높이기 위해 SELECT 문을 실행하기 전에 잠시 기다립니다. 또는 Aurora은(는) SELECT을(를) 계속하기 전에 결과 복제가 완료될 때까지 자동으로 대기할 수 있습니다.

이 예제에는 eventual의 읽기 일관성 설정이 있습니다. INSERT 문 바로 다음에 SELECT 문을 실행하면 여전히 COUNT(*)의 값이 반환됩니다. 이 값은 새로운 행이 삽입되기 전의 행 수를 반영합니다. 잠시 후에 SELECT를 다시 실행하면 업데이트된 행 수가 반환됩니다. SELECT 명령문은 대기하지 않습니다.

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
```

```
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

읽기 일관성 설정이 session인 경우 INSERT 직후 SELECT 명령문은 INSERT 명령문의 변경 사항이 표시될 때까지 대기합니다. 후속 SELECT 명령문은 대기하지 않습니다.

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

읽기 일관성 설정이 여전히 session인 경우, INSERT 문을 수행한 후 짧은 대기 시간을 도입하면 다음 번 SELECT 문이 실행되는 시간까지 업데이트된 행 수를 사용할 수 있습니다.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

읽기 일관성 설정이 global인 경우, 각 SELECT 문은 쿼리를 수행하기 전에 명령문의 시작 시간을 기준으로 모든 데이터 변경 사항이 표시되도록 보장하기 위해 대기합니다. 각 SELECT 문에 대한 대기 시간은 기본 클러스터와 보조 클러스터 간의 복제 지연 양에 따라 다릅니다.

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
```

```
+-----------+
1 row in set (0.66 sec)
```

Aurora MySQL에서 쓰기 전달을 사용하여 멀티파트 문 실행

DML 문은 `INSERT ... SELECT` 문 또는 `DELETE ... WHERE` 문과 같이 여러 부분으로 구성될 수 있습니다. 이 경우 전체 명령문은 기본 클러스터로 전달되어 기본 클러스터에서 실행됩니다.

Aurora MySQL에서 쓰기 전달을 사용한 트랜잭션

트랜잭션이 기본 클러스터로 전달되는지 여부는 트랜잭션의 액세스 모드에 따라 다릅니다. `SET TRANSACTION` 문 또는 `START TRANSACTION` 문을 사용하여 트랜잭션에 대한 액세스 모드를 지정할 수 있습니다. transaction_read_only 세션 변수의 값을 변경하여 트랜잭션 액세스 모드를 지정할 수도 있습니다. 쓰기 전달이 활성화된 DB 클러스터에 연결되어 있는 동안에만 이 세션 값을 변경할 수 있습니다.

장기 실행 트랜잭션이 상당한 기간 동안 명령문을 실행하지 않으면 유휴 제한 시간을 초과할 수 있습니다. 이 기간의 기본값은 1분입니다. 이 값을 최대 하루까지 늘릴 수 있습니다. 유휴 시간 제한을 초과하는 트랜잭션은 기본 클러스터에서 취소됩니다. 다음 번 후속 명령문을 제출하면 시간 제한 오류가 수신됩니다. 그런 다음 Aurora는 트랜잭션을 롤백합니다.

이러한 유형의 오류는 쓰기 전달을 사용할 수 없는 다른 경우에 발생할 수 있습니다. 예를 들어 기본 클러스터를 다시 시작하거나 쓰기 전달 구성 설정을 끄면 Aurora에서 쓰기 전달을 사용하는 트랜잭션이 취소됩니다.

Aurora MySQL에서 쓰기 전달에 대한 구성 파라미터

Aurora 클러스터 파라미터 그룹에는 쓰기 전달 기능에 대한 설정이 포함되어 있습니다. 이러한 파라미터는 클러스터 파라미터이므로 각 클러스터의 모든 DB 인스턴스에 이러한 변수의 동일한 값이 있어야 합니다. 이러한 파라미터에 대한 자세한 내용은 다음 표에 요약되어 있으며, 표 뒤에 사용 참고 사항이 나와 있습니다.

이름	범위	유형	기본값	유효값
`aurora_fwd_master_idle_timeout` (Aurora MySQL 버전 2)	전 세계	부호 없는 정수	60	1–86,400
`aurora_fwd_master_max_connections_pct` (Aurora MySQL 버전 2)	전 세계	부호 없는 긴 정수	10	0–90

이름	범위	유형	기본값	유효값
aurora_fwd_writer_idle_time out (Aurora MySQL 버전 3)	전 세계	부호 없는 정수	60	1–86,400
aurora_fwd_writer_max_conne ctions_pct (Aurora MySQL 버전 3)	전 세계	부호 없는 긴 정수	10	0–90
aurora_replica_read_consist ency	버전 2 및 3.04보다 낮은 버전 3의 경우 세션, 버전 3.04 이상인 경우 글로벌	Enum	"(Null)	EVENTUAL, SESSION, GLOBAL

보조 클러스터에서 들어오는 쓰기 요청을 제어하려면 기본 클러스터에서 다음과 같은 설정을 사용합니다.

- aurora_fwd_master_idle_timeout, aurora_fwd_writer_idle_timeout: 기본 클러스터가 보조 클러스터에서 전달된 연결에 대한 활동을 닫기 전에 대기하는 시간(초)입니다. 세션이 이 기간 동안 유휴 상태로 유지되면 Aurora에서 세션이 취소됩니다.
- aurora_fwd_master_max_connections_pct, aurora_fwd_writer_max_connections_pct: 리더에서 전달된 쿼리를 처리하기 위해 라이터 DB 인스턴스에서 사용할 수 있는 데이터베이스 연결의 상한입니다. 이 값은 기본 클러스터의 라이터 DB 인스턴스에 대한 max_connections 설정의 백분율로 표시됩니다. 예를 들어 max_connections가 800이고 aurora_fwd_master_max_connections_pct 또는aurora_fwd_writer_max_connections_pct가 10이면 라이터는 최대 80개의 동시 전달 세션을 허용합니다. 이러한 연결은 max_connections 설정을 통해 관리되는 동일한 연결 풀에서 발생합니다.

  하나 이상의 보조 클러스터에 쓰기 전달이 활성화된 경우 이 설정은 기본 클러스터에만 적용됩니다. 값을 줄이면 기존 연결은 영향을 받지 않습니다. Aurora는 보조 클러스터에서 새 연결을 생성하려고 할 때 설정의 새 값을 고려합니다. 기본값은 max_connections 값의 10% 를 나타내는 10입니다. 보조 클러스터에서 쿼리 전달을 활성화하는 경우 보조 클러스터의 쓰기 작업이 성공하려면 이 설정

에 0이 아닌 값이 있어야 합니다. 값이 0이면 쓰기 작업은 ER_CON_COUNT_ERROR 메시지와 함께 오류 코드 Not enough connections on writer to handle your request를 수신합니다.

aurora_replica_read_consistency 파라미터는 쓰기 전달을 활성화합니다. 이는 각 세션에서 사용합니다. 읽기 일관성 수준에 대해 EVENTUAL, SESSION 또는 GLOBAL을(를) 지정할 수 있습니다. 일관성 수준에 대한 자세한 내용은 Aurora MySQL에서 쓰기 전달의 격리 및 일관성 단원을 참조하세요. 이 파라미터에는 다음과 같은 규칙이 적용됩니다.

- 기본값은 "(비어 있음)입니다.
- aurora_replica_read_consistency가 EVENTUAL 또는 SESSION 또는 GLOBAL로 설정된 경우에만 세션에서 쓰기 전달을 사용할 수 있습니다. 이 파라미터는 쓰기 전달이 활성화되어 있고 Aurora 글로벌 데이터베이스에 있는 보조 클러스터의 리더 인스턴스에만 적용됩니다.
- 다중 선언문 트랜잭션 내에서 이 변수를 설정하거나 (비어있는 경우) 설정 해제 (이미 설정된 경우) 할 수 없습니다. 그러나 이러한 트랜잭션 중에 하나의 유효한 값 (EVENTUAL, SESSION, 또는 GLOBAL) 에서 다른 유효한 값 (EVENTUAL, SESSION, 또는 GLOBAL) 으로 변경할 수 있습니다.
- 보조 클러스터에서 쓰기 전달이 활성화되지 않은 경우 이 변수는 SET일 수 없습니다.

Aurora MySQL에서 쓰기 전달에 대한 Amazon CloudWatch 지표

다음 Amazon CloudWatch 지표는 하나 이상의 보조 클러스터에서 쓰기 전달을 사용할 때 기본 클러스터에 적용됩니다. 이러한 지표는 모두 기본 클러스터의 라이터 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위	설명
AuroraDMLRejectedMasterFull	개수	라이터 DB 인스턴스에서 세션이 꽉 차서 거부되는 전달된 쿼리 수입니다. Aurora MySQL 버전 2용
AuroraDMLRejectedWriterFull	개수	라이터 DB 인스턴스에서 세션이 꽉 차서 거부되는 전달된 쿼리 수입니다. Aurora MySQL 버전 3용.

CloudWatch 지표	단위	설명
`ForwardingMasterDMLLatency`	밀리초	라이터 DB 인스턴스에서 전달된 각 DML 문을 처리하는 평균 시간입니다. 보조 클러스터가 쓰기 요청을 전달하는 시간이나 변경 사항을 보조 클러스터에 다시 복제하는 시간은 포함되지 않습니다. Aurora MySQL 버전 2용
`ForwardingMasterDMLThroughput`	초당 개수	이 라이터 DB 인스턴스에서 초당 처리되는 전달된 DML 문 수입니다. Aurora MySQL 버전 2용
`ForwardingMasterOpenSessions`	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. Aurora MySQL 버전 2용
`ForwardingWriterDMLLatency`	밀리초	라이터 DB 인스턴스에서 전달된 각 DML 문을 처리하는 평균 시간입니다. 보조 클러스터가 쓰기 요청을 전달하는 시간이나 변경 사항을 보조 클러스터에 다시 복제하는 시간은 포함되지 않습니다. Aurora MySQL 버전 3용.

CloudWatch 지표	단위	설명
`ForwardingWriterDMLThroughput`	초당 개수	이 라이터 DB 인스턴스에서 초당 처리되는 전달된 DML 문 수입니다. Aurora MySQL 버전 3용.
`ForwardingWriterOpenSessions`	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. Aurora MySQL 버전 3용.

다음 CloudWatch 지표는 각 보조 클러스터에 적용됩니다. 이러한 지표는 쓰기 전달이 활성화된 보조 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위	설명
`ForwardingReplicaDMLLatency`	밀리초	복제본에 있는 전달된 DML의 평균 응답 시간입니다.
`ForwardingReplicaDMLThroughput`	초당 개수	초당 처리되는 전달된 DML 문 수입니다.
`ForwardingReplicaOpenSessions`	개수	리더 DB 인스턴스에서 쓰기 전달을 사용하는 세션 수입니다.
`ForwardingReplicaReadWaitLatency`	밀리초	리더 DB 인스턴스의 `SELECT` 명령문이 기본 클러스터를 따라잡기 위해 대기하는 평균 대기 시간입니다. 리더 DB 인스턴스가 쿼리를 처리하기 전에 대기하는 정도는 `aurora_replica_read_consistency` 설정에 따라 다릅니다.

CloudWatch 지표	단위	설명
ForwardingReplicaReadWaitThroughput	초당 개수	쓰기를 전달하는 모든 세션에서 초당 처리되는 SELECT 문의 총 수입니다.
ForwardingReplicaSelectLatency	밀리초	전달된 SELECT 지연 시간, 모니터링 기간 내에 전달된 모든 SELECT 명령문의 평균입니다.
ForwardingReplicaSelectThroughput	초당 개수	모니터링 기간 내 전달된 SELECT 초당 평균 처리량입니다.

쓰기 전달을 위한 Aurora MySQL 상태 변수

다음 Aurora MySQL 상태 변수는 하나 이상의 보조 클러스터에서 쓰기 전달을 사용할 때 기본 클러스터에 적용됩니다. 이러한 지표는 모두 기본 클러스터의 라이터 DB 인스턴스에서 측정됩니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_master_dml_stmt_count	개수	이 라이터 DB 인스턴스로 전달된 총 DML 문 수입니다. Aurora MySQL 버전 2용
Aurora_fwd_master_dml_stmt_duration	마이크로초	이 라이터 DB 인스턴스로 전달된 DML 문의 총 기간입니다. Aurora MySQL 버전 2용
Aurora_fwd_master_open_sessions	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. Aurora MySQL 버전 2용
Aurora_fwd_master_select_stmt_count	개수	이 라이터 DB 인스턴스로 전달된 총 SELECT 문 수입니다.

Aurora MySQL 상태 변수	단위	설명
		Aurora MySQL 버전 2용
Aurora_fwd_master_select_stmt_duration	마이크로초	이 라이터 DB 인스턴스로 전달된 SELECT 문의 총 기간입니다. Aurora MySQL 버전 2용
Aurora_fwd_writer_dml_stmt_count	개수	이 라이터 DB 인스턴스로 전달된 총 DML 문 수입니다. Aurora MySQL 버전 3용.
Aurora_fwd_writer_dml_stmt_duration	마이크로초	이 라이터 DB 인스턴스로 전달된 DML 문의 총 기간입니다.
Aurora_fwd_writer_open_sessions	개수	라이터 DB 인스턴스에 있는 전달된 세션 수입니다. Aurora MySQL 버전 3용.
Aurora_fwd_writer_select_stmt_count	개수	이 라이터 DB 인스턴스로 전달된 총 SELECT 문 수입니다. Aurora MySQL 버전 3용.
Aurora_fwd_writer_select_stmt_duration	마이크로초	이 라이터 DB 인스턴스로 전달된 SELECT 문의 총 기간입니다. Aurora MySQL 버전 3용.

다음 Aurora MySQL 상태 변수는 각 보조 클러스터에 적용됩니다. 이러한 지표는 쓰기 전달이 활성화된 보조 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_replica_dml_stmt_count	개수	이 리더 DB 인스턴스에서 전달된 총 DML 문 수입니다.

Aurora MySQL 상태 변수	단위	설명
Aurora_fwd_replica _dml_stmt_duration	마이크로초	이 리더 DB 인스턴스에서 전달된 모든 DML 문의 총 기간입니다.
Aurora_fwd_replica _errors_session_li mit	개수	다음 오류 조건 중 하나로 인해 기본 클러스터에서 거부한 세션 수입니다. • 라이터 가득 참 • 진행 중인 전달된 명령문이 너무 많음
Aurora_fwd_replica _open_sessions	개수	리더 DB 인스턴스에서 쓰기 전달을 사용하는 세션 수입니다.
Aurora_fwd_replica _read_wait_count	개수	이 리더 DB 인스턴스에서 총 쓰기 후 읽기 대기 횟수입니다.
Aurora_fwd_replica _read_wait_duration	마이크로초	이 리더 DB 인스턴스의 읽기 일관성 설정으로 인한 총 대기 기간입니다.
Aurora_fwd_replica _select_stmt_count	개수	이 리더 DB 인스턴스에서 전달된 총 SELECT 문 수입니다.
Aurora_fwd_replica _select_stmt_durat ion	마이크로초	이 리더 DB 인스턴스에서 전달된 SELECT 문의 총 기간입니다.

Aurora PostgreSQL 글로벌 데이터베이스에서 쓰기 전달 사용

주제

- Aurora PostgreSQL의 쓰기 전달을 사용할 수 있는 리전 및 버전
- Aurora PostgreSQL에서 쓰기 전달 활성화
- Aurora PostgreSQL에서 보조 클러스터에 쓰기 전달이 활성화되어 있는지 확인

- Aurora PostgreSQL에서 쓰기 전달과 애플리케이션 및 SQL 호환성
- Aurora PostgreSQL에서 쓰기 전달의 격리 및 일관성
- 쓰기 전달을 사용한 트랜잭션 액세스 모드
- Aurora PostgreSQL에서 쓰기 전달을 사용하여 멀티파트 문 실행
- Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터
- Aurora PostgreSQL에서 쓰기 전달에 대한 Amazon CloudWatch 지표
- Aurora PostgreSQL에서 쓰기 전달을 위한 대기 이벤트

Aurora PostgreSQL의 쓰기 전달을 사용할 수 있는 리전 및 버전

Aurora PostgreSQL 버전 16 이상 메이저 버전에서는 글로벌 쓰기 전달이 모든 마이너 버전에서 지원됩니다. 이전 Aurora PostgreSQL 버전의 경우 글로벌 쓰기 전달은 버전 15.4 이상 마이너 버전과 버전 14.9 이상 마이너 버전에서 지원됩니다. 쓰기 전달은 Aurora PostgreSQL 기반 글로벌 데이터베이스를 사용할 수 있는 모든 AWS 리전에서 사용할 수 있습니다.

Aurora PostgreSQL 글로벌 데이터베이스의 버전 및 리전 가용성에 대한 자세한 정보는 Aurora PostgreSQL을 사용하는 Aurora 전역 데이터베이스 섹션을 참조하세요.

Aurora PostgreSQL에서 쓰기 전달 활성화

기본적으로 Aurora Global Database에 보조 클러스터를 추가할 때는 쓰기 전달이 활성화되지 않습니다. 보조 DB 클러스터를 생성하는 동안 또는 생성 후 언제든지 보조 DB 클러스터에 대한 쓰기 전달을 활성화할 수 있습니다. 필요한 경우 나중에 비활성화할 수 있습니다. 쓰기 전달을 활성화하거나 비활성화해도 가동 중지나 재부팅이 발생하지 않습니다.

> **Note**
> 가끔 쓰기가 발생하고 한 트랜잭션에서 최신 쓰기를 읽는 기능인 쓰기 후 읽기 일관성이 필요한 애플리케이션에 로컬 쓰기 전달을 사용할 수 있습니다.

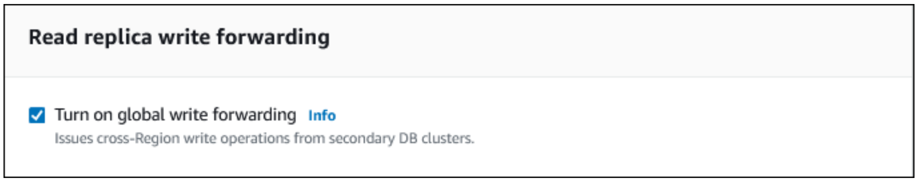
콘솔

콘솔에서 보조 DB 클러스터를 생성하거나 수정할 때 쓰기 전달을 활성화하거나 비활성화할 수 있습니다.

보조 DB 클러스터를 생성할 때 쓰기 전달 활성화 또는 비활성화

보조 DB 클러스터를 새로 생성할 때 읽기 전용 복제본 쓰기 전달에서 전역 쓰기 전달 켜기켜기 확인란을 선택하여 쓰기 전달을 활성화합니다. 또는 확인란의 선택을 취소하여 비활성화할 수도 있습니다. 보조 DB 클러스터를 만들려면 Amazon Aurora DB 클러스터 생성 섹션의 DB 엔진에 대한 지침을 따르세요.

다음 스크린샷은 전역 쓰기 전달 켜기 확인란이 선택된 읽기 전용 복제본 쓰기 전달 섹션을 보여줍니다.



보조 DB 클러스터를 수정할 때 쓰기 전달 활성화 또는 비활성화

콘솔에서 쓰기 전달을 활성화하거나 비활성화하도록 보조 DB 클러스터를 수정할 수 있습니다.

콘솔에서 쓰기 전달을 활성화하거나 비활성화하도록 보조 DB 클러스터를 수정하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 보조 DB 클러스터를 선택하고 수정을 선택합니다.
4. 읽기 전용 복제본 쓰기 전달 섹션에서 전역 쓰기 전달 켜기 확인란을 선택하거나 선택을 취소합니다.
5. 계속을 선택합니다.
6. 수정 예약에서 즉시 적용을 선택합니다. 예약된 다음 유지 관리 기간에 적용을 선택하면 Aurora가 이 설정을 무시하고 쓰기 전달을 즉시 활성화합니다.
7. 클러스터 수정을 선택합니다.

AWS CLI

AWS CLI를 사용하여 쓰기 전달을 활성화하려면 `--enable-global-write-forwarding` 옵션을 사용합니다. 이 옵션은 create-db-cluster 명령을 사용하여 새 보조 클러스터를 생성할 때 작동합니다. 이 옵션은 modify-db-cluster 명령을 사용하여 기존 보조 클러스터를 수정할 때도 작동합니다. 이렇게 하려면 글로벌 데이터베이스에서 쓰기 전달을 지원하는 Aurora 버전을 사용해야 합니다. 이러한 동일

한 CLI 명령과 함께 `--no-enable-global-write-forwarding` 옵션을 사용하여 쓰기 전달을 비활성화할 수 있습니다.

다음 프로시저는 AWS CLI를 사용하여 글로벌 클러스터의 보조 DB 클러스터에 대한 쓰기 전달을 활성화하거나 비활성화하는 방법을 설명합니다.

기존 보조 DB 클러스터에서 쓰기 전달을 활성화하거나 비활성화하는 방법

- modify-db-cluster AWS CLI 명령을 호출하고 다음 값을 입력합니다.
  - `--db-cluster-identifier` – DB 클러스터의 이름입니다.
  - 활성화할 경우 `--enable-global-write-forwarding`, 비활성화할 경우 `--no-enable-global-write-forwarding`

  다음 예시에서는 DB 클러스터 `sample-secondary-db-cluster`의 쓰기 전달을 활성화합니다.

  대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

  Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

RDS API

Amazon RDS API를 사용하여 쓰기 전달을 활성화하려면 `EnableGlobalWriteForwarding` 파라미터를 `true`로 설정합니다. 이 파라미터는 CreateDBCluster 작업을 사용하여 새 보조 클러스터를 생성할 때 작동합니다. 이 파라미터는 ModifyDBCluster 작업을 사용하여 기존 보조 클러스터를 수정할 때도 작동합니다. 이렇게 하려면 글로벌 데이터베이스에서 쓰기 전달을 지원하는 Aurora 버전을 사용해야 합니다. `EnableGlobalWriteForwarding` 파라미터를 `false`로 설정하여 쓰기 전달을 비활성화할 수 있습니다.

Aurora PostgreSQL에서 보조 클러스터에 쓰기 전달이 활성화되어 있는지 확인

보조 클러스터에서 쓰기 전달을 사용할 수 있는지 여부를 확인하려면 클러스터에 `"GlobalWriteForwardingStatus": "enabled"` 특성이 있는지 여부를 확인할 수 있습니다.

AWS Management Console에서 클러스터에 대한 세부 정보 페이지의 구성 탭에 `Read replica write forwarding`이 표시됩니다. 모든 클러스터에 대한 글로벌 쓰기 전달 설정의 상태를 보려면 다음 AWS CLI 명령을 실행합니다.

보조 클러스터에는 쓰기 전달이 켜져 있는지 또는 꺼져 있는지를 나타내는 `"enabled"` 또는 `"disabled"` 값이 표시됩니다. `null` 값은 해당 클러스터에 쓰기 전달을 사용할 수 없음을 나타냅니다. 클러스터가 글로벌 데이터베이스의 일부가 아니거나, 보조 클러스터 대신 기본 클러스터입니다. 또한 쓰기 전달이 켜지거나 꺼지는 중인 경우 이 값은 `"enabling"` 또는 `"disabling"`일 수 있습니다.

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

글로벌 쓰기 전달이 활성화된 모든 보조 클러스터를 찾으려면 다음 명령을 실행하십시오. 이 명령은 또한 클러스터의 판독기 엔드포인트를 반환합니다. Aurora Global Database에서 보조 클러스터에서 기본 클러스터로의 쓰기 전달을 사용할 때는 보조 클러스터의 읽기 장치 엔드포인트를 사용합니다.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Aurora PostgreSQL에서 쓰기 전달과 애플리케이션 및 SQL 호환성

특정 명령문은 쓰기 전달과 함께 글로벌 데이터베이스에서 사용할 때 허용되지 않거나 부실 결과를 생성할 수 있습니다. 또한 사용자 정의 함수 및 사용자 정의 프로시저는 지원되지 않습니다. 따라서 `EnableGlobalWriteForwarding` 설정은 보조 클러스터에 대해 기본적으로 꺼져 있습니다. 이 설정을 켜기 전에, 애플리케이션 코드가 이러한 제한 사항의 영향을 받지 않는지 확인하십시오.

다음과 같은 종류의 SQL 문을 쓰기 전달과 함께 사용할 수 있습니다.

- `INSERT`, `DELETE` 및 `UPDATE`와 같은 데이터 조작 언어(DML) 문
- `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }` 문
- `PREPARE` 및 `EXECUTE` 문
- 이 목록에 있는 문이 포함된 `EXPLAIN` 문

다음과 같은 종류의 SQL 문은 쓰기 전달에서 지원되지 않습니다.

- 데이터 정의 언어(DDL) 문
- `ANALYZE`
- `CLUSTER`
- `COPY`
- 커서 - 커서는 지원되지 않으므로 쓰기 전달을 사용하기 전에 커서를 닫아야 합니다.
- `GRANT`|`REVOKE`|`REASSIGN OWNED`|`SECURITY LABEL`
- `LOCK`
- `SAVEPOINT` 문
- `SELECT INTO`
- `SET CONSTRAINTS`
- `TRUNCATE`

- VACUUM

Aurora PostgreSQL에서 쓰기 전달의 격리 및 일관성

쓰기 전달을 사용하는 세션에서는 REPEATABLE READ 및 READ COMMITTED 격리 수준을 사용할 수 있습니다. 하지만 SERIALIZABLE 격리 수준은 지원되지 않습니다.

보조 클러스터에서 읽기 일관성 정도를 제어할 수 있습니다. 읽기 일관성 수준은 기본 클러스터에서 일부 또는 모든 변경 사항이 복제되도록 각 읽기 작업 전에 보조 클러스터가 대기하는 정도를 결정합니다. 읽기 일관성 수준을 조정하여 세션에서 전달된 모든 쓰기 작업이 후속 쿼리 전에 보조 클러스터에 표시되도록 할 수 있습니다. 또한 이 설정을 사용하여 보조 클러스터의 쿼리에 항상 기본 클러스터의 최신 업데이트가 표시되게 할 수 있습니다. 이는 다른 세션이나 다른 클러스터에서 제출된 경우에도 마찬가지입니다. 애플리케이션에 대해 이러한 유형의 동작을 지정하려면 apg_write_forward.consistency_mode 파라미터의 값을 적절히 선택합니다. apg_write_forward.consistency_mode 파라미터는 쓰기 전달이 활성화된 보조 클러스터에만 영향을 미칩니다.

> **Note**
>
> apg_write_forward.consistency_mode 파라미터에 대해 SESSION, EVENTUAL, GLOBAL 또는 OFF 값을 지정할 수 있습니다. 기본적으로 이 값은 SESSION로 설정됩니다. 값을 OFF로 설정하면 세션에서 쓰기 전달이 비활성화됩니다.

일관성 수준을 높이면 애플리케이션은 변경 사항이 AWS 리전 간에 전파될 때까지 더 오랜 시간 동안 대기합니다. 빠른 응답 시간과 쿼리가 실행되기 전에 다른 위치에서 변경한 내용을 완전히 사용할 수 있도록 보장하는 것 사이의 균형을 선택할 수 있습니다.

사용 가능한 각 일관성 보장 모드 설정의 효과는 다음과 같습니다.

- SESSION - 쓰기 전달을 사용하는 보조 AWS 리전의 모든 쿼리가 해당 세션에서 수행한 모든 변경의 결과를 표시합니다. 변경 내용은 트랜잭션이 커밋되었는지 여부와 상관없이 표시됩니다. 필요한 경우 쿼리는 전달된 쓰기 작업의 결과가 현재 리전으로 복제될 때까지 대기합니다. 쿼리는 다른 리전이나 현재 리전 내의 다른 세션에서 수행한 쓰기 작업에서 업데이트된 결과가 나올 때까지 대기하지 않습니다.
- EVENTUAL - 쓰기 전달을 사용하는 보조 AWS 리전의 쿼리가 복제 지연으로 인해 약간 오래된 데이터를 표시할 수 있습니다. 동일한 세션에서 쓰기 작업의 결과는 쓰기 작업이 기본 리전에서 수행되어 현재 리전으로 복제될 때까지 표시되지 않습니다. 쿼리는 업데이트된 결과를 사용할 수 있을 때까지

대기하지 않습니다. 따라서 쿼리는 명령문의 타이밍 및 복제 지연 양에 따라 이전 데이터 또는 업데이트된 데이터를 검색할 수 있습니다.

- GLOBAL - 보조 AWS 리전의 세션에 해당 세션에서 변경한 내용이 표시됩니다. 또한 기본 AWS 리전 및 기타 보조 AWS 리전에서 커밋된 모든 변경 내용도 표시됩니다. 각 쿼리는 세션 지연 양에 따라 다른 기간 동안 대기할 수 있습니다. 쿼리가 시작된 시간을 기준으로, 보조 클러스터가 기본 클러스터에서 커밋된 모든 데이터로 최신 상태가 되면 쿼리가 진행됩니다.
- OFF - 세션에서 쓰기 전달이 비활성화되었습니다.

쓰기 전달과 관련된 모든 파라미터에 대한 자세한 내용은 [Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터](#) 단원을 참조하십시오.

쓰기 전달을 사용한 트랜잭션 액세스 모드

트랜잭션 액세스 모드가 읽기 전용으로 설정된 경우 쓰기 전달이 사용되지 않습니다. DB 클러스터에 연결되어 있고 쓰기 전달이 활성화된 세션에 연결되어 있는 경우에만 읽기 쓰기를 사용하도록 액세스 모드를 설정할 수 있습니다.

트랜잭션 액세스 모드에 대한 자세한 내용은 [트랜잭션 설정](#)을 참조하세요.

Aurora PostgreSQL에서 쓰기 전달을 사용하여 멀티파트 문 실행

DML 문은 `INSERT ... SELECT` 문 또는 `DELETE ... WHERE` 문과 같이 여러 부분으로 구성될 수 있습니다. 이 경우 전체 명령문은 기본 클러스터로 전달되어 기본 클러스터에서 실행됩니다.

Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터

Aurora 클러스터 파라미터 그룹에는 쓰기 전달 기능에 대한 설정이 포함되어 있습니다. 이러한 파라미터는 클러스터 파라미터이므로 각 클러스터의 모든 DB 인스턴스에 이러한 변수의 동일한 값이 있어야 합니다. 이러한 파라미터에 대한 자세한 내용은 다음 표에 요약되어 있으며, 표 뒤에 사용 참고 사항이 나와 있습니다.

이름	범위	유형	기본값	유효값
apg_write_forward.connect_timeout	세션	초	30	0–2147483647
apg_write_forward.consistency_mode	세션	enum	세션	SESSION, EVENTUAL,

이름	범위	유형	기본값	유효값
				GLOBAL, OFF
apg_write_forward.idle_in_transaction_session_timeout	세션	밀리초	86400000	0–2147483647
apg_write_forward.idle_session_timeout	세션	밀리초	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	전 세계	int	25	1–100

apg_write_forward.max_forwarding_connections_percent 파라미터는 리더에서 전달된 쿼리를 처리하기 위해 사용할 수 있는 데이터베이스 연결의 상한입니다. 이 값은 기본 클러스터의 라이터 DB 인스턴스에 대한 max_connections 설정의 백분율로 표시됩니다. 예를 들어 max_connections가 800이고 apg_write_forward.max_forwarding_connections_percent가 10이면 라이터는 최대 80개의 동시 전달 세션을 허용합니다. 이러한 연결은 max_connections 설정을 통해 관리되는 동일한 연결 풀에서 발생합니다. 하나 이상의 보조 클러스터에 쓰기 전달이 활성화된 경우 이 설정은 기본 클러스터에만 적용됩니다.

보조 클러스터에서 다음 설정을 사용하세요.

- apg_write_forward.consistency_mode - 보조 클러스터에서 읽기 일관성 정도를 제어하는 세션 수준 파라미터입니다. 유효한 값은 SESSION, EVENTUAL, GLOBAL 또는 OFF입니다. 기본적으로 이 값은 SESSION로 설정됩니다. 값을 OFF로 설정하면 세션에서 쓰기 전달이 비활성화됩니다. 일관성 수준에 대한 자세한 내용은 [Aurora PostgreSQL에서 쓰기 전달의 격리 및 일관성](#) 단원을 참조하세요. 이 파라미터는 쓰기 전달이 활성화되어 있고 Aurora Global Database에 있는 보조 클러스터의 리더 인스턴스에만 적용됩니다.
- apg_write_forward.connect_timeout - 기본 클러스터에 연결할 때 보조 클러스터가 연결을 끊기 전에 대기하는 최대 시간(초)입니다. 값이 0이면 무한정 대기할 수 있다는 뜻입니다.
- apg_write_forward.idle_in_transaction_session_timeout - 기본 클러스터가 보조 클러스터에서 전달되고 열린 트랜잭션이 있는 연결을 닫기 전에 활동을 대기하는 시간(밀리초)입니다. 세션이 이 기간을 넘어 트랜잭션에서 유휴 상태로 유지되면 Aurora에서 세션이 종료됩니다. 0 값은 제한 시간을 비활성화합니다.

- `apg_write_forward.idle_session_timeout` - 기본 클러스터가 보조 클러스터에서 전달된 연결을 닫기 전에 활동을 대기하는 시간(밀리초)입니다. 세션이 이 기간을 넘어 유휴 상태로 유지되면 Aurora에서 세션이 종료됩니다. `0` 값은 제한 시간을 비활성화합니다.

Aurora PostgreSQL에서 쓰기 전달에 대한 Amazon CloudWatch 지표

다음 Amazon CloudWatch 지표는 하나 이상의 보조 클러스터에서 쓰기 전달을 사용할 때 기본 클러스터에 적용됩니다. 이러한 지표는 모두 기본 클러스터의 라이터 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위 및 설명
`AuroraForwardingWriterDMLThroughput`	개수(초당). 이 라이터 DB 인스턴스에서 초당 처리되는 전달된 DML 문 수입니다.
`AuroraForwardingWriterOpenSessions`	개수. 전달된 쿼리를 처리하는 이 라이터 DB 인스턴스의 열린 세션 수입니다.
`AuroraForwardingWriterTotalSessions`	개수. 라이터 DB 인스턴스에 있는 전달된 세션의 총 개수입니다.

다음 CloudWatch 지표는 각 보조 클러스터에 적용됩니다. 이러한 지표는 쓰기 전달이 활성화된 보조 클러스터의 각 리더 DB 인스턴스에서 측정됩니다.

CloudWatch 지표	단위 및 설명
`AuroraForwardingReplicaCommitThroughput`	개수(초당). 이 복제본이 초당 전달하는 세션의 커밋 수입니다.
`AuroraForwardingReplicaDMLLatency`	밀리초. 복제본에 있는 전달된 DML의 평균 응답 시간(밀리초)입니다.
`AuroraForwardingReplicaDMLThroughput`	개수(초당). 이 복제본에서 초당 처리되는 전달된 DML 문 수입니다.
`AuroraForwardingReplicaErrorSessionsLimit`	개수. 최대 연결 또는 최대 쓰기 전달 연결 한도에 도달하여 기본 클러스터에서 거부한 세션 수입니다.

CloudWatch 지표	단위 및 설명
AuroraForwardingReplicaOpen Sessions	개수. 복제본 인스턴스에서 쓰기 전달을 사용하는 세션 수입니다.
AuroraForwardingReplicaRead WaitLatency	밀리초. 복제본이 기본 클러스터의 LSN과 일치할 때까지 기다리는 평균 대기 시간(밀리초)입니다. 리더 DB 인스턴스가 대기하는 정도는 apg_write_forward.consisten cy_mode 설정에 따라 다릅니다. 이 설정에 대한 자세한 내용은 the section called "Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터" 섹션을 참조하세요.

Aurora PostgreSQL에서 쓰기 전달을 위한 대기 이벤트

Amazon Aurora는 Aurora PostgreSQL에서 쓰기 전달을 사용할 때 다음과 같은 대기 이벤트를 생성합니다.

주제

IPC:AuroraWriteForwardConnect

`IPC:AuroraWriteForwardConnect` 이벤트는 보조 DB 클러스터의 백엔드 프로세스가 기본 DB 클러스터의 라이터 노드에 대한 연결이 열리기를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

이 이벤트는 보조 리전의 리더 노드에서 기본 DB 클러스터의 라이터 노드로의 연결 시도 횟수가 늘어날수록 증가합니다.

작업

보조 노드에서 기본 리전 라이터 노드로의 동시 연결 수를 줄이세요.

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint` 이벤트는 전달된 쓰기 작업의 결과가 현재 리전으로 복제될 때까지 보조 DB 클러스터에서 노드에서 대기하는 시간을 나타냅니다. 이 이벤트는 세션 수준 `apg_write_forward.consistency_mode` 파라미터가 다음 중 하나로 설정된 경우에만 생성됩니다.

- `SESSION` - 보조 노드에 대한 쿼리는 해당 세션에서 수행한 모든 변경의 결과를 대기합니다.
- `GLOBAL` - 보조 노드에 대한 쿼리는 해당 세션의 변경 결과와 글로벌 클러스터의 기본 리전 및 기타 보조 리전 모두에서 커밋된 모든 변경 내용을 포함하여 해당 세션의 변경 결과를 기다립니다.

`apg_write_forward.consistency_mode` 파라미터 설정에 대한 자세한 내용은 the section called “Aurora PostgreSQL에서 쓰기 전달에 대한 구성 파라미터” 섹션을 참조하세요.

대기 증가의 가능한 원인

대기 시간이 길어지는 일반적인 원인은 다음과 같습니다.

- Amazon CloudWatch `ReplicaLag` 지표로 측정한 바와 같이 복제 지연이 증가했습니다. 이 지표에 대한 자세한 정보는 Aurora PostgreSQL 복제 모니터링 섹션을 참조하세요.
- 기본 리전의 라이터 노드 또는 보조 노드의 로드가 증가했습니다.

작업

애플리케이션 요구 사항에 따라 일관성 모드를 변경합니다.

IPC:AuroraWriteForwardExecute

`IPC:AuroraWriteForwardExecute` 이벤트는 보조 DB 클러스터의 백엔드 프로세스가 전달된 쿼리가 완료되고 기본 DB 클러스터의 라이터 노드로부터 결과를 얻기를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 기본 리전의 라이터 노드에서 많은 수의 행을 가져옵니다.
- 보조 노드와 기본 리전 라이터 노드 사이의 네트워크 지연 시간이 증가하면 보조 노드가 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 보조 노드의 로드가 증가하면 보조 노드에서 기본 리전 라이터 노드로의 쿼리 요청 전송이 지연될 수 있습니다.
- 기본 리전 라이터 노드의 로드가 증가하면 라이터 노드에서 보조 노드로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 필요한 데이터만 검색하도록 쿼리를 최적화합니다.
- 데이터 조작 언어(DML) 작업을 최적화하여 필요한 데이터만 수정합니다.
- 보조 노드 또는 기본 리전 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

`IPC:AuroraWriteForwardGetGlobalConsistencyPoint` 이벤트는 글로벌 일관성 모드를 사용하는 보조 DB 클러스터의 백엔드 프로세스가 쿼리를 실행하기 전에 라이터 노드로부터 글로벌 일관성 보장 지점을 가져오기를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 보조 노드와 기본 리전 라이터 노드 사이의 네트워크 지연 시간이 증가하면 보조 노드가 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 보조 노드의 로드가 증가하면 보조 노드에서 기본 리전 라이터 노드로의 쿼리 요청 전송이 지연될 수 있습니다.
- 기본 리전 라이터 노드의 로드가 증가하면 라이터 노드에서 보조 노드로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 애플리케이션 요구 사항에 따라 일관성 모드를 변경합니다.
- 보조 노드 또는 기본 리전 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactAbort

`IPC:AuroraWriteForwardXactAbort` 이벤트는 보조 DB 클러스터의 백엔드 프로세스가 원격 정리 쿼리 결과를 기다리고 있을 때 발생합니다. 쓰기 전달 트랜잭션이 중단된 후 프로세스를 적절한 상태로 되돌리기 위해 정리 쿼리가 실행됩니다. Amazon Aurora는 오류가 발견되었거나 사용자가 명시적인 `ABORT` 명령을 실행했거나 실행 중인 쿼리를 취소했을 때 이러한 작업을 수행합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 보조 노드와 기본 리전 라이터 노드 사이의 네트워크 지연 시간이 증가하면 보조 노드가 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 보조 노드의 로드가 증가하면 보조 노드에서 기본 리전 라이터 노드로의 정리 쿼리 요청 전송이 지연될 수 있습니다.
- 기본 리전 라이터 노드의 로드가 증가하면 라이터 노드에서 보조 노드로의 데이터 전송이 지연될 수 있습니다.

작업

대기 이벤트의 원인에 따라 다른 작업을 권장합니다.

- 트랜잭션이 중단된 원인을 조사하세요.
- 보조 노드 또는 기본 리전 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactCommit

`IPC:AuroraWriteForwardXactCommit` 이벤트는 보조 DB 클러스터의 백엔드 프로세스가 전달된 커밋 트랜잭션 명령의 결과를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 보조 노드와 기본 리전 라이터 노드 사이의 네트워크 지연 시간이 증가하면 보조 노드가 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 보조 노드의 로드가 증가하면 보조 노드에서 기본 리전 라이터 노드로의 쿼리 요청 전송이 지연될 수 있습니다.
- 기본 리전 라이터 노드의 로드가 증가하면 라이터 노드에서 보조 노드로의 데이터 전송이 지연될 수 있습니다.

작업

보조 노드 또는 기본 리전 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

IPC:AuroraWriteForwardXactStart

`IPC:AuroraWriteForwardXactStart` 이벤트는 보조 DB 클러스터의 백엔드 프로세스가 전달된 시작 트랜잭션 명령의 결과를 기다리고 있을 때 발생합니다.

대기 증가의 가능한 원인

대기가 늘어나는 일반적인 원인은 다음과 같습니다.

- 보조 노드와 기본 리전 라이터 노드 사이의 네트워크 지연 시간이 증가하면 보조 노드가 라이터 노드로부터 데이터를 수신하는 데 걸리는 시간이 늘어납니다.
- 보조 노드의 로드가 증가하면 보조 노드에서 기본 리전 라이터 노드로의 쿼리 요청 전송이 지연될 수 있습니다.
- 기본 리전 라이터 노드의 로드가 증가하면 라이터 노드에서 보조 노드로의 데이터 전송이 지연될 수 있습니다.

작업

보조 노드 또는 기본 리전 라이터 노드가 CPU 또는 네트워크 대역폭의 제약을 받는 경우 CPU 용량이 더 크거나 네트워크 대역폭이 더 큰 인스턴스 유형으로 변경하는 것이 좋습니다.

Amazon Aurora Global Database에서 전환 또는 장애 조치 사용

Aurora Global Database 기능은 Aurora DB 클러스터를 통해 단일 AWS 리전에서 제공되는 표준 고가용성보다 더 많은 비즈니스 연속성 및 재해 복구(BCDR) 보호 기능을 제공합니다. Aurora Global

Database를 사용하면 드물지만 예기치 않게 발생하는 리전 간 재해나 전체 서비스 수준 중단을 신속하게 복구할 계획을 세울 수 있습니다.

다음 지침과 절차를 참조하여 Aurora Global Database 기능을 사용하여 BCDR 전략을 계획, 테스트 및 구현할 수 있습니다.

주제

비즈니스 연속성 및 재해 복구 계획

비즈니스 연속성 및 재해 복구 전략을 계획하려면 다음 업계 용어와 이러한 용어가 Aurora Global Database 기능과 어떻게 연관되는지 이해하는 것이 좋습니다.

재해 복구는 일반적으로 다음 2가지 비즈니스 목표에 따라 이루어집니다.

- Recovery Time Objective(RTO) – 재해 또는 서비스 중단이 발생한 후 시스템이 정상 작동 상태로 돌아가는 데 걸리는 시간입니다. 즉 RTO는 가동 중지 시간을 측정합니다. Aurora Global Database의 경우, RTO는 분 단위가 될 수 있습니다.
- Recovery Point Objective(RPO) – 재해 또는 서비스 중단이 발생한 후 손실될 수 있는 데이터의 양(시간 단위로 측정)입니다. 이러한 데이터 손실은 일반적으로 비동기식 복제 지연으로 인해 발생합니다. Aurora Global Database의 경우, RPO는 일반적으로 초 단위로 측정됩니다. Aurora PostgreSQL–기반 전역 데이터베이스를 사용하면, `rds.global_db_rpo` 파라미터를 사용하여 RPO에 대한 상한선을 설정하고 추적할 수 있지만, 이렇게 하면 기본 클러스터 라이터 노드에서의 트랜잭션 처리에 영향을 줄 수 있습니다. 자세한 내용은 Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO 관리 섹션을 참조하세요.

Aurora Global Database로 전환 또는 장애 조치를 수행하려면 보조 DB 클러스터를 기본 DB 클러스터로 승격해야 합니다. '리전별 중단'이라는 용어는 다양한 장애 시나리오를 설명하는 데 자주 사용됩니다. 최악의 시나리오로는 수백 제곱마일에 영향을 미치는 치명적인 이벤트가 발생하여 광범위한 중단이 발생하는 경우를 예로 들 수 있습니다. 그러나 대부분의 중단은 훨씬 더 국소적으로 발생하므로 클라우드 서비스 또는 고객 시스템의 극히 일부에만 영향을 미칩니다. 중단의 전체 범위를 고려하여 리전

간 장애 조치가 적절한 솔루션인지 확인하고 상황에 적합한 장애 조치 방법을 선택합니다. 장애 조치 또는 전환 접근 방식을 사용해야 하는지 여부는 다음의 특정 중단 시나리오에 따라 달라집니다.

- 장애 조치 – 이 접근 방식을 사용하면 예상치 못한 중단으로부터 서비스를 복구할 수 있습니다. 이 접근 방식을 사용하면 리전 간 장애 조치를 Aurora Global Database의 보조 DB 클러스터 중 하나에 수행합니다. 이 접근 방식의 RPO는 일반적으로 초 단위로 측정되는 0을 제외한 값입니다. 데이터 손실량은 장애 시점에 AWS 리전 전반에서 발생하는 Aurora Global Database 복제 지연 시간에 따라 달라집니다. 자세한 내용은 [계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구]를 참조하세요.
- 전환 – 이 작업의 이전 명칭은 '계획된 관리형 장애 조치'입니다. 모든 Aurora 클러스터 및 상호 작용하는 기타 서비스가 정상 상태인 운영 유지 관리 및 기타 계획된 운영 절차와 같은 제어된 시나리오에는 이 접근 방식을 사용합니다. 이 기능은 다른 변경 작업을 수행하기 전에 보조 DB 클러스터를 기본 DB 클러스터와 동기화하므로 RPO는 0입니다(데이터 손실 없음). 자세한 내용은 [Amazon Aurora Global Database에서 전환 수행]를 참조하세요.

> **Note**
>
> 헤드리스 보조 Aurora DB 클러스터로 전환하거나 장애 조치를 수행하려면 먼저 DB 인스턴스를 추가해야 합니다. 헤드리스 DB 클러스터에 관해 자세히 알아보려면 [보조 리전에 헤드리스 Aurora DB 클러스터 생성] 단원을 참조하세요.

Amazon Aurora Global Database에서 전환 수행

> **Note**
>
> 전환의 이전 명칭은 계획된 관리형 장애 조치입니다.

전환을 사용하면 기본 클러스터의 리전을 정기적으로 변경할 수 있습니다. 이 접근 방식은 운영 유지 관리 및 기타 계획된 운영 절차 등 제어된 시나리오를 대상으로 개발되었습니다.

전환은 일반적으로 3가지의 경우에 사용됩니다.

- 특정 산업에서 필요로 하는 '리전별 순환' 요구 사항을 살펴봅니다. 예를 들어 금융 서비스 규정에 따라 재해 복구 절차가 정기적으로 실행되도록 보장하려면 Tier-0 시스템을 몇 개월 동안 다른 리전으로 전환해야 할 수 있습니다.

- 여러 리전의 'follow-the-sun' 애플리케이션을 살펴봅니다. 각기 다른 시간대 전반에서 업무 시간을 기준으로 여러 리전별로 지연 시간이 짧은 쓰기 기능을 제공하고자 하는 기업을 예로 들 수 있습니다.
- 데이터 손실 제로의 방법이며, 장애 조치 후 기존의 기본 리전으로 페일백하는 데 유용합니다.

> **Note**
>
> 전환은 모든 Aurora 클러스터와 상호 작용하는 기타 서비스가 정상 상태인 Aurora Global Database에서 사용하도록 설계되었습니다. 예상치 못한 중단이 발생한 상태에서 복구하려면 계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구 단원에 나온 적절한 절차를 따르세요.
> 기본 및 보조 DB 클러스터에 있는 메이저 및 마이너 엔진 버전이 동일한 경우에만 Aurora Global Database에서 관리형 리전 간 전환을 수행할 수 있습니다. 엔진 및 엔진 버전에 따라 패치 수준이 동일해야 하거나 패치 수준이 다를 수 있습니다. 패치 수준이 다른 기본 클러스터와 보조 클러스터 간에 이러한 작업을 허용하는 엔진 및 엔진 버전 목록은 관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성 섹션을 참조하세요. 전환을 시작하기 전에 글로벌 클러스터의 엔진 버전을 확인하여 관리형 리전 간 전환을 지원하는지 확인하고 필요한 경우 업그레이드합니다.

전환 중에 Aurora는 선택한 보조 리전의 클러스터를 기본 클러스터로 만듭니다. 전환 메커니즘은 글로벌 데이터베이스의 기존 복제 토폴로지를 유지합니다. 동일한 리전에 동일한 수의 Aurora 클러스터가 있습니다. Aurora는 대상 보조 리전 클러스터가 프라이머리 리전 클러스터와 완전히 동기화될 때까지 기다린 다음, 전환 프로세스를 시작합니다. 그런 다음, 기본 리전의 DB 클러스터는 읽기 전용이 됩니다. 선택한 보조 클러스터는 읽기 전용 노드 중 하나를 전체 라이터 상태로 승격하고, 이에 따라 보조 클러스터가 기본 클러스터의 역할을 맡을 수 있습니다. 프로세스 시작 시 대상 보조 클러스터가 프라이머리 클러스터와 동기화되었으므로, 새로운 프라이머리 클러스터는 데이터 손실 없이 Aurora 글로벌 데이터베이스에 대한 작업을 계속합니다. 기본 클러스터와 선택한 보조 클러스터가 새 역할을 맡으므로 데이터베이스를 잠시 사용할 수 없습니다.

> **Note**
>
> 전환을 수행한 후 Aurora PostgreSQL의 복제 슬롯을 관리하려면 Aurora PostgreSQL 논리적 슬롯 관리 섹션을 참조하세요.

애플리케이션 가용성을 최적화하려면 이 기능을 사용하기 전에 다음 작업을 수행하는 것이 좋습니다.

- 사용량이 적은 시간이나 기본 DB 클러스터에 대한 쓰기가 최소인 시간에 이 작업을 수행합니다.
- Aurora 전역 데이터베이스의 모든 보조 Aurora DB 클러스터에 대한 지연 시간을 확인합니다. 모든 Aurora PostgreSQL 기반 글로벌 데이터베이스 및 엔진 버전 3.04.0 이상 또는 2.12.0 이상으로 시작하는 Aurora MySQL 기반 글로벌 데이터베이스의 경우 Amazon CloudWatch를 사용하여 모든 보조 DB 클러스터에 대한 `AuroraGlobalDBRPOLag` 지표를 확인하세요. 하위 마이너 버전의 Aurora MySQL 기반 글로벌 데이터베이스의 경우에는 `AuroraGlobalDBReplicationLag` 지표를 확인하세요. 이 지표를 통해 보조 DB 클러스터가 기본 DB 클러스터에 비해 얼마나 뒤처져 있는지(밀리초 단위) 알 수 있습니다. 이 값은 Aurora가 전환을 완료하는 데 걸리는 시간에 직접적으로 비례합니다. 따라서 지연 값이 클수록 전환 시간이 더 오래 걸립니다. 이러한 지표를 검사할 때는 현재 기본 클러스터에서 검사합니다.

  CloudWatch의 Aurora 지표에 대한 자세한 내용은 Amazon Aurora에 대한 클러스터 수준 지표 단원을 참조하세요.
- 전환 중에 승격되는 보조 DB 클러스터는 이전 기본 DB 클러스터와 구성 설정이 다를 수 있습니다. Aurora Global Database 클러스터의 모든 클러스터에서 다음 유형의 구성 설정을 일관되게 유지하는 것이 좋습니다. 이렇게 하면 전환 후 성능 문제, 워크로드 비호환성 및 기타 변칙적인 동작을 최소화하는 데 도움이 됩니다.
  - 필요한 경우 새 기본 클러스터에 대한 Aurora DB 클러스터 파라미터 그룹 구성 - 보조 DB 클러스터가 기본 클러스터 역할을 맡도록 승격되면 보조 클러스터의 파라미터 그룹이 기본 클러스터와 다르게 구성될 수 있습니다. 그런 경우, 승격된 보조 DB 클러스터의 파라미터 그룹을 기본 클러스터의 설정에 맞게 수정합니다. 자세한 방법은 Aurora 글로벌 데이터베이스에 대한 파라미터 수정 단원을 참조하십시오.
  - Amazon CloudWatch Events 및 경보 등의 모니터링 도구 및 옵션 구성 – 승격된 DB 클러스터를 전역 데이터베이스에 필요한 것과 동일한 로깅 기능, 경보 등으로 구성합니다. 파라미터 그룹과 마찬가지로, 이러한 기능에 대한 구성은 전환 프로세스 중에 기본 클러스터에서 상속되지 않습니다. 복제 지연과 같은 일부 CloudWatch 지표는 보조 리전에서만 사용할 수 있습니다. 따라서 전환으로 인해 해당 지표를 보고 경보를 설정하는 방법이 달라짐에 따라 사전 정의된 대시보드를 변경해야 할 수도 있습니다. Aurora DB 클러스터 및 모니터링에 대한 자세한 정보는 Amazon CloudWatch로 Amazon Aurora 지표 모니터링 섹션을 참조하세요.
  - 다른 AWS 서비스와의 통합 구성 - Aurora Global Database가 AWS Secrets Manager, AWS Identity and Access Management, Amazon S3 및 AWS Lambda 등의 AWS 서비스와 통합되는 경우, 필요에 따라 이러한 서비스와의 통합이 구성되어 있는지 확인해야 합니다. Aurora 전역 데이터베이스를 IAM, Amazon S3, Lambda과(와) 통합하는 방법에 대한 자세한 내용은 다른 AWS 서비스와 함께 Amazon Aurora Global Database 사용 섹션을 참조하세요. Secrets Manager에 대한 자세한 내용은 AWS 리전에 걸쳐 AWS Secrets Manager에서 비밀 복제를 자동화하는 방법을 참조하세요.

Aurora Global Database 라이터 엔드포인트를 사용하는 경우 애플리케이션의 연결 설정을 변경할 필요가 없습니다. DNS 변경 사항이 전파되었고, 새 기본 클러스터에 연결하여 쓰기 작업을 수행할 수 있는지 확인합니다. 그런 다음 애플리케이션의 전체 작업을 재개할 수 있습니다.

애플리케이션 연결이 글로벌 라이터 엔드포인트 대신 이전 기본 클러스터의 클러스터 엔드포인트를 사용한다고 가정해 보겠습니다. 이 경우 새 기본 클러스터의 클러스터 엔드포인트를 사용하도록 애플리케이션 연결 설정을 변경해야 합니다. Aurora 전역 데이터베이스를 생성할 때 제공된 이름을 수락한 경우, 애플리케이션에 있는 승격된 클러스터의 엔드포인트 문자열에서 `-ro`을(를) 제거하여 엔드포인트를 변경할 수 있습니다. 예를 들어, 보조 클러스터의 엔드포인트 my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com은(는) 해당 클러스터가 기본 클러스터로 승격될 때 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`이(가) 됩니다.

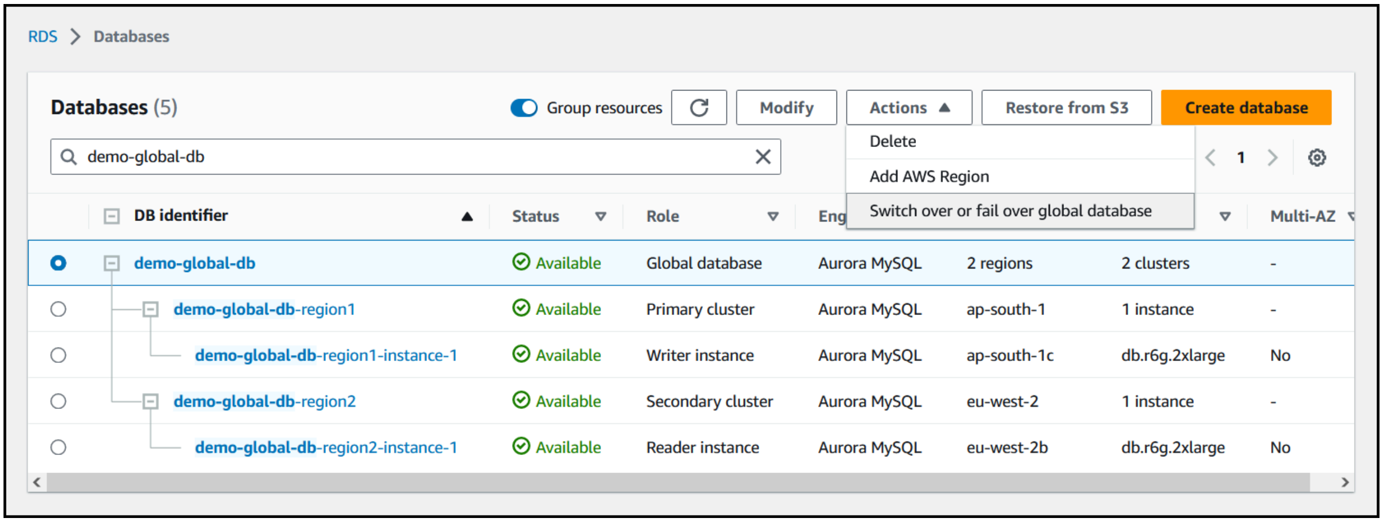
RDS 프록시를 사용하는 경우, 애플리케이션의 쓰기 작업을 새 기본 클러스터와 연결된 프록시의 적절한 읽기/쓰기 엔드포인트로 리디렉션해야 합니다. 이 프록시 엔드포인트는 기본 엔드포인트이거나 사용자 지정 읽기/쓰기 엔드포인트일 수 있습니다. 자세한 내용은 RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식을 참조하세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora Global Database 전환을 수행할 수 있습니다.

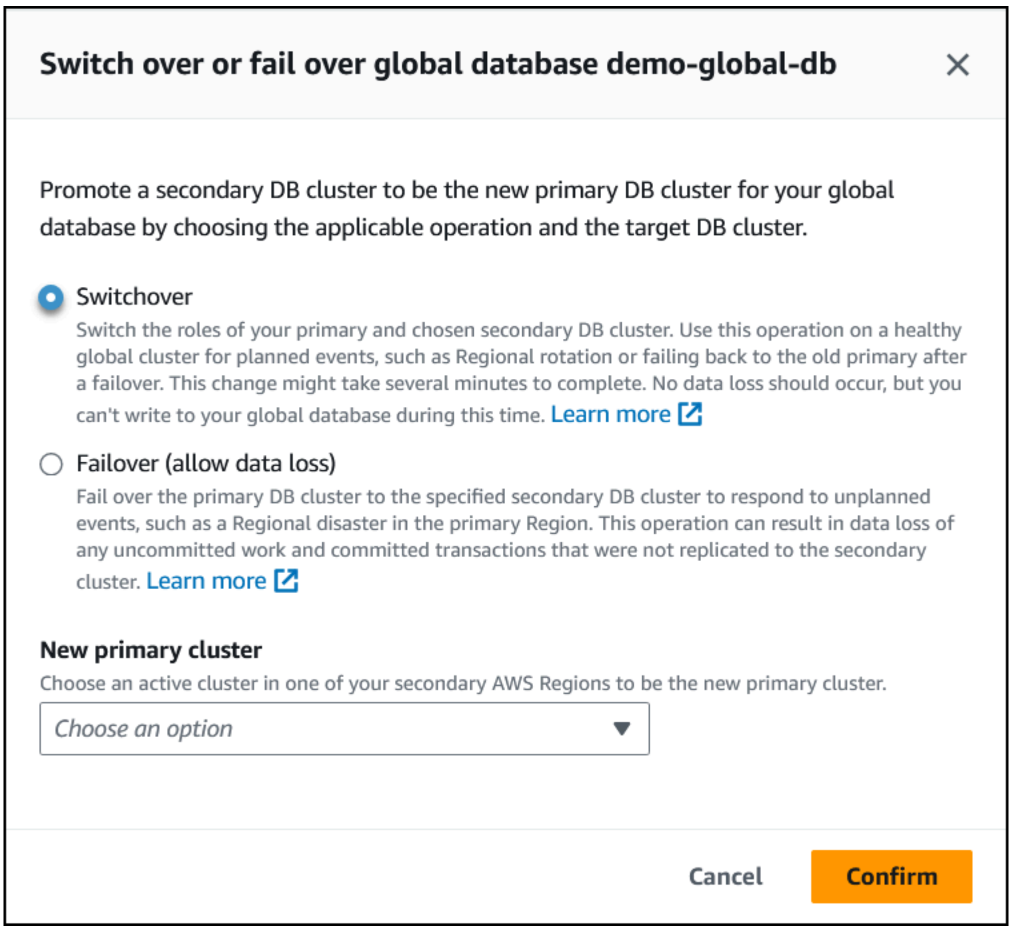
콘솔

Aurora Global Database에서 전환을 수행하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 전환을 수행하려는 Aurora Global Database를 찾습니다.
3. 작업 메뉴에서 글로벌 데이터베이스 전환 또는 장애 조치를 선택합니다.



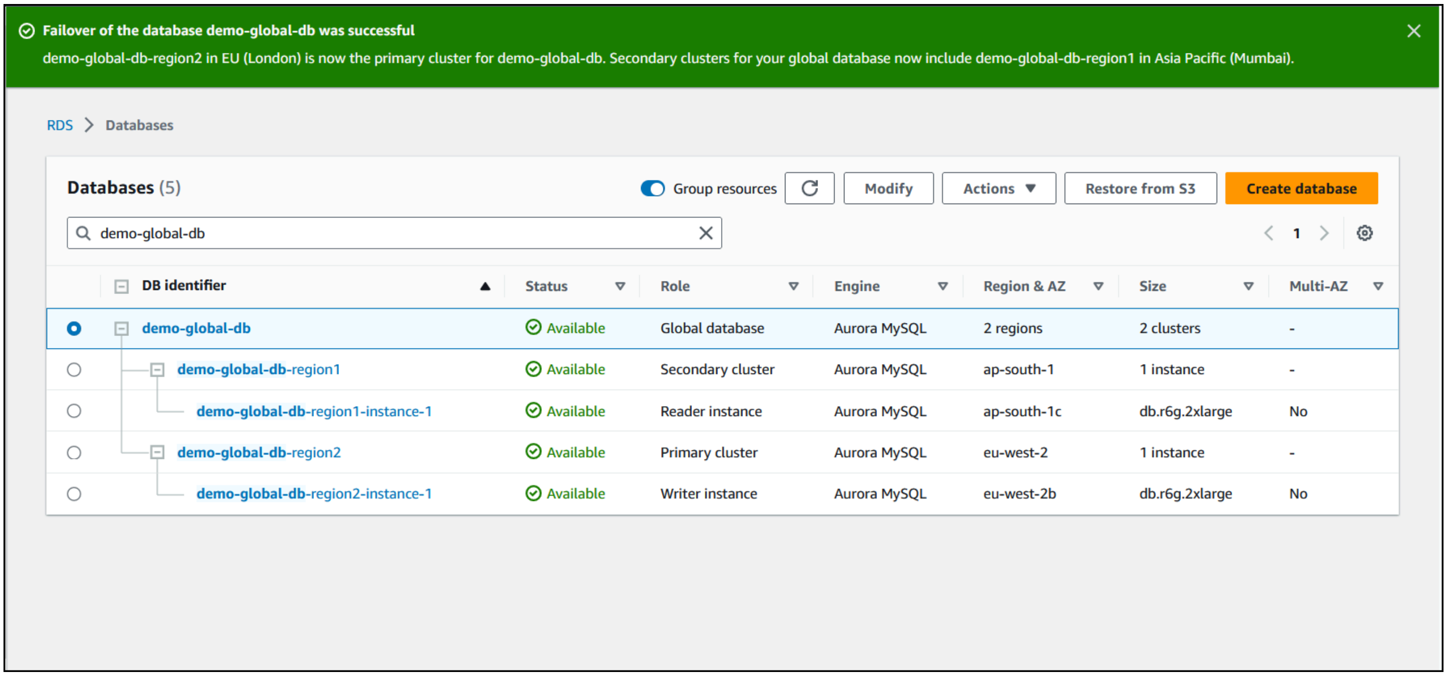
4. 전환을 선택합니다.



5. 새 기본 클러스터에서 보조 AWS 리전 중 하나의 활성 클러스터를 새 기본 클러스터로 선택합니다.

6. 확인을 선택합니다.

전환이 완료되면 다음 이미지와 같이 데이터베이스 목록에서 Aurora DB 클러스터와 해당 클러스터의 현재 상태를 확인할 수 있습니다.



AWS CLI

Aurora Global Database에서 전환을 수행하는 방법

[switchover-global-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/switchover-global-cluster.html) CLI 명령을 사용하여 Aurora Global Database의 전환을 수행합니다. 명령을 사용하여 다음 파라미터에 대한 값을 전달합니다.

- `--region` – Aurora Global Database의 기본 DB 클러스터가 실행 중인 AWS 리전을 지정합니다.
- `--global-cluster-identifier` – Aurora 전역 데이터베이스의 이름을 지정합니다.
- `--target-db-cluster-identifier` – Aurora 전역 데이터베이스의 기본 클러스터로 승격시킬 Aurora DB 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

Windows의 경우:

```
aws rds --region region_of_primary ^
```

```
    switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

RDS API

Aurora Global Database의 전환을 수행하려면 SwitchoverGlobalCluster API 작업을 실행합니다.

계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구

드문 경우지만, Aurora Global Database의 기본 AWS 리전에서 예상치 못한 중단이 발생할 수 있습니다. 이럴 경우, 기본 Aurora DB 클러스터와 해당 라이터 노드를 사용할 수 없으며 기본 클러스터와 보조 클러스터 간의 복제가 중단됩니다. 가동 중지 시간(RTO)과 데이터 손실(RPO)을 둘 다 최소화하기 위해 신속하게 리전 간 장애 조치를 수행할 수 있습니다.

Aurora Global Database에는 재해 복구 상황에서 사용할 수 있는 두 가지 장애 조치 방법이 있습니다.

- 관리형 장애 조치 - 이는 재해 복구를 수행할 때 권장하는 방법입니다. 이 방법을 사용하는 경우 Aurora는 기존의 기본 리전을 다시 사용할 수 있게 될 때 자동으로 글로벌 데이터베이스에 보조 리전으로 이를 다시 추가합니다. 따라서 글로벌 클러스터의 기존 토폴로지가 계속 유지됩니다. 이 방법을 사용하는 방법을 알아보려면 Aurora Global Database에서 계획된 관리형 장애 조치 수행 단원을 참조하세요.
- 수동 장애 조치 – 이 대체 방법은 가령 기본 리전과 보조 리전이 비호환 엔진 버전을 실행 중인 관계로 관리형 장애 조치를 사용할 수 없을 때 사용할 수 있습니다. 이 방법을 사용하는 방법을 알아보려면 Aurora Global Database에서 수동 장애 조치 수행 단원을 참조하세요.

> **Important**
>
> 두 장애 조치 방법을 사용할 경우, 장애 조치 이벤트가 발생하기 전에 선택한 보조 리전에 쓰기 트랜잭션 데이터가 복제되지 않으면 해당 데이터가 손실될 수 있습니다. 하지만 선택한 보조 DB 클러스터의 DB 인스턴스를 기본 라이터 DB 인스턴스로 승격시키는 복구 프로세스는 데이터가 트랜잭션 형태로 일관된 상태를 유지하도록 보장해 줍니다. 장애 조치도 브레인 분할 문제에 취약합니다.

Aurora Global Database에서 계획된 관리형 장애 조치 수행

이 접근 방식은 실제 리전별 재해 이벤트 또는 전체 서비스 수준 중단이 발생하는 경우 비즈니스 연속성을 유지하기 위해 개발되었습니다.

관리형 장애 조치 중에는 선택한 보조 리전의 보조 클러스터가 새 기본 클러스터가 됩니다. 선택한 보조 클러스터에서는 읽기 전용 노드 중 하나가 전체 라이터 상태로 승격됩니다. 이 단계를 통해 클러스터는 기본 클러스터의 역할을 맡게 됩니다. 클러스터가 새 역할을 맡는 동안에는 데이터베이스를 일시적으로 사용할 수 없습니다. 이전 기본 리전이 정상이고 다시 사용 가능한 상태가 되면 Aurora는 자동으로 글로벌 클러스터에 보조 리전으로 다시 추가합니다. 따라서 Aurora Global Database의 기존 복제 토폴로지가 유지됩니다.

> **Note**
>
> 장애 조치를 수행한 후 Aurora PostgreSQL의 복제 슬롯을 관리하려면 [Aurora PostgreSQL 논리적 슬롯 관리](#) 섹션을 참조하세요.

> **Note**
>
> 기본 및 보조 DB 클러스터에 있는 메이저 및 마이너 엔진 버전이 동일한 경우에만 Aurora Global Database에서 관리형 리전 간 장애 조치를 수행할 수 있습니다. 엔진 및 엔진 버전에 따라 패치 수준이 동일해야 하거나 패치 수준이 다를 수 있습니다. 패치 수준이 다른 기본 클러스터와 보조 클러스터 간에 이러한 작업을 허용하는 엔진 및 엔진 버전 목록은 [관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성](#) 섹션을 참조하세요. 장애 조치를 시작하기 전에 글로벌 클러스터의 엔진 버전을 확인하여 관리형 리전 간 전환을 지원하는지 확인하고 필요한 경우 업그레이드합니다. 엔진 버전에 동일한 패치 수준이 필요하지만 서로 다른 패치 수준을 실행 중인 경우 [Aurora Global Database에서 수동 장애 조치 수행](#)에 나온 단계에 따라 수동으로 장애 조치를 수행할 수 있습니다.

관리형 장애 조치는 선택한 보조 리전과 현재 기본 리전 간에 데이터가 동기화될 때까지 기다리지 않습니다. Aurora Global Database는 데이터를 비동기적으로 복제하기 때문에 모든 트랜잭션이 전체 읽기/쓰기 기능을 수락하도록 승격되기 전에 선택한 보조 AWS 리전에 복제되지 않을 수 있습니다.

데이터가 일관된 상태에 있는지 확인하기 위해 Aurora는 복구 후 이전 기본 리전에 대한 새 스토리지 볼륨을 생성합니다. Aurora는 장애가 발생하는 시점에 기존 스토리지 볼륨의 스냅샷을 만들고자 시도한 후, AWS 리전에 새 스토리지 볼륨을 생성합니다. 이렇게 하면 스냅샷을 복원하고 누락된 데이터를 복원할 수 있습니다. 이 작업이 성공하면 Aurora는 AWS Management Console의 스냅샷 섹션에 이름이 `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`*인 스냅샷을 배치합니다. `describe-db-cluster-snapshots` AWS CLI 명령 또는 `DescribeDBClusterSnapshots` API 작업을 사용하여 스냅샷에 대한 세부 정보를 볼 수도 있습니다.

관리형 장애 조치를 시작할 때 Aurora는 또한 가용성이 높은 Aurora 스토리지 계층을 통해 쓰기 트래픽을 중단하려고 시도합니다. 이 메커니즘을 '펜싱 쓰기'라고 합니다. 프로세스가 성공하면 Aurora는 쓰기가 중지되었음을 알리는 RDS 이벤트를 내보냅니다. 한 리전에서 AZ가 여러 번 실패하는 경우 쓰기 펜싱 프로세스가 적시에 성공하지 못할 수 있습니다. 이 경우 Aurora는 쓰기 중지 프로세스가 시간 초과되었음을 알리는 RDS 이벤트를 내보냅니다. 네트워크에서 이전 기본 클러스터에 연결할 수 있는 경우 Aurora는 이러한 이벤트를 거기에 기록합니다. 그렇지 않은 경우 Aurora는 새 기본 클러스터에 이벤트를 기록합니다. 이러한 이벤트에 대한 자세한 내용은 DB 클러스터 이벤트 섹션을 참조하세요. 펜싱 쓰기는 최선의 노력이므로 이전 기본 리전에서 쓰기가 일시적으로 수락되어 브레인 분할 문제가 발생할 수 있습니다.

Aurora Global Database에서 장애 조치를 수행하기 전에 다음 작업을 완료하는 것이 좋습니다. 이렇게 하면 브레인 분할 문제가 발생하거나 이전 기본 클러스터의 스냅샷에서 복제되지 않은 데이터를 복구할 가능성이 최소화됩니다.

- 쓰기가 Aurora Global Database의 기본 클러스터로 전송되는 경우를 방지하려면 애플리케이션을 오프라인으로 전환합니다.
- 기본 DB 클러스터에 연결하는 모든 애플리케이션이 글로벌 라이터 엔드포인트를 사용하고 있는지 확인합니다. 이 엔드포인트의 값은 전환 또는 장애 조치로 인해 새 리전이 기본 클러스터가 되더라도 동일하게 유지됩니다. Aurora는 글로벌 엔드포인트를 통해 제출된 쓰기 작업에 대한 데이터 손실 가능성을 최소화하기 위해 추가 보호 조치를 구현합니다. 글로벌 라이터 엔드포인트에 대한 자세한 내용은 Amazon Aurora Global Database에 연결 섹션을 참조하세요.
- 글로벌 라이터 엔드포인트와 애플리케이션 또는 네트워킹 계층 캐시 DNS 값을 사용하는 경우 DNS 캐시의 TTL(Time-to-live)을 5초와 같은 낮은 값으로 줄입니다. 이렇게 하면 애플리케이션이 DNS 변경 사항을 글로벌 라이터 엔드포인트에 빠르게 등록합니다. Aurora는 이전 기본 리전의 쓰기를 차단하려고 시도하지만 작업이 반드시 성공한다는 보장은 없습니다. DNS 캐시 기간을 줄이면 브레인 분할 문제가 발생할 가능성이 더욱 줄어듭니다. 또는 Aurora가 글로벌 라이터 엔드포인트의 DNS 변경을 관찰한 시기를 알려주는 RDS 이벤트를 확인할 수 있습니다. 이렇게 하면 애플리케이션 쓰기 트래픽을 다시 시작하기 전에 애플리케이션이 DNS 변경도 등록했는지 확인할 수 있습니다.
- Aurora 전역 데이터베이스의 모든 보조 Aurora DB 클러스터에 대한 지연 시간을 확인합니다. 복제 지연이 가장 적게 소요되는 보조 리전을 선택하면 현재 장애가 발생한 기본 리전에서 데이터 손실을 최소화할 수 있습니다.

  모든 버전의 Aurora PostgreSQL 기반 글로벌 데이터베이스 및 엔진 버전 3.04.0 이상 또는 2.12.0 이상으로 시작하는 Aurora MySQL 기반 글로벌 데이터베이스의 경우 Amazon CloudWatch를 사용하여 모든 보조 DB 클러스터에 대한 `AuroraGlobalDBRPOLag` 지표를 확인하세요. 하위 마이너 버전의 Aurora MySQL 기반 글로벌 데이터베이스의 경우에는 `AuroraGlobalDBReplicationLag` 지

표를 확인하세요. 이 지표를 통해 보조 DB 클러스터가 기본 DB 클러스터에 비해 얼마나 뒤처져 있는지(밀리초 단위) 알 수 있습니다.

CloudWatch의 Aurora 지표에 대한 자세한 내용은 Amazon Aurora에 대한 클러스터 수준 지표 단원을 참조하세요.

관리형 장애 조치 중에, 선택한 보조 DB 클러스터가 기본 DB 클러스터처럼 새 역할로 승격됩니다. 하지만 기본 DB 클러스터의 다양한 구성 옵션은 상속되지 않습니다. 구성이 일치하지 않으면 성능 문제, 워크로드 비호환성, 기타 비정상적인 동작이 발생할 수 있습니다. 이러한 문제를 방지하려면, 다음에 대해 Aurora 전역 데이터베이스 클러스터 간의 차이점을 해결하는 것이 좋습니다.

- 필요한 경우 새 기본 클러스터에 대한 DB 클러스터 파라미터 그룹 Aurora구성 – Aurora Global Database의 각 Aurora 클러스터에 대해 Aurora DB 클러스터 파라미터 그룹을 개별적으로 구성할 수 있습니다. 즉, 보조 DB 클러스터가 기본 클러스터 역할을 맡도록 승격되면 보조 클러스터의 파라미터 그룹이 기본 클러스터와 다르게 구성될 수 있습니다. 그런 경우, 승격된 보조 DB 클러스터의 파라미터 그룹을 기본 클러스터의 설정에 맞게 수정합니다. 자세한 방법은 Aurora 글로벌 데이터베이스에 대한 파라미터 수정 단원을 참조하십시오.
- Amazon CloudWatch Events 및 경보 등의 모니터링 도구 및 옵션 구성 – 승격된 DB 클러스터를 전역 데이터베이스에 필요한 것과 동일한 로깅 기능, 경보 등으로 구성합니다. 파라미터 그룹과 마찬가지로, 이러한 기능에 대한 구성은 장애 조치 프로세스 중에 기본 클러스터에서 상속되지 않습니다. 복제 지연과 같은 일부 CloudWatch 지표는 보조 리전에서만 사용할 수 있습니다. 따라서 장애 조치로 인해 해당 지표를 보고 경보를 설정하는 방법이 달라짐에 따라 사전 정의된 대시보드를 변경해야 할 수도 있습니다. Aurora DB 클러스터 모니터링에 대한 자세한 정보는 Amazon CloudWatch로 Amazon Aurora 지표 모니터링 섹션을 참조하세요.
- 다른 AWS 서비스와의 통합 구성 - Aurora Global Database가 AWS Secrets Manager, AWS Identity and Access Management, Amazon S3 및 AWS Lambda와 같은 다른 AWS 서비스와 통합되는 경우, 보조 리전에서 액세스하는 데 필요한 요구 사항에 맞게 구성되어 있는지 확인해야 합니다. Aurora 전역 데이터베이스를 IAM, Amazon S3, Lambda과(와) 통합하는 방법에 대한 자세한 내용은 다른 AWS 서비스와 함께 Amazon Aurora Global Database 사용 섹션을 참조하세요. Secrets Manager에 대한 자세한 내용은 AWS 리전에 걸쳐 AWS Secrets Manager에서 비밀 복제를 자동화하는 방법을 참조하세요.

일반적으로 선택한 보조 클러스터는 몇 분 이내에 기본 역할을 맡게 됩니다. 새 기본 리전의 라이터 DB 인스턴스를 사용할 수 있게 되면 애플리케이션을 노드에 연결하고 워크로드를 재개할 수 있습니다. Aurora에서 새 기본 클러스터가 승격되고 난 후, 모든 추가 보조 리전 클러스터가 자동으로 재구축됩니다.

Aurora Global Database는 비동기식 복제를 사용하므로 보조 리전마다 복제 지연 시간이 다를 수 있습니다. Aurora는 이러한 보조 리전을 재구축하여 새로운 기본 리전 클러스터와 정확히 동일한 시점의 데이터를 확보하도록 지원합니다. 전체 재구축 작업에 소요되는 시간은 스토리지 볼륨의 크기와 리전 간 거리에 따라 몇 분에서 몇 시간이 걸릴 수 있습니다. 보조 리전 클러스터의 재구축이 새 기본 리전에서 완료되면 읽기 권한으로 액세스할 수 있습니다.

새 기본 라이터가 승격되어 사용 가능해지면 새 기본 리전의 클러스터에서 Aurora Global Database의 읽기 및 쓰기 작업을 처리할 수 있습니다.

글로벌 엔드포인트를 사용하는 경우 애플리케이션의 연결 설정을 변경할 필요가 없습니다. DNS 변경 사항이 전파되었고, 새 기본 클러스터에 연결하여 쓰기 작업을 수행할 수 있는지 확인합니다. 그런 다음 애플리케이션의 전체 작업을 재개할 수 있습니다.

글로벌 엔드포인트를 사용하지 않는 경우 새로 승격된 기본 DB 클러스터에 클러스터 엔드포인트를 사용하도록 애플리케이션의 엔드포인트를 변경해야 합니다. Aurora 전역 데이터베이스를 생성할 때 제공된 이름을 수락한 경우, 애플리케이션에 있는 승격된 클러스터의 엔드포인트 문자열에서 `-ro`을(를) 제거하여 엔드포인트를 변경할 수 있습니다.

예를 들어, 보조 클러스터의 엔드포인트 `my-global.cluster-`**`ro`**`-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`은(는) 해당 클러스터가 기본 클러스터로 승격될 때 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`이(가) 됩니다.

RDS 프록시를 사용하는 경우, 애플리케이션의 쓰기 작업을 새 기본 클러스터와 연결된 프록시의 적절한 읽기/쓰기 엔드포인트로 리디렉션해야 합니다. 이 프록시 엔드포인트는 기본 엔드포인트이거나 사용자 지정 읽기/쓰기 엔드포인트일 수 있습니다. 자세한 내용은 [RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식]을 참조하세요.

글로벌 데이터베이스 클러스터의 기존 토폴로지를 복원하기 위해 Aurora는 기존 기본 리전의 가용성을 모니터링합니다. 해당 리전이 정상이고 다시 사용 가능한 상태가 되면 Aurora는 자동으로 글로벌 클러스터에 보조 리전으로 다시 추가합니다. Aurora는 장애가 발생하는 시점에 기존 스토리지 볼륨의 스냅샷을 만들고자 시도한 후, 기존 기본 리전에 새 스토리지 볼륨을 생성합니다. 이렇게 하면 누락된 데이터를 복구하는 데 사용할 수 있습니다. 이 작업이 성공하면 Aurora는 이름이 `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`*인 스냅샷을 생성합니다. 이 스냅샷은 AWS Management Console의 스냅샷 섹션에서 찾을 수 있습니다. 또한 이 스냅샷은 [DescribeDBClusterSnapshots] API 작업에서 반환된 정보에 나열되어 있으니 해당 위치에서 확인할 수 있습니다.

> **Note**
>
> 기존 스토리지 볼륨의 스냅샷은 시스템 스냅샷으로, 기존 기본 클러스터에 구성된 백업 보존 기간이 적용됩니다. 보존 기간 이후에도 이 스냅샷을 보존하려면 스냅샷을 복사하여 수동 스냅샷으로 저장할 수 있습니다. 요금 등 스냅샷 복사에 대해 자세히 알아보려면 DB 클러스터 스냅샷 복사 단원을 참조하세요.

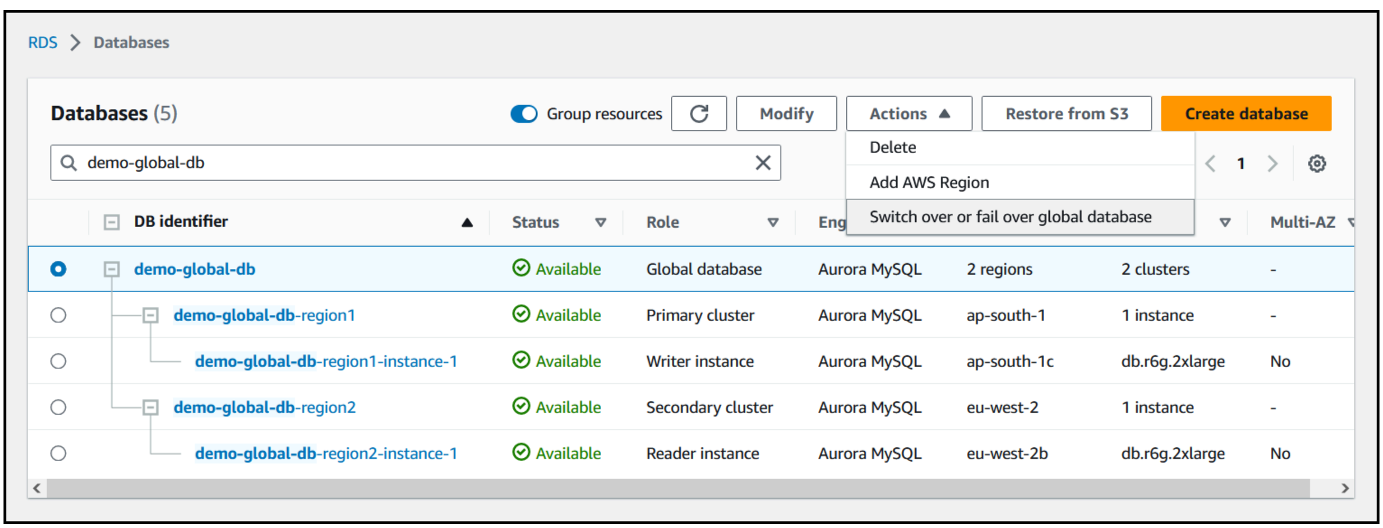
기존 토폴로지가 복원된 후 비즈니스 및 워크로드에 가장 적합한 시점에 전환 작업을 수행하여 글로벌 데이터베이스를 기존의 기본 리전으로 페일백할 수 있습니다. 이렇게 하려면 Amazon Aurora Global Database에서 전환 수행 단원의 절차를 따르세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora Global Database에서 장애 조치를 수행할 수 있습니다.

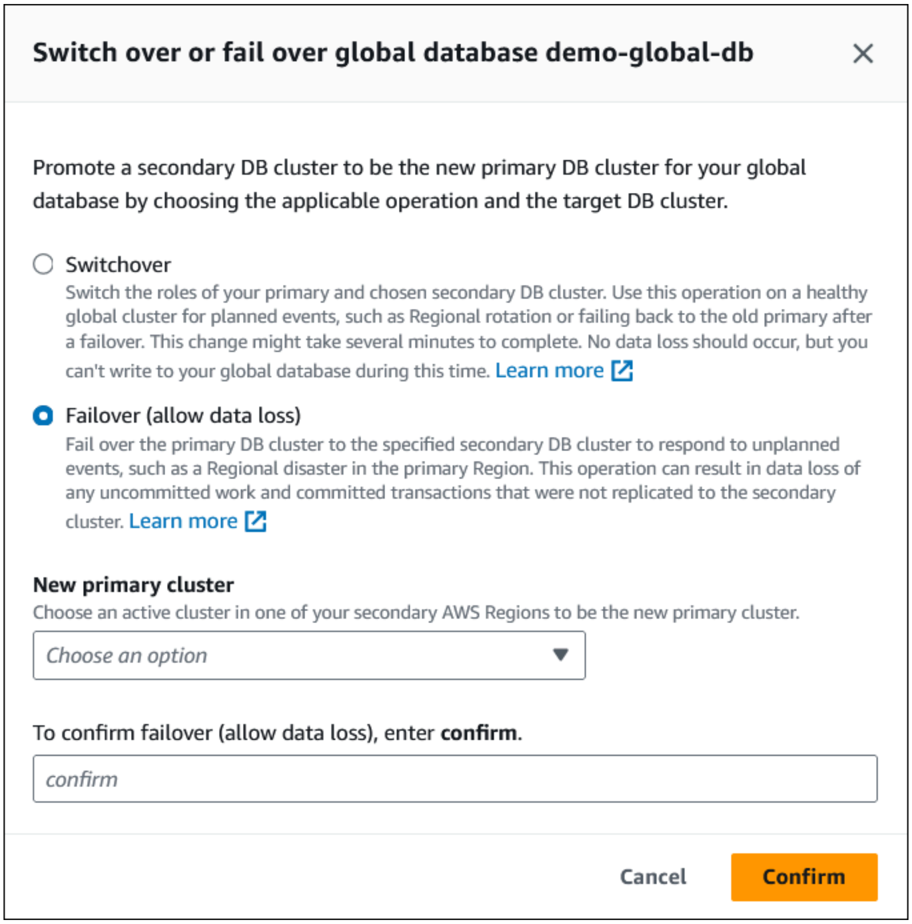
콘솔

Aurora Global Database에서 관리형 장애 조치를 수행하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 장애 조치를 수행할 Aurora Global Database를 찾습니다.
3. 작업 메뉴에서 글로벌 데이터베이스 전환 또는 장애 조치를 선택합니다.

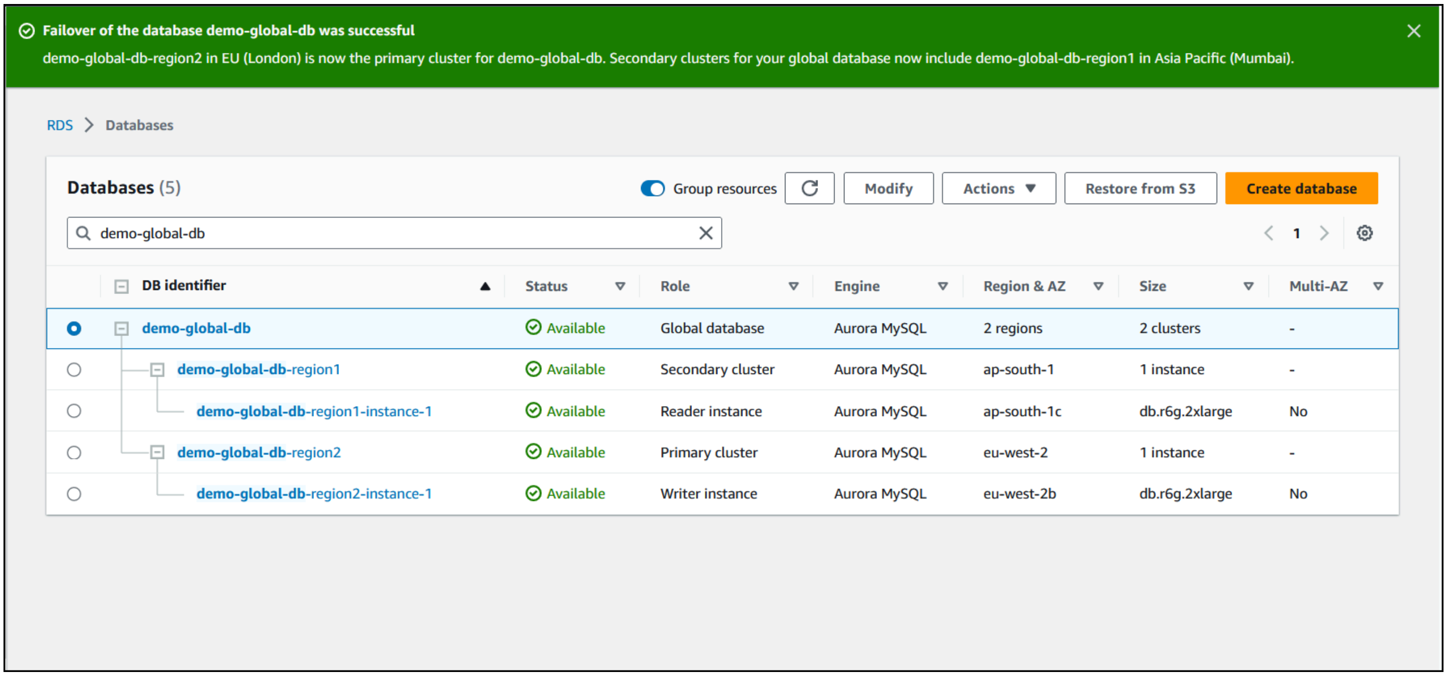


4. 장애 조치(데이터 손실 허용)를 선택합니다.



5. 새 기본 클러스터에서 보조 AWS 리전 중 하나의 활성 클러스터를 새 기본 클러스터로 선택합니다.
6. **`confirm`**을 입력한 후 확인을 선택합니다.

장애 조치가 완료되면 다음 이미지와 같이 데이터베이스 목록에서 Aurora DB 클러스터와 해당 클러스터의 현재 상태를 확인할 수 있습니다.



AWS CLI

Aurora Global Database에서 관리형 장애 조치를 수행하는 방법

`failover-global-cluster` CLI 명령을 사용하여 Aurora Global Database의 장애 조치를 수행합니다. 명령을 사용하여 다음 파라미터에 대한 값을 전달합니다.

- `--region` – Aurora Global Database의 새 기본 클러스터로 사용하려는 보조 DB 클러스터가 실행 중인 AWS 리전을 지정합니다.
- `--global-cluster-identifier` – Aurora 전역 데이터베이스의 이름을 지정합니다.
- `--target-db-cluster-identifier` – Aurora Global Database의 새 기본 클러스터로 승격시킬 Aurora DB 클러스터의 Amazon 리소스 이름(ARN)을 지정합니다.
- `--allow-data-loss` – 명시적으로 전환 작업이 아닌 장애 조치 작업으로 설정합니다. 비동기식 복제 구성 요소에서 모든 복제된 데이터가 보조 리전으로 전송되는 작업이 완료되지 않으면 장애 조치 작업으로 인해 일부 데이터가 손실될 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

Windows의 경우:

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

RDS API

Aurora Global Database에서 장애 조치를 수행하려면 FailoverGlobalCluster API 작업을 실행합니다.

Aurora Global Database에서 수동 장애 조치 수행

일부 시나리오에서는 관리형 장애 조치 프로세스를 사용하지 못할 수 있습니다. 일례로 기본 및 보조 DB 클러스터가 호환되는 엔진 버전을 실행하지 않는 경우를 들 수 있습니다. 이 경우 본 수동 프로세스에 따라 대상 보조 리전으로 장애 조치를 수행할 수 있습니다.

> **Tip**
>
> 이 프로세스를 사용하기 전에 프로세스를 이해하는 것이 좋습니다. 리전 전반의 문제의 첫 신호가 나타날 때 신속하게 진행할 수 있도록 계획을 세우세요. Amazon CloudWatch를 정기적으로 사용하여 보조 클러스터의 지연 시간을 추적하면 복제 작업 지연이 최소한으로 발생하는 보조 리전을 식별하는 데 대비할 수 있습니다. 계획을 테스트하여 절차가 완전하고 정확한지 여부와 직원이 실제로 이러한 일이 발생하기 전에 재해 복구 장애 조치를 수행할 수 있도록 교육을 받았는지 확인합니다.

기본 리전에서 계획되지 않은 중단이 발생한 후 보조 클러스터로 수동 장애 조치를 수행하는 방법

1. 중단 시 AWS 리전의 기본 Aurora DB 클러스터에 대한 DML 문 및 기타 쓰기 작업을 중지합니다.
2. 보조 AWS 리전에서 새 기본 DB 클러스터로 사용할 Aurora DB 클러스터를 식별합니다. Aurora Global Database에 2개 이상의 보조 AWS 리전이 있는 경우, 복제 지연 시간이 가장 적은 보조 클러스터를 선택합니다.
3. 선택한 보조 DB 클러스터를 Aurora 전역 데이터베이스에서 분리합니다.

   Aurora 전역 데이터베이스에서 보조 DB 클러스터를 제거하면 기본 클러스터에서 이 보조 클로스터로의 복제가 즉시 중지되고 전체 읽기/쓰기 기능이 있는 프로비저닝된 독립형 Aurora DB 클러스터로 승격됩니다. 중단 시 리전의 기본 클러스터와 연결된 다른 보조 Aurora DB 클러스터는 계속

사용할 수 있으며 애플리케이션의 호출을 수락할 수 있습니다. 클러스터는 리소스도 소비합니다. Aurora 전역 데이터베이스를 다시 생성하므로, 다음 단계에서 새 Aurora 전역 데이터베이스를 생성하기 전에 다른 보조 DB 클러스터를 제거하세요. 이렇게 하면 Aurora 전역 데이터베이스의 DB 클러스터 간의 데이터 불일치(브레인 분할 문제)를 피할 수 있습니다.

분리 단계에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거 단원을 참조하세요.

4. 새로운 엔드포인트를 사용하여 현재의 이 독립형 Aurora DB 클러스터로 모든 쓰기 작업을 전송하도록 애플리케이션을 재구성합니다. Aurora 전역 데이터베이스를 생성할 때 제공된 이름을 수락한 경우, 애플리케이션에 있는 클러스터의 엔드포인트 문자열에서 `-ro`을(를) 제거하여 엔드포인트를 변경할 수 있습니다.

   예를 들어, 보조 클러스터의 엔드포인트 `my-global.cluster-`**`ro`**`-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`은(는) 해당 클러스터가 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 전역 데이터베이스에서 분리될 때 Aurora 이(가) 됩니다.

   이 Aurora DB 클러스터는 다음 단계에서 리전을 추가하기 시작하면 새 Aurora 전역 데이터베이스의 기본 클러스터가 됩니다.

   RDS 프록시를 사용하는 경우, 애플리케이션의 쓰기 작업을 새 기본 클러스터와 연결된 프록시의 적절한 읽기/쓰기 엔드포인트로 리디렉션해야 합니다. 이 프록시 엔드포인트는 기본 엔드포인트이거나 사용자 지정 읽기/쓰기 엔드포인트일 수 있습니다. 자세한 내용은 RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식을 참조하세요.

5. DB 클러스터에 AWS 리전을 추가합니다. 이렇게 하면 기본 클러스터에서 보조 클러스터로의 복제 프로세스가 시작됩니다. 리전을 추가하는 자세한 단계는 Amazon Aurora Global Database에 AWS 리전 추가 단원을 참조하세요.

6. 필요에 따라 더 많은 AWS 리전를 추가하여 애플리케이션을 지원하는 데 필요한 토폴로지를 다시 생성합니다.

애플리케이션 쓰기가 이러한 변경 전, 중, 후에 올바른 Aurora DB 클러스터로 전송되는지 확인합니다. 이렇게 하면 Aurora 전역 데이터베이스의 DB 클러스터 간의 데이터 불일치(브레인 분할 문제)를 피할 수 있습니다.

AWS 리전에서의 중단에 대한 응답으로 재구성한 경우, 중단이 해결된 후 AWS 리전을 기본으로 되돌릴 수 있습니다. 이렇게 하려면 새 글로벌 데이터베이스에 이전 AWS 리전을 추가한 다음, 전환 프로세스를 사용하여 해당 역할을 전환합니다. Aurora Global Database에서는 전환을 지원하는 Aurora

PostgreSQL 또는 Aurora MySQL 버전을 사용해야 합니다. 자세한 내용은 [Amazon Aurora Global Database에서 전환 수행](#) 섹션을 참조하세요.

Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO 관리

Aurora PostgreSQL 기반 글로벌 데이터베이스를 사용하면 `rds.global_db_rpo` 파라미터를 사용하여 Aurora Global Database의 Recovery Point Objective(RPO)를 관리할 수 있습니다. RPO는 중단 이벤트로 인해 손실될 수 있는 최대 데이터 양을 나타냅니다.

Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO를 설정하면 Aurora은(는) 모든 보조 클러스터의 RPO 지연 시간을 모니터링하여 하나 이상의 보조 클러스터가 대상 RPO 기간 내에 있도록 합니다. RPO 지연 시간은 또 다른 시간 기반 지표입니다.

RPO는 장애 조치 후 데이터베이스가 새 AWS 리전에서 작업을 재개할 때 사용됩니다. Aurora는 다음과 같이 기본 클러스터에서 트랜잭션을 커밋(또는 차단)하기 위해 RPO 및 RPO 지연 시간을 평가합니다.

- 하나 이상의 보조 DB 클러스터의 RPO 지연 시간이 이 RPO보다 적은 경우 트랜잭션을 커밋합니다.
- 모든 보조 DB 클러스터의 RPO 지연 시간이 이 RPO보다 큰 경우 트랜잭션을 차단합니다. 또한 이벤트를 PostgreSQL 로그 파일에 기록하고 차단된 세션을 보여주는 "대기" 이벤트를 내보냅니다.

즉, 모든 보조 클러스터가 대상 RPO보다 뒤처지는 경우, Aurora은(는) 보조 클러스터 중 하나 이상이 캐치될 때까지 기본 클러스터에서 트랜잭션을 일시 중지합니다. 하나 이상의 세컨더리 데이터베이스 클러스터의 지연 시간이 이 RPO보다 적으면 즉시 일시 중지된 트랜잭션이 재개되고 다시 커밋됩니다. 결과적으로 RPO가 충족될 때까지 트랜잭션이 커밋될 수 없습니다.

`rds.global_db_rpo` 형식은 동적 파라미터입니다. 지연 시간이 충분히 감소할 때까지 모든 쓰기 트랜잭션이 중단되지 않도록 하려는 경우, 신속하게 재설정할 수 있습니다. 이 경우 Aurora는 잠시 대기한 후 변경 사항을 인식하고 구현합니다.

> **Important**
>
> AWS 리전이 두 개뿐인 글로벌 데이터베이스에서는 `rds.global_db_rpo` 파라미터의 기본값을 보조 리전의 파라미터 그룹에 유지하는 것이 좋습니다. 그렇지 않으면 기본 AWS 리전이 손실되어 장애 조치를 수행할 경우 Aurora가 트랜잭션을 일시 중지할 수 있습니다. 대신 Aurora가 이전 장애 AWS 리전에서 클러스터 재구축을 완료할 때까지 기다렸다가 이 파라미터를 변경하여 최대 RPO를 적용합니다.

이 파라미터를 다음에 설명된 대로 설정하면 해당 파라미터가 생성하는 지표도 모니터링할 수 있습니다. `psql` 또는 다른 도구를 사용하여 Aurora 전역 데이터베이스의 기본 DB 클러스터를 쿼리하고 Aurora PostgreSQL–기반 전역 데이터베이스 작업에 대한 자세한 정보를 얻을 수 있습니다. 자세한 방법은 Aurora PostgreSQL 기반 글로벌 데이터베이스 모니터링 단원을 참조하십시오.

주제

- 복구 시점 목표 설정
- 복구 시점 목표 보기
- 복구 시점 목표 비활성화

복구 시점 목표 설정

`rds.global_db_rpo` 파라미터는 PostgreSQL 데이터베이스에 대한 RPO 설정을 제어합니다. 이 파라미터는 Aurora PostgreSQL에서 지원됩니다. `rds.global_db_rpo`에 대한 유효한 값의 범위는 20초~2,147,483,647초(68년)입니다. 비즈니스 요구 사항과 사용 사례에 맞는 현실적인 값을 선택하세요. 예를 들어, RPO에 대해 최대 10분 허용할 수 있습니다. 이 경우, 값을 600으로 설정합니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora PostgreSQL–기반 글로벌 데이터베이스에 대해 이 값을 설정할 수 있습니다.

콘솔

RPO를 설정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Aurora 전역 데이터베이스의 기본 클러스터를 선택하고 구성 탭을 열어 DB 클러스터 파라미터 그룹을 찾습니다. 예를 들어, Aurora PostgreSQL 11.7을 실행하는 기본 DB 클러스터의 기본 파라미터 그룹은 `default.aurora-postgresql11`입니다.

   파라미터 그룹은 직접 편집할 수 없습니다. 대신, 다음 작업을 수행합니다.

   - 적절한 기본 파라미터 그룹을 시작 지점으로 사용하여 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 예를 들어, `default.aurora-postgresql11`을(를) 기반으로 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다.
   - 사용자 지정 DB 파라미터 그룹에서 사용 사례에 맞게 rds.global_db_rpo 파라미터 값을 설정합니다. 유효한 값은 20초에서 최대 정수 값인 2,147,483,647(68년)까지입니다.
   - 수정된 DB 클러스터 파라미터 그룹을 Aurora DB 클러스터에 적용합니다.

자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 섹션을 참조하세요.

AWS CLI

`rds.global_db_rpo` 파라미터를 설정하려면 modify-db-cluster-parameter-group CLI 명령을 사용합니다. 명령에서, 기본 클러스터의 파라미터 그룹 이름과 RPO 파라미터 값을 지정합니다.

다음 예제에서는 `my_custom_global_parameter_group`이라는 기본 DB 클러스터 파라미터 그룹의 RPO를 600초(10분)로 설정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS API

`rds.global_db_rpo` 파라미터를 수정하려면 Amazon RDS ModifyDBClusterParameterGroup API 작업을 사용합니다.

복구 시점 목표 보기

전역 데이터베이스의 RPO(복구 지점 목표)는 각 DB 클러스터의 `rds.global_db_rpo` 파라미터에 저장됩니다. 보려고 하는 보조 클러스터의 엔드포인트에 연결하고 이 값에 대해 인스턴스를 쿼리하는 데 `psql`을(를) 사용할 수 있습니다.

```
db-name=>show rds.global_db_rpo;
```

이 파라미터를 설정하지 않으면 쿼리에서 다음을 반환합니다.

```
rds.global_db_rpo
--------------------
 -1
```

```
(1 row)
```

다음 응답은 1분 RPO 설정이 있는 보조 DB 클러스터에서 온 것입니다.

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

클러스터의 모든 `rds.global_db_rpo` 파라미터 값을 가져오기 위해 CLI를 사용하여 Aurora DB 클러스터에서 `user`이(가) 활성 상태인지 확인할 수 있는 값을 가져올 수도 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

Windows의 경우:

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
 --source user
```

이 명령은 모든 `user` 파라미터에 대해 다음과 유사한 출력을 반환합니다. 이는 `default-engine` 또는 `system` DB 클러스터 파라미터가 아닙니다.

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
```

```
                    "provisioned"
                ]
            }
        ]
}
```

클러스터 파라미터 그룹의 파라미터 보기에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 값 보기 단원을 참조하세요.

복구 시점 목표 비활성화

RPO를 비활성화하려면 `rds.global_db_rpo` 파라미터를 재설정합니다. AWS Management Console, AWS CLI 또는 RDS API를 사용하여 파라미터를 재설정할 수 있습니다.

콘솔

RPO를 비활성화하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 기본 DB 클러스터 파라미터 그룹을 선택합니다.
4. 파라미터 편집을 선택합니다.
5. rds.global_db_rpo 파라미터 옆의 상자를 선택합니다.
6. 재설정을 선택합니다.
7. 화면에 DB 파라미터 그룹의 파라미터 재설정이 표시되면 파라미터 재설정을 선택합니다.

콘솔에서 파라미터를 재설정하는 방법에 대한 자세한 내용은 Amazon Aurora에서 DB 클러스터 파라미터 그룹의 파라미터 수정 단원을 참조하십시오.

AWS CLI

`rds.global_db_rpo` 파라미터를 재설정하려면 reset-db-cluster-parameter-group 명령을 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
```

```
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

Windows의 경우:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS API

`rds.global_db_rpo` 파라미터를 재설정하려면 Amazon RDS API [ResetDBClusterParameterGroup](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ResetDBClusterParameterGroup.html) 작업을 사용합니다.

Global Database 보조 클러스터에 대한 교차 리전 복원력

Aurora PostgreSQL 버전 16.6, 15.10, 14.15, 13.18, 12.22 이상 및 Aurora MySQL 버전 3.09 이상에는 보조 리전 읽기 전용 복제본이 하드웨어 장애, 교차 AWS 리전 네트워크 중단, 클러스터 간 대량 데이터 전송 등과 같은 예기치 않은 이벤트 중에 서비스 연속성을 유지할 수 있도록 하는 가용성 개선이 포함되어 있습니다.

애플리케이션 요청에서 읽기 전용 복제본을 여전히 사용할 수 있지만 예기치 않은 이벤트가 해결될 때까지 복제 지연이 계속 증가할 수 있습니다. `AuroraGlobalDBProgressLag` CloudWatch 지표를 사용하여 기본 클러스터와 보조 클러스터 간의 지연을 모니터링할 수 있습니다. 클러스터 볼륨과 보조 클러스터의 DB 인스턴스 간의 지연을 포함하여 엔드투엔드 지연을 측정하려면 `AuroraGlobalDBProgressLag` 및 `AuroraReplicaLag` CloudWatch 지표의 값을 추가합니다. 지표에 대한 자세한 내용은 [Amazon Aurora용 지표 참조](#) 섹션을 참조하세요.

이러한 예기치 않은 이벤트 중에 Aurora MySQL 및 이전 버전의 Aurora PostgreSQL에 대한 Global Database 읽기 가용성이 영향을 받을 수 있습니다.

Aurora PostgreSQL 16.6, 15.10, 14.15, 13.18 및 12.22의 새로운 기능에 대한 자세한 내용은 Aurora PostgreSQL Release Notes의 [PostgreSQL 16.6](#)을 참조하세요.

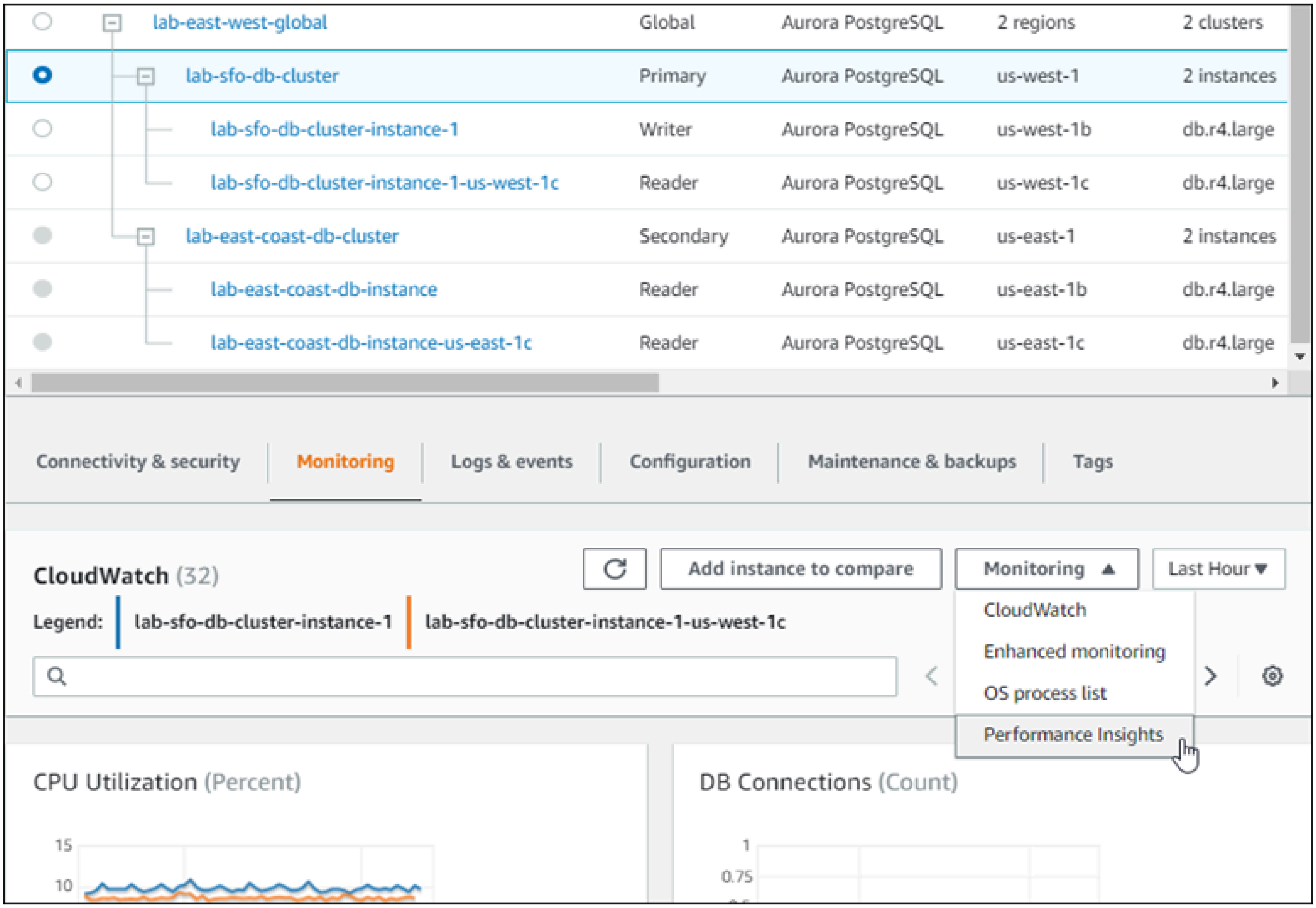
Aurora MySQL 버전 3.09 이상의 새로운 기능에 대한 자세한 내용은 Aurora MySQL 릴리스 정보의 [Database engine updates for Amazon Aurora MySQL version 3](#)를 참조하세요.

Amazon Aurora Global Database 모니터링

Aurora Global Database를 구성하는 Aurora DB 클러스터를 생성할 때 DB 클러스터의 성능을 모니터링할 수 있는 다양한 옵션을 선택할 수 있습니다. 옵션에는 다음이 포함됩니다.

- Amazon RDS 성능 인사이트 – 기본 Aurora 데이터베이스 엔진에서 성능 스키마를 활성화합니다. 성능 인사이트 및 Aurora Global Database에 대한 자세한 내용은 Amazon RDS 성능 인사이트를 사용하여 Amazon Aurora Global Database 모니터링 단원을 참조하십시오.
- 향상된 모니터링 – CPU의 프로세스 또는 스레드 사용률에 대한 메트릭을 생성합니다. 향상된 모니터링에 대한 자세한 내용은 Enhanced Monitoring을 사용하여 OS 지표 모니터링 섹을 참조하세요.
- Amazon CloudWatch Logs – 지정된 로그 유형을 CloudWatch Logs 에 게시합니다. 오류 로그는 기본적으로 게시되지만 Aurora 데이터베이스 엔진과 관련된 다른 로그를 선택할 수 있습니다.
  - Aurora MySQL–기반 Aurora DB 클러스터의 경우 감사 로그, 일반 로그 및 느린 쿼리 로그를 내보낼 수 있습니다.
  - Aurora PostgreSQL 기반 Aurora DB 클러스터의 경우 PostgreSQL 로그를 내보낼 수 있습니다.
- Aurora MySQL 기반 글로벌 데이터베이스의 경우 특정 `information_schema` 테이블을 쿼리하여 Aurora Global Database 및 해당 인스턴스의 상태를 확인할 수 있습니다. 자세한 방법은 Aurora MySQL 기반 글로벌 데이터베이스 모니터링을 참조하세요.
- Aurora PostgreSQL–기반 글로벌 데이터베이스의 경우 특정 함수를 사용하여 Aurora Global Database 및 해당 인스턴스의 상태를 확인할 수 있습니다. 자세한 방법은 Aurora PostgreSQL 기반 글로벌 데이터베이스 모니터링 단원을 참조하십시오.

다음 스크린샷은 Aurora Global Database에 있는 기본 Aurora DB 클러스터의 모니터링 탭에서 사용할 수 있는 몇 가지 옵션을 보여 줍니다.



자세한 내용은 Amazon Aurora 클러스터에서 지표 모니터링 단원을 참조하십시오.

Amazon RDS 성능 인사이트를 사용하여 Amazon Aurora Global Database 모니터링

Amazon RDS 성능 개선 도우미와 Aurora Global Database를 함께 사용할 수 있습니다. Aurora Global Database의 각 Aurora DB 클러스터에 대해 이 기능을 개별적으로 활성화할 수 있습니다. 이렇게 하려면 데이터베이스 생성 페이지의 추가 구성 섹션에서 성능 인사이트 사용을 선택합니다. 또는 Aurora DB 클러스터를 실행하여 실행한 후 이 기능을 사용하도록 수정할 수 있습니다. Aurora Global Database의 일부인 각 클러스터에 대해 성능 개선 도우미를 활성화하거나 비활성화할 수 있습니다.

성능 통찰력에서 만든 보고서는 글로벌 데이터베이스의 각 클러스터에 적용됩니다. 이미 성능 개선 도우미를 사용하고 있는 Aurora Global Database에 새로운 보조 AWS 리전을 추가하려면 새로 추가된 클러스터에서 성능 개선 도우미를 활성화해야 합니다. 기존 글로벌 데이터베이스에서 성능 개선 도우미 설정을 상속하지는 않습니다.

글로벌 데이터베이스에 연결된 DB 인스턴스에 대한 성능 개선 도우미 페이지를 보면서 AWS 리전를 전환할 수 있습니다. 그러나 AWS 리전를 전환한 직후에는 성능 정보를 보지 못할 수 있습니다. DB 인스턴스는 각 AWS 리전에서 이름이 동일할 수 있지만 연결된 성능 개선 도우미 URL은 DB 인스턴스마다 다릅니다. AWS 리전를 전환한 후에 성능 개선 도우미 탐색 창에서 DB 인스턴스의 이름을 다시 선택하세요.

글로벌 데이터베이스와 연결된 DB 인스턴스의 경우, 성능에 영향을 미치는 요인은 AWS 리전에 따라 다를 수 있습니다. 예를 들어 각 AWS 리전의 DB 인스턴스는 용량이 서로 다를 수 있습니다.

성능 인사이트 사용에 대한 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하십시오.

데이터베이스 활동 스트림을 사용하여 Aurora Global Database 모니터링

데이터베이스 활동 스트림을 사용하면 글로벌 데이터베이스의 DB 클러스터에서 감사 활동에 대한 경보를 모니터링하고 설정할 수 있습니다. 데이터베이스 활동 스트림을 각 DB 클러스터에서 개별적으로 시작합니다. 각 클러스터는 자체 AWS 리전 내의 자체 Kinesis 스트림에 감사 데이터를 제공합니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 단원을 참조하십시오.

Aurora MySQL 기반 글로벌 데이터베이스 모니터링

Aurora MySQL 기반 글로벌 데이터베이스의 상태를 보려면 information_schema.aurora_global_db_status 및 information_schema.aurora_global_db_instance_status 테이블을 쿼리합니다.

> **Note**
>
> `information_schema.aurora_global_db_status` 및 `information_schema.aurora_global_db_instance_status` 테이블은 Aurora MySQL 버전 3.04.0 이상의 글로벌 데이터베이스에서만 사용할 수 있습니다.

Aurora MySQL 기반 글로벌 데이터베이스를 모니터하는 방법

1. MySQL 클라이언트를 사용하여 글로벌 데이터베이스 기본 클러스터 엔드포인트에 연결합니다. 연결 방법에 대한 자세한 내용은 Amazon Aurora Global Database에 연결 단원을 참조하십시오.
2. mysql 명령의 `information_schema.aurora_global_db_status` 테이블을 쿼리하여 기본 볼륨과 보조 볼륨을 나열합니다. 이 쿼리는 다음 예와 같이 글로벌 데이터베이스 보조 DB 클러스터의 지연 시간을 반환합니다.

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
-----------+---------------------+-------------------------------
+-------------------------+--------------------------------
+------------------------
us-east-1  |           183537946 |                             0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                           428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

출력에는 다음 열이 포함된 글로벌 데이터베이스의 각 DB 클러스터에 대한 행이 포함됩니다.

- AWS_REGION – 이 DB 클러스터가 속한 AWS 리전입니다. 엔진별 AWS 리전을 나열한 테이블은 리전 가용성 섹션을 참조하세요.
- HIGHEST_LSN_WRITTEN – 이 DB 클러스터에 현재 작성된 가장 높은 로그 시퀀스 번호(LSN)입니다.

  LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
- DURABILITY_LAG_IN_MILLISECONDS - 보조 DB 클러스터의 `HIGHEST_LSN_WRITTEN`과 기본 DB 클러스터의 `HIGHEST_LSN_WRITTEN` 간 타임스탬프 값 차이입니다. 이 값은 Aurora Global Database의 기본 DB 클러스터에서 항상 0입니다.
- RPO_LAG_IN_MILLISECONDS – Recovery Point Objective(RPO) 지연 시간입니다. RPO 지연은 Aurora Global Databse의 기본 DB 클러스터에 저장된 후 가장 최근의 사용자 트랜잭션 COMMIT 보조 DB 클러스터에 저장하는 데 걸리는 시간입니다. 이 값은 Aurora Global Database의 기본 DB 클러스터에서 항상 0입니다.

  간단히 말해서 이 지표는 Aurora Global Database의 각 Aurora MySQL DB 클러스터에 대한 복구 시점 목표, 즉 중단 시 손실될 수 있는 데이터의 양을 계산합니다. 지연과 마찬가지로 RPO는 시간 단위로 측정됩니다.
- LAST_LAG_CALCULATION_TIMESTAMP – `DURABILITY_LAG_IN_MILLISECONDS` 및 `RPO_LAG_IN_MILLISECONDS`에 대한 값이 마지막으로 계산된 시점을 나타내는 타임스탬프입니다. 시간 값(예: `1970-01-01 00:00:00+00`)은 이것이 기본 DB 클러스터임을 의미합니다.

- OLDEST_READ_VIEW_TRX_ID - 라이터 DB 인스턴스가 삭제할 수 있는 가장 오래된 트랜잭션의 ID입니다.

3. `information_schema.aurora_global_db_instance_status` 테이블을 쿼리하여 기본 DB 클러스터와 보조 DB 클러스터 모두에 대한 모든 보조 DB 인스턴스를 나열합니다.

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID             |               SESSION_ID               | AWS_REGION
 | DURABLE_LSN  | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
---------------------+--------------------------------------+------------
+-------------+----------------------+------------------------
+----------------------+-----------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                    0 |                       0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                      0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                    677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |            183537698 |                20806930 |
 183537691 |                     21
(4 rows)
```

출력에는 다음 열이 포함된 글로벌 데이터베이스의 각 DB 인스턴스에 대한 행이 포함됩니다.

- SERVER_ID – DB 인스턴스의 서버 식별자입니다.
- SESSION_ID – 현재 세션의 고유 식별자입니다. `MASTER_SESSION_ID`의 값은 Writer(프라이머리) DB 인스턴스를 식별합니다.
- AWS_REGION - 이 DB 인스턴스가 속한 AWS 리전입니다. 엔진별 AWS 리전을 나열한 테이블은 리전 가용성 섹션을 참조하세요.
- DURABLE_LSN – 스토리지에서 뛰어난 내구성을 갖게 된 LSN입니다.
- HIGHEST_LSN_RECEIVED – 라이터 DB 인스턴스에서 DB 인스턴스가 수신한 가장 높은 LSN입니다.

- OLDEST_READ_VIEW_TRX_ID - 라이터 DB 인스턴스가 삭제할 수 있는 가장 오래된 트랜잭션의 ID입니다.
- OLDEST_READ_VIEW_LSN – DB 인스턴스가 스토리지에서 읽는 데 사용하는 가장 오래된 LSN입니다.
- VISIBILITY_LAG_IN_MSEC - 기본 DB 클러스터의 리더의 경우 이 DB 인스턴스가 라이터 DB 인스턴스보다 얼마나 뒤처지는지 밀리초 단위로 나타낸 것입니다. 보조 DB 클러스터의 리더의 경우 이 DB 인스턴스가 보조 볼륨보다 얼마나 뒤처지는지 밀리초 단위로 나타낸 것입니다.

이러한 값이 시간에 따라 어떻게 바뀌는지 확인하려면 테이블 삽입에 한 시간이 걸리는 다음 트랜잭션 블록을 살펴보십시오.

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

경우에 따라 BEGIN 문 이후 기본 DB 클러스터와 보조 DB 클러스터 사이에 네트워크 연결 끊김이 발생할 수 있습니다. 그렇다면 보조 DB 클러스터의 DURABILITY_LAG_IN_MILLISECONDS 값이 증가하기 시작합니다. INSERT 명령문 마지막에 DURABILITY_LAG_IN_MILLISECONDS의 값이 1시간입니다. 하지만 기본 DB 클러스터와 보조 DB 클러스터 간에 커밋된 모든 사용자 데이터가 여전히 동일하기 때문에 RPO_LAG_IN_MILLISECONDS 값은 0입니다. COMMIT 명령문이 완료되는 즉시 RPO_LAG_IN_MILLISECONDS 값이 증가합니다.

Aurora PostgreSQL 기반 글로벌 데이터베이스 모니터링

Aurora PostgreSQL 기반 글로벌 데이터베이스의 상태를 보려면 `aurora_global_db_status` 및 `aurora_global_db_instance_status` 함수를 사용합니다.

> **Note**
>
> Aurora PostgreSQL만 `aurora_global_db_status` 및 `aurora_global_db_instance_status` 함수를 지원합니다.

Aurora PostgreSQL기반 글로벌 데이터베이스를 모니터하는 방법

1. psql과 같은 PostgreSQL 유틸리티를 사용하여 글로벌 데이터베이스 기본 클러스터 엔드포인트에 연결합니다. 연결 방법에 대한 자세한 내용은 [Amazon Aurora Global Database에 연결](#) 단원을 참조하십시오.

2. psql 명령의 `aurora_global_db_status` 함수를 사용하여 기본 볼륨과 보조 볼륨을 나열합니다. 이는 글로벌 데이터베이스 보조 DB 클러스터의 지연 시간을 보여 줍니다.

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+---------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |             0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

출력에는 다음 열이 포함된 글로벌 데이터베이스의 각 DB 클러스터에 대한 행이 포함됩니다.

- aws_region - 이 DB 클러스터가 있는 AWS 리전입니다. 엔진별 AWS 리전을 나열한 테이블은 [리전 가용성] 섹션을 참조하세요.
- highest_lsn_written – 이 DB 클러스터에 현재 작성된 가장 높은 로그 시퀀스 번호(LSN)입니다.

  LSN(로그 시퀀스 번호)은 데이터베이스 트랜잭션 로그의 레코드를 식별하는 고유한 순차적 번호입니다. LSN은 더 큰 LSN이 더 이후의 트랜잭션을 나타내도록 정렬됩니다.
- durability_lag_in_msec – 보조 DB 클러스터에 작성된 가장 높은 로그 시퀀스 번호(`highest_lsn_written`)와 기본 DB 클러스터의 `highest_lsn_written` 간의 타임스탬프 차이입니다.
- rpo_lag_in_msec – 복구 시점 목표(RPO) 지연 시간입니다. 이 지연 시간은 보조 DB 클러스터에 저장된 가장 최근의 사용자 트랜잭션 커밋과 기본 DB 클러스터에 저장된 가장 최근의 사용자 트랜잭션 커밋 간의 시간 차이입니다.
- last_lag_calculation_time – `durability_lag_in_msec` 및 `rpo_lag_in_msec`에 대한 값이 마지막으로 계산된 타임스탬프입니다.
- feedback_epoch – 보조 DB 클러스터가 상시 대기 방식 정보를 생성할 때 사용하는 epoch입니다.

  핫 스탠바이는 서버가 복구 또는 대기 모드에 있는 동안 DB 클러스터가 연결하고 쿼리할 수 있는 때입니다. 핫 스탠바이 피드백은 핫 스탠바이 상태일 때 DB 클러스터에 대한 정보입니다. 자세한 내용은 PostgreSQL 설명서의 [핫 스탠바이]를 참조하십시오.

- feedback_xmin – 보조 DB 클러스터가 사용하는 최소(가장 오래된) 활성 트랜잭션 ID입니다.

3. aurora_global_db_instance_status 함수를 사용하여 기본 DB 클러스터와 보조 DB 클러스터 모두에 대한 모든 보조 DB 인스턴스를 나열합니다.

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
---------------------------------------------+-------------------------------------
+------------+------------+-----------------+----------------+---------------
+----------------------+------------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                 |               |
                      |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |               2 |    3315479243 |
        93763985095 |                     10
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |               2 |    3315479243 |
        93763985089 |                   1017
(3 rows)
```

출력에는 다음 열이 포함된 글로벌 데이터베이스의 각 DB 인스턴스에 대한 행이 포함됩니다.

- server_id – DB 인스턴스의 서버 식별자입니다.
- session_id – 현재 세션의 고유 식별자입니다.
- aws_region - 이 DB 인스턴스가 있는 AWS 리전입니다. 엔진별 AWS 리전을 나열한 테이블은 리전 가용성 섹션을 참조하세요.
- durable_lsn – 스토리지에서 뛰어난 내구성을 갖도록 만든 LSN입니다.
- highest_lsn_rcvd – 라이터 DB 인스턴스에서 DB 인스턴스가 수신한 가장 높은 LSN입니다.
- feedback_epoch – DB 인스턴스가 핫 스탠바이 정보를 생성할 때 사용하는 epoch입니다.

  상시 대기 상황은 서버가 복구 또는 대기 모드에 있는 동안 DB 인스턴스가 연결하고 쿼리할 수 있는 기간입니다. 핫 스탠바이 피드백은 핫 스탠바이 상태일 때 DB 인스턴스에 대한 정보입니다. 자세한 내용은 핫 스탠바이에 관한 PostgreSQL 설명서를 참조하십시오.
- feedback_xmin – DB 인스턴스가 사용하는 최소(가장 오래된) 활성 트랜잭션 ID입니다.

- oldest_read_view_lsn – DB 인스턴스가 스토리지에서 읽는 데 사용하는 가장 오래된 LSN입니다.
- visibility_lag_in_msec – 이 DB 인스턴스가 라이터 DB 인스턴스보다 지연된 시간입니다.

이러한 값이 시간에 따라 어떻게 바뀌는지 확인하려면 테이블 삽입에 한 시간이 걸리는 다음 트랜잭션 블록을 살펴보십시오.

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

경우에 따라 `BEGIN` 문 이후 기본 DB 클러스터와 보조 DB 클러스터 사이에 네트워크 연결 끊김이 발생할 수 있습니다. 이러한 경우 보조 DB 클러스터의 `durability_lag_in_msec` 값이 증가하기 시작합니다. `INSERT` 문의 끝에서 `durability_lag_in_msec` 값은 1시간입니다. 하지만 기본 DB 클러스터와 보조 DB 클러스터 간에 커밋된 모든 사용자 데이터가 여전히 동일하기 때문에 이 `rpo_lag_in_msec` 값은 0입니다. `COMMIT` 문이 완료되는 즉시 `rpo_lag_in_msec` 값이 증가합니다.

다른 AWS 서비스와 함께 Amazon Aurora Global Database 사용

Aurora Global Database를 Amazon S3 및 AWS Lambda와 같은 다른 AWS 서비스와 함께 사용할 수 있습니다. 이렇게 하려면 글로벌 데이터베이스의 모든 Aurora DB 클러스터가 각 AWS 리전에서 동일한 권한, 외부 함수 등을 가져야 합니다. Aurora 글로벌 데이터베이스의 읽기 전용 Aurora 보조 DB 클러스터를 기본 역할로 승격할 수 있으므로 Aurora 글로벌 데이터베이스에서 사용하려는 모든 서비스에 대해 모든 Aurora DB 클러스터에서 미리 쓰기 권한을 설정하는 것이 좋습니다.

다음 절차에서는 각 AWS 서비스에 대해 수행할 작업을 요약합니다.

Aurora Global Database에서 AWS Lambda 함수를 호출하려면

1. Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출의 절차를 수행합니다.
2. Aurora 글로벌 데이터베이스의 각 클러스터에 대해 새 (IAM) 역할의 IAM (ARN) 을 설정합니다.
3. Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 Lambda 함수를 호출하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 Aurora 글로벌 데이터베이스의 각 클러스터와 연결합니다.

4. Lambda로의 발신 연결을 허용하도록 Aurora 글로벌 데이터베이스의 각 클러스터를 구성합니다. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

Amazon S3에서 데이터를 로드하는 방법

1. Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드의 절차를 수행합니다.
2. 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 `aurora_load_from_s3_role` 또는 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정합니다. `aurora_load_from_s3_role`에 대해 지정된 IAM 역할이 없는 경우, Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.
3. Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 S3에 액세스하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.
4. S3로 아웃바운드 연결을 허용하도록 글로벌 데이터베이스의 각 Aurora 클러스터를 구성합니다. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

쿼리된 데이터를 Amazon S3 저장하는 방법

1. Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 또는 Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기의 절차를 수행합니다.
2. 글로벌 데이터베이스의 각 Aurora 클러스터에 대해 `aurora_select_into_s3_role` 또는 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정합니다. `aurora_select_into_s3_role`에 대해 지정된 IAM 역할이 없는 경우, Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.
3. Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 S3에 액세스하도록 허용하려면 Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성에서 생성한 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.
4. S3로 아웃바운드 연결을 허용하도록 글로벌 데이터베이스의 각 Aurora 클러스터를 구성합니다. 지침은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 섹션을 참조하세요.

Amazon Aurora Global Database 업그레이드

Aurora Global Database 업그레이드는 Aurora DB 클러스터를 업그레이드하는 것과 동일한 절차를 따릅니다. 하지만 프로세스를 시작하기 전에 주의해야 할 몇 가지 중요한 차이점이 다음에 있습니다.

기본 및 보조 DB 클러스터를 동일한 버전으로 업그레이드하는 것이 좋습니다. 기본 및 보조 DB 클러스터에 있는 메이저, 마이너, 패치 수준 엔진 버전이 동일한 경우 Aurora Global Database에서 관리형 리전 간 데이터베이스 장애 조치만 수행할 수 있습니다. 하지만 패치 수준은 마이너 엔진 버전에 따라 다를 수 있습니다. 자세한 내용은 관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성 단원을 참조하십시오.

메이저 버전 업그레이드

Amazon Aurora Global Database의 메이저 버전 업그레이드를 수행하는 경우, 글로벌 데이터베이스 클러스터를 포함하는 개별 클러스터 대신 글로벌 데이터베이스 클러스터를 업그레이드합니다.

Aurora PostgreSQL 글로벌 데이터베이스를 상위 메이저 버전으로 업그레이드하는 방법은 글로벌 데이터베이스에 대한 메이저 업그레이드 섹션을 참조하세요.

> **Note**
>
> Aurora PostgreSQL 기반 Aurora Global Database 사용 시 복구 시점 목표(RPO) 기능이 켜져 있는 경우 Aurora DB 엔진의 메이저 버전 업그레이드를 수행할 수 없습니다. RPO 기능에 대한 자세한 내용은 Aurora PostgreSQL–기반 전역 데이터베이스에 대한 RPO 관리 섹션을 참조하세요.

Aurora MySQL 글로벌 데이터베이스를 상위 메이저 버전으로 업그레이드하는 방법은 글로벌 데이터베이스에 대한 현재 위치 메이저 업그레이드 섹션을 참조하세요.

> **Note**
>
> Aurora MySQL 기반 Aurora Global Database 사용 시 `lower_case_table_names` 파라미터를 기본값으로 설정하고 글로벌 데이터베이스를 재부팅하는 경우에만 Aurora MySQL 버전 2에서 버전 3으로 현재 위치 업그레이드를 수행할 수 있습니다. `lower_case_table_names` 사용 시 Aurora MySQL 버전 3으로 메이저 버전 업그레이드를 수행하려면 다음 프로세스를 사용하세요.

1. 글로벌 클러스터에서 모든 보조 DB 클러스터를 제거합니다. Amazon Aurora 글로벌 데이터베이스에서 클러스터 제거 섹션의 단계를 따르세요.
2. 기본 리전의 엔진 버전을 Aurora MySQL 버전 3으로 업그레이드합니다. 현재 위치 업그레이드 수행 방법 섹션의 단계를 따르세요.
3. 글로벌 클러스터에 보조 리전을 추가합니다. Amazon Aurora Global Database에 AWS 리전 추가 섹션의 단계를 따르세요.

대신 스냅샷 복원 방법을 사용할 수도 있습니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 단원을 참조하십시오.

마이너 버전 업그레이드

Aurora Global Database에서 마이너 업그레이드를 수행하는 경우 기본 클러스터를 업그레이드하기 전에 모든 보조 클러스터를 업그레이드합니다.

Aurora PostgreSQL 글로벌 데이터베이스를 상위 마이너 버전으로 업그레이드하는 방법은 마이너 버전 업그레이드 수행 및 패치 적용 방법 섹션을 참조하세요. Aurora MySQL 글로벌 데이터베이스를 상위 마이너 버전으로 업그레이드하는 방법은 엔진 버전을 수정하여 Aurora MySQL 업그레이드 섹션을 참조하세요.

업그레이드를 수행하기 전에 다음 고려 사항을 검토하세요.

- 보조 클러스터의 마이너 버전을 업그레이드해도 기본 클러스터의 가용성이나 사용에는 어떤 영향도 미치지 않습니다.
- 보조 클러스터에 하나 이상의 DB 인스턴스가 있어야 마이너 업그레이드를 수행할 수 있습니다.
- Aurora MySQL Global Database를 버전 2.11.*로 업그레이드하려는 경우 기본 및 보조 DB 클러스터를 패치 수준을 포함하여 정확히 동일한 버전으로 업그레이드해야 합니다.
- Aurora PostgreSQL Global Database를 업그레이드하려는 경우 기본 및 보조 DB 클러스터를 정확히 동일한 버전 및 패치 수준으로 업그레이드해야 합니다. 패치 수준을 업데이트하려면 보조 클러스터에서 보류 중인 모든 유지 관리 작업을 적용합니다.
- 관리형 리전 간 데이터베이스 전환 또는 장애 조치를 지원하려면 기본 및 보조 DB 클러스터를 패치 수준을 비롯해 정확히 동일한 버전으로 업그레이드해야 할 수 있습니다. 이 요구 사항은 Aurora MySQL 및 일부 Aurora PostgreSQL 버전에 적용됩니다. 서로 다른 패치 수준을 실행하는 클러스터 간의 전환 및 장애 조치를 허용하는 버전 목록은 관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성 섹션을 참조하세요.

관리형 리전 간 전환 및 장애 조치를 위한 패치 수준 호환성

Aurora 글로벌 데이터베이스에서 다음과 같은 마이너 엔진 버전 중 하나를 실행 중인 경우, 기본 및 보조 DB 클러스터의 패치 수준이 일치하지 않아도 관리형 교차 리전 전환 또는 장애 조치를 수행할 수 있습니다. 이 목록에 있는 버전보다 이전 마이너 엔진 버전의 경우, 기본 및 보조 DB 클러스터는 동일한 메이저, 마이너 및 패치 수준을 실행하여 관리형 리전 간 전환 또는 장애 조치를 수행해야 합니다. 기본 클러스터, 보조 클러스터 또는 둘 다에 대한 업그레이드를 계획할 때는 다음 표의 버전 정보와 참고 사항을 검토해야 합니다.

> **Note**
>
> 계획되지 않은 수동 리전 간 장애 조치의 경우, 대상 보조 DB 클러스터가 기본 DB 클러스터와 동일한 메이저 및 마이너 엔진 버전을 실행하는 한 장애 조치 프로세스를 수행할 수 있습니다. 이 경우 패치 수준이 일치하지 않아도 됩니다.
> 엔진 버전에 동일한 패치 수준이 필요한 경우 Aurora Global Database에서 수동 장애 조치 수행에 나온 단계에 따라 수동으로 장애 조치를 수행할 수 있습니다.

데이터베이스 엔진	마이너 엔진 버전	Notes
Aurora MySQL	마이너 버전 없음	Aurora MySQL 마이너 버전 중 어느 것도 기본 및 보조 DB 클러스터 간 패치 수준이 다른 관리형 리전 간 전환 또는 장애 조치를 허용하지 않습니다.
Aurora PostgreSQL	• 버전 15 이상의 메이저 버전 • 버전 14.5 이상의 마이너 버전 • 버전 13.8 이상의 마이너 버전 • 버전 12.12 이상의 마이너 버전 • 버전 11.17 이상의 마이너 버전	이전 열에 나열된 엔진 버전을 사용하는 경우, 기본 DB 클러스터와 보조 DB 클러스터의 패치 수준이 서로 달라도 기본 DB 클러스터에서 보조 DB 클러스터로 관리형 리전 간 전환 또는 장애 조치를 수행할 수 있습니다. 이보다 이전 마이너 버전을 사용하는 경우 기본 및 보조 DB 클러스터의 패치 수준이 일치하는 경우에만 관리형 리전 간 전환 또는 장애 조치를 수행할 수 있습니다.

데이터베이스 엔진	마이너 엔진 버전	Notes
		Warning 글로벌 데이터베이스의 클러스터를 다음 패치 버전으로 업데이트하면 글로벌 데이터베이스의 모든 클러스터가 이러한 패치 버전 중 하나 또는 새로운 버전 중 하나를 실행할 때까지 교차 리전 전환 또는 장애 조치를 수행할 수 없습니다. • 패치 버전 16.1.6, 16.2.4, 16.3.2 및 16.4.2 • 패치 버전 15.3.8, 15.4.9, 15.5.6, 15.6.4, 15.7.2 및 15.8.2 • 패치 버전 14.8.8, 14.9.9, 14.10.6, 14.11.4, 14.12.2 및 14.13.2

Amazon RDS Proxy for Aurora

Amazon RDS 프록시를 사용하면 애플리케이션이 데이터베이스 연결을 풀링하고 공유하도록 허용하여 확장 기능을 향상할 수 있습니다. RDS 프록시는 애플리케이션 연결을 유지하면서 예비 DB 인스턴스에 자동으로 연결하여 데이터베이스 장애에 대한 애플리케이션의 복원력을 높입니다. RDS 프록시를 사용하면 데이터베이스에 대해 AWS Identity and Access Management(IAM) 인증을 사용하고 AWS Secrets Manager에 자격 증명을 안전하게 저장합니다.

RDS 프록시를 사용하여 예기치 않은 데이터베이스 트래픽 급증을 처리할 수 있습니다. 급증을 처리하지 않으면 이러한 현상으로 인해 연결을 초과 구독하거나 새 연결이 빠른 속도로 생성되어 문제가 발생할 수 있습니다. RDS 프록시는 데이터베이스 연결 풀을 설정하고 이 풀에서 연결을 재사용합니다. 이 접근 방식은 매번 새 데이터베이스 연결을 여는 데서 오는 메모리 및 CPU 오버헤드 를 방지합니다. 과다 구독으로부터 데이터베이스를 보호하기 위해 생성되는 데이터베이스 연결 수를 제어할 수 있습니다.

RDS 프록시는 연결 풀에서 즉시 제공할 수 없는 애플리케이션 연결을 대기열에 추가하거나 제한합니다. 대기 시간이 증가할 수 있지만 애플리케이션은 갑작스러운 데이터베이스 장애 또는 압도 없이 계속 확장될 수 있습니다. 연결 요청이 지정된 한도를 초과하는 경우 RDS Proxy는 애플리케이션 연결을 거부합니다(즉, 부하 감소). 동시에 RDS가 사용 가능한 용량으로 제공할 수 있는 부하에 대해 예측 가능한 성능을 유지합니다.

자격 증명을 처리하고 각 새 연결에 대한 보안 연결을 설정하는 데 필요한 오버헤드를 줄일 수 있습니다. RDS 프록시는 데이터베이스를 대신하여 해당 작업 중 일부를 처리할 수 있습니다.

RDS 프록시는 지원하는 엔진 버전과 완전히 호환됩니다. 코드 변경 없이 대부분의 애플리케이션에 RDS 프록시를 활성화할 수 있습니다. 지원되는 엔진 버전의 목록은 Amazon RDS 프록시를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

주제

- [Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링]
- [RDS 프록시 이벤트 작업]
- [RDS 프록시 명령줄 예제]
- [RDS 프록시 문제 해결]
- [RDS Proxy를 AWS CloudFormation에서 사용]
- [Aurora 글로벌 데이터베이스에 RDS 프록시 사용]

리전 및 버전 사용 가능 여부

데이터베이스 엔진 버전 지원 및 AWS 리전에 지정된 RDS 프록시의 가용성에 대한 자세한 내용은 [Amazon RDS 프록시를 지원하는 리전 및 Aurora DB 엔진] 단원을 참조하십시오.

RDS 프록시의 할당량 및 제한 사항

RDS Proxy에는 다음과 같은 제한 사항이 적용됩니다.

- 각 AWS 계정 ID는 프록시 20개로 제한됩니다. 애플리케이션에 더 많은 프록시가 필요한 경우 AWS Management Console의 Service Quotas 페이지에서 증량을 요청하세요. Service Quotas 페이지에서 Amazon Relational Database Service(Amazon RDS)를 선택하고 프록시를 찾아 할당량 증량을 요청합니다. AWS는 자동으로 할당량을 늘리거나 지원의 요청에 대한 검토를 보류할 수 있습니다.
- 각 프록시는 최대 200개의 연결된 Secrets Manager 암호를 보유할 수 있습니다. 따라서 각 프록시는 지정된 시간에 최대 200개의 서로 다른 사용자 계정으로 연결할 수 있습니다.
- 각 프록시에는 기본 엔드포인트가 있습니다. 또한 각 프록시에 최대 20개의 프록시 엔드포인트를 추가할 수 있습니다. 이러한 엔드포인트를 생성, 조회, 수정 및 삭제할 수 있습니다.
- Aurora 클러스터에서, 기본 프록시 엔드포인트를 사용하는 모든 연결은 Aurora 라이터 인스턴스에 의해 처리됩니다. 읽기 집약적 워크로드에 대한 로드 밸런싱을 수행하기 위해 프록시에 대한 읽기 전용 엔드포인트를 생성할 수 있습니다. 해당 엔드포인트는 클러스터의 리더 엔드포인트에 연결을 전달합니다. 이렇게 하면 프록시 연결에서 Aurora 읽기 확장성의 이점을 활용할 수 있습니다. 자세한 내용은 [프록시 엔드포인트 개요] 섹션을 참조하세요.
- Aurora Serverless v2 클러스터에서는 RDS 프록시를 사용할 수 있지만, Aurora Serverless v1 클러스터에서는 사용할 수 없습니다.
- RDS Proxy는 데이터베이스와 동일한 Virtual Private Cloud(VPC)에 있어야 합니다. 데이터베이스에는 공개적으로 액세스할 수는 있지만 프록시에는 공개적으로 액세스할 수 없습니다. 예를 들어, 로컬 호스트에서 데이터베이스의 프로토타입을 생성하는 경우 프록시에 연결하는 데 필요한 네트워크 요

구 사항을 설정하지 않는 한 프록시에 연결할 수 없습니다. 이는 로컬 호스트가 프록시의 VPC 외부에 있기 때문입니다.

> **Note**
>
> Aurora DB 클러스터의 경우 VPC 간 액세스를 활성화할 수 있습니다. 이렇게 하려면 프록시를 위한 추가 엔드포인트를 생성하고 해당 엔드포인트에 다른 VPC, 서브넷 및 보안 그룹을 지정합니다. 자세한 내용은 VPC 간에 Aurora 데이터베이스 액세스 단원을 참조하십시오.

- 테넌시가 `dedicated`로 설정된 VPC에서는 RDS 프록시를 사용할 수 없습니다.
- IAM 인증이 활성화된 Aurora DB 클러스터와 함께 RDS 프록시를 사용하는 경우 사용자 인증을 확인합니다. 프록시를 통해 연결하는 모든 사용자가 로그인 보안 인증 정보를 통해 인증되어야 합니다. Secrets Manager 및 RDS 프록시의 IAM 지원에 대한 자세한 내용은 AWS Secrets Manager에서 RDS 프록시용 데이터베이스 자격 증명 설정 및 RDS 프록시에 대한 AWS Identity and Access Management(IAM) 정책 설정 섹션을 참조하세요.
- SSL 호스트 이름 검증을 사용할 경우, 사용자 지정 DNS와 함께 RDS 프록시를 사용할 수 없습니다.
- 각 프록시는 단일 대상 DB 클러스터와 연결될 수 있습니다. 그러나 여러 프록시를 동일한 DB 클러스터와 연결할 수 있습니다.
- 텍스트 크기가 16KB보다 큰 문을 사용하면 프록시가 세션을 현재 연결에 고정합니다.
- 특정 리전에는 프록시를 만들 때 고려해야 하는 가용 영역(AZ) 제한이 있습니다. 미국 동부(버지니아 북부) 리전은 `use1-az3` 가용 영역에서 RDS 프록시를 지원하지 않습니다. 미국 서부(캘리포니아 북부) 리전은 `usw1-az2` 가용 영역에서 RDS 프록시를 지원하지 않습니다. 프록시 생성 시 서브넷을 선택할 때는 위에서 언급한 가용 영역에서 서브넷을 선택하지 않도록 하세요.
- 현재 RDS 프록시는 모든 전역 조건 컨텍스트 키를 지원하지 않습니다.

  전역 조건 컨텍스트 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.
- RDS Custom for SQL Server에는 RDS 프록시를 사용할 수 없습니다.
- 프록시에 데이터베이스 파라미터 그룹 수정을 반영하려면 변경 사항을 즉시 적용하기로 선택한 경우에도 인스턴스 재부팅이 필요합니다. 클러스터 수준 파라미터의 경우 클러스터 전체 재부팅이 필요합니다.
- 프록시 대상을 등록하면 프록시가 `rdsproxyadmin` DB 사용자를 자동으로 만듭니다. 프록시 기능에 필수적인 보호 대상 사용자입니다. 어떤 방식으로든 `rdsproxyadmin` 사용자의 변조를 방지해야 합니다. `rdsproxyadmin` 사용자 또는 해당 권한을 삭제하거나 수정하면 애플리케이션에서 프록시를 전혀 사용하지 못하게 될 수 있습니다.

각 DB 엔진의 추가 제한 사항은 다음 섹션을 참조하세요.

- Aurora MySQL 추가 제한 사항
- Aurora PostgreSQL 추가 제한 사항

Aurora MySQL 추가 제한 사항

Aurora MySQL 데이터베이스를 사용하는 RDS 프록시에는 다음과 같은 추가 제한 사항이 적용됩니다.

- `caching_sha2_password` 인증에 대한 RDS Proxy 지원에는 보안(TLS) 연결이 필요합니다.
- `caching_sha2_password`에 대한 RDS Proxy 지원에는 특정 go-sql 드라이버 버전과의 호환성 문제가 있는 것으로 알려져 있습니다.
- MySQL 8.4 C 드라이버를 사용할 때 파라미터 수가 준비된 문에서 자리 표시자 수를 초과하는 경우 `mysql_stmt_bind_named_param` API가 잘못된 형식의 패킷을 형성할 수 있습니다. 이로 인해 잘못된 응답이 발생합니다. 자세한 내용은 MySQL bug report를 참조하세요.
- 현재 모든 프록시는 MySQL에 대한 포트 3306에서 수신합니다. 프록시는 여전히 데이터베이스 설정에서 지정한 포트를 사용하여 데이터베이스에 연결됩니다.
- EC2 인스턴스에서 실행되는 자체 관리형 MySQL 데이터베이스에는 RDS 프록시를 사용할 수 없습니다.
- DB 파라미터 그룹의 `read_only` 파라미터가 `1`로 설정된 MySQL DB 인스턴스에는 RDS 프록시를 사용할 수 없습니다.
- RDS 프록시는 MySQL 압축 모드를 지원하지 않습니다. 예를 들어 `mysql` 명령의 `--compress` 또는 `-C` 옵션에서 사용하는 압축을 지원하지 않습니다.
- RDS 프록시가 동일한 데이터베이스 연결을 재사용하여 다른 쿼리를 실행하면 `GET DIAGNOSTIC` 명령을 처리하는 데이터베이스 연결에서 부정확한 정보가 반환될 수 있습니다. 이는 RDS 프록시가 데이터베이스 연결을 멀티플렉싱할 때 발생할 수 있습니다.
- `SET LOCAL` 같은 일부 SQL 문 및 함수는 고정을 야기하지 않고 연결 상태를 변경할 수 있습니다. 최신 고정 동작은 RDS 프록시 고정 방지 단원을 참조하십시오.
- 다중 문 쿼리에서는 `ROW_COUNT()` 함수를 사용할 수 없습니다.
- RDS 프록시는 하나의 TLS 레코드에서 여러 응답 메시지를 처리할 수 없는 클라이언트 애플리케이션을 지원하지 않습니다.
- RDS Proxy는 MySQL 이중 암호를 지원하지 않습니다.

> **Important**
>
> MySQL 데이터베이스와 관련된 프록시의 경우 초기화 쿼리에서 구성 파라미터 `sql_auto_is_null`을 `true`로 설정하거나 0이 아닌 값으로 설정하지 마시기 바랍니다. 이렇게 하면 잘못된 애플리케이션 동작이 발생할 수 있습니다.

Aurora PostgreSQL 추가 제한 사항

Aurora PostgreSQL 데이터베이스를 사용하는 RDS 프록시에는 다음과 같은 추가 제한 사항이 적용됩니다.

- RDS 프록시는 PostgreSQL에 대한 세션 고정 필터를 지원하지 않습니다.
- 현재 모든 프록시는 PostgreSQL에 대한 포트 5432에서 수신합니다.
- PostgreSQL의 경우 RDS 프록시는 현재 `CancelRequest`를 실행하여 클라이언트에서 쿼리를 취소하는 것을 지원하지 않습니다. Ctrl+C를 사용하여 대화형 psql 세션에서 장기 실행 쿼리를 취소하는 경우를 예로 들 수 있습니다.
- PostgreSQL 함수 lastval의 결과가 항상 정확하지는 않습니다. 해결 방법으로 `RETURNING` 절과 함께 INSERT 문을 사용합니다.
- RDS 프록시는 현재 스트리밍 복제 모드를 지원하지 않습니다.
- `default` 데이터베이스가 있어야 합니다.
- `ALTER ROLE`을 사용하여 `SET ROLE`로 사용자 역할을 변경하는 경우 해당 사용자와 프록시의 후속 연결에서 고정이 발생하는 경우 이 역할 설정이 사용되지 않을 수 있습니다. 이를 방지하려면 프록시를 사용할 때 프록시의 초기화 쿼리에 `SET ROLE`을 사용합니다. 자세한 내용은 Amazon Aurora용 프록시 만들기에서 초기화 쿼리 섹션을 참조하세요.

> **Important**
>
> PostgreSQL 데이터베이스를 사용하는 기존 프록시의 경우, `SCRAM`만 사용하도록 데이터베이스 인증을 수정하면 최대 60초 동안 프록시를 사용할 수 없게 됩니다. 문제를 방지하려면 다음 중 하나를 수행합니다.
>
> - 데이터베이스에서 `SCRAM` 및 `MD5` 인증이 모두 허용되는지 확인합니다.
> - `SCRAM` 인증만 사용하려면 새 프록시를 생성하고, 애플리케이션 트래픽을 새 프록시로 마이그레이션한 다음 이전에 데이터베이스와 연결된 프록시를 삭제합니다.

RDS Proxy 사용 대상 계획

RDS Proxy를 사용하여 가장 많은 혜택을 누릴 수 있는 DB 인스턴스, 클러스터 및 애플리케이션을 결정할 수 있습니다. 이렇게 하려면 다음 요소를 고려하십시오.

- '연결이 너무 많음' 오류가 발생한 DB 클러스터는 프록시와 연결하기에 좋은 후보입니다. 이는 종종 `ConnectionAttempts` CloudWatch 지표의 값이 높다는 특징이 있습니다. 프록시를 사용하면 애플리케이션은 많은 클라이언트 연결을 열 수 있고, 프록시는 DB 클러스터에 대한 장기 연결을 더 적게 관리합니다.
- T2 또는 T3 같이 더 작은 규모의 AWS 인스턴스 클래스를 사용하는 DB 클러스터의 경우 프록시를 사용하면 메모리 부족 상태를 방지할 수 있습니다. 또한 연결을 설정하는 데 필요한 CPU 오버헤드를 줄일 수 있습니다. 이러한 상태는 많은 수의 연결을 처리할 때 발생할 수 있습니다.
- 또한 특정 Amazon CloudWatch 지표를 모니터링하여 DB 클러스터에서 특정 유형의 제한에 접근하는지 확인할 수 있습니다. 이러한 제한은 연결 수와 연결 관리와 관련된 메모리에 대한 것입니다. 또한 특정 CloudWatch 지표를 모니터링하여 DB 클러스터에서 수명이 짧은 여러 연결을 처리하고 있는지 확인할 수 있습니다. 이러한 연결을 열고 닫으면 데이터베이스에 성능 오버헤드가 발생할 수 있습니다. 모니터링할 지표에 대한 자세한 내용은 Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링 단원을 참조하십시오.
- AWS Lambda 함수도 프록시를 사용하기 위한 좋은 후보가 될 수 있습니다. 이러한 함수는 RDS Proxy가 제공하는 연결 풀링의 이점을 누릴 수 있는 단기 데이터베이스 연결을 자주 만듭니다. Lambda 애플리케이션 코드에서 데이터베이스 자격 증명을 관리하는 대신 Lambda 함수에 대해 이미 가지고 있는 IAM 인증을 활용할 수 있습니다.
- 일반적으로 많은 수의 데이터베이스 연결을 열고 닫으며 내장된 연결 풀링 메커니즘이 없는 애플리케이션이 프록시를 사용하기에 적합합니다.
- 오랜 기간 동안 많은 수의 연결을 열린 상태로 유지하는 애플리케이션은 일반적으로 프록시를 사용하는 것이 좋습니다. SaaS(Software as a Service) 또는 전자 상거래와 같은 업종의 애플리케이션은 연결을 열린 상태로 두고 데이터베이스 요청에 대한 지연 시간을 최소화하는 경우가 많습니다. RDS 프록시를 사용하면 애플리케이션은 DB 클러스터에 직접 연결할 때보다 더 많은 연결을 열어 둘 수 있습니다.
- 모든 DB 클러스터에 대해 이러한 인증을 설정하기는 복잡하기 때문에 IAM 인증 Secrets Manager를 채택하지 않았을 수 있습니다. 그렇다면 기존 인증 방법을 그대로 두고 인증을 프록시에 위임할 수 있습니다. 프록시는 특정 애플리케이션에 대한 클라이언트 연결에 인증 정책을 적용할 수 있습니다. Lambda 애플리케이션 코드에서 데이터베이스 자격 증명을 관리하는 대신 Lambda 함수에 대해 이미 가지고 있는 IAM 인증을 활용할 수 있습니다.

- RDS 프록시를 사용하면 데이터베이스 장애에 대해 애플리케이션 복원력이 높아지고 투명성이 높아집니다. RDS 프록시는 도메인 이름 시스템(DNS) 캐시를 우회하여 Aurora 다중 AZ 데이터베이스의 장애 조치 시간을 최대 66% 단축합니다. 또한 RDS 프록시는 애플리케이션 연결을 유지하면서 트래픽을 새 데이터베이스 인스턴스로 자동 라우팅합니다. 따라서 애플리케이션의 장애 조치 투명성이 높아집니다.

RDS Proxy 개념 및 용어

RDS 프록시를 사용하여 Amazon Aurora DB 클러스터에 대한 연결 관리를 간소화할 수 있습니다.

RDS Proxy는 클라이언트 애플리케이션과 데이터베이스 사이의 네트워크 트래픽을 처리합니다. 데이터베이스 프로토콜을 이해하여 능동적으로 수행합니다. 그런 다음 애플리케이션의 SQL 작업과 데이터베이스의 결과 집합을 기반으로 동작을 조정합니다.

RDS Proxy는 데이터베이스의 연결 관리를 위한 메모리 및 CPU 오버헤드를 줄입니다. 애플리케이션이 많은 연결을 동시에 열 때 데이터베이스에 필요한 메모리 및 CPU 리소스가 더 적습니다. 또한 오랫동안 유휴 상태를 유지하는 연결을 닫았다가 다시 여는 데 애플리케이션의 논리가 필요하지 않습니다. 마찬가지로, 데이터베이스 문제의 경우 연결을 다시 설정하는 데 더 적은 애플리케이션 논리가 필요합니다.

RDS Proxy의 인프라는 고가용성이고 여러 가용 영역(AZ)에 배포됩니다. RDS 프록시의 계산, 메모리 및 스토리지는 Aurora DB 클러스터와 별개입니다. 이러한 분리는 데이터베이스 서버의 오버헤드를 줄여 데이터베이스 워크로드를 처리하는 데 리소스가 투입될 수 있습니다. RDS Proxy 계산 리소스는 서버리스로 데이터베이스 워크로드에 따라 자동으로 조정됩니다.

주제

RDS Proxy 개념 개요

RDS Proxy는 연결 풀링 및 다음 섹션에 설명된 다른 기능을 수행하기 위해 인프라를 처리합니다. RDS 콘솔의 프록시 페이지에서 표시된 프록시를 볼 수 있습니다.

각 프록시는 단일 Aurora DB 클러스터에 대한 연결을 처리합니다. Aurora 프로비저닝된 클러스터의 경우 프록시가 현재 라이터 인스턴스를 자동으로 결정합니다.

프록시가 열린 상태를 유지하고 데이터베이스 애플리케이션이 사용할 수 있는 연결이 연결 풀을 구성합니다.

기본적으로 RDS Proxy는 세션에서 각 트랜잭션 후에 연결을 재사용할 수 있습니다. 이 트랜잭션 수준 재사용을 멀티플렉싱이라고 합니다. RDS Proxy가 일시적으로 연결 풀에서 연결을 제거하여 재사용할 경우 해당 작업을 연결 대여라고 합니다. 그렇게 하는 것이 안전하면 RDS Proxy는 연결 풀에 해당 연결을 반환합니다.

경우에 따라 RDS Proxy는 현재 세션 외부에서 데이터베이스 연결을 재사용하는 것이 안전한지 확신할 수 없습니다. 이러한 경우 세션이 끝날 때까지 동일한 연결에서 세션을 유지합니다. 이 대체 동작을 고정이라고 합니다.

프록시에는 기본 엔드포인트가 있습니다. Amazon Aurora DB 클러스터로 작업할 때 이 엔드포인트에 연결합니다. 클러스터에 직접 연결하는 읽기/쓰기 엔드포인트에 연결하는 대신 이 작업을 수행합니다. Aurora 클러스터에 대한 특수 용도 엔드포인트는 계속 사용할 수 있습니다. Aurora DB 클러스터의 경우, 추가 읽기/쓰기 및 읽기 전용 엔드포인트를 생성할 수도 있습니다. 자세한 내용은 프록시 엔드포인트 개요 단원을 참조하십시오.

예를 들어 연결 풀링 없이 읽기/쓰기 연결을 위해 클러스터 엔드포인트에 계속 연결할 수 있습니다. 로드 밸런싱된 읽기 전용 연결을 위해 리더 엔드포인트에 계속 연결할 수 있습니다. 클러스터의 특정 DB 인스턴스에 대한 진단 및 문제 해결을 위해 인스턴스 엔드포인트에 계속 연결할 수 있습니다. AWS Lambda 같은 다른 AWS 서비스를 사용하여 RDS 데이터베이스에 연결하는 경우 프록시 엔드포인트를 사용하도록 연결 설정을 변경합니다. 예를 들어 RDS Proxy 기능을 활용하면서 Lambda 함수가 데이터베이스에 액세스할 수 있도록 프록시 엔드포인트을 지정합니다.

각 프록시는 대상 그룹을 포함합니다. 이 대상 그룹은 프록시가 연결할 수 있는 Aurora DB 클러스터를 구현합니다. Aurora 클러스터의 경우 기본적으로 대상 그룹은 해당 클러스터의 모든 DB 인스턴스와 연결됩니다. 이렇게 하면 프록시가 클러스터에서 라이터 인스턴스로 승격되는 Aurora DB 인스턴스에 연결할 수 있습니다. 프록시와 연결된 Aurora DB 클러스터를 해당 프록시의 대상이라고 합니다. 편의를 위해 콘솔을 통해 프록시를 생성하면 RDS Proxy는 해당 대상 그룹을 생성하고 연결된 대상을 자동으로 등록합니다.

엔진 패밀리는 동일한 DB 프로토콜을 사용하는 관련 데이터베이스 엔진 집합입니다. 생성하는 각 프록시에 대해 엔진 패밀리를 선택합니다.

연결 풀링

각 프록시는 연결된 Aurora DB의 라이터와 리더 인스턴스에 대한 연결 풀링을 개별적으로 수행합니다. 연결 풀링은 연결을 열고 닫으며 많은 연결을 동시에 열린 상태로 유지하는 데 관련된 오버헤드를 줄이는 최적화 기능입니다. 이 오버헤드에는 각각의 새로운 연결을 처리하는 데 필요한 메모리가 포함됩니다. 또한 각 연결을 닫고 새 연결을 여는 데는 CPU 오버헤드가 수반됩니다. 예를 들어 Transport Layer Security/Secure Sockets Layer(TLS/SSL) 핸드셰이킹, 인증, 협상 기능 등이 포함됩니다. 연결 풀링은 애플리케이션 논리를 단순화합니다. 동시 연결 수를 최소화하기 위해 애플리케이션 코드를 작성할 필요가 없습니다.

또한 각 프록시는 연결 재사용이라고도 하는 연결 멀티플렉싱을 수행합니다. 멀티플렉싱을 사용하면 RDS 프록시가 하나의 기본 데이터베이스 연결을 사용하여 한 트랜잭션에 대한 모든 작업을 수행합니다. 그런 다음 RDS 는 다음 트랜잭션에 대해 다른 연결을 사용할 수 있습니다. 사용자는 프록시에 대해 많은 동시 연결을 열 수 있고, 프록시는 DB 인스턴스 또는 클러스터에 대해 더 적은 수의 연결을 열린 상태로 유지합니다. 이렇게 하면 데이터베이스 서버에서 연결에 대한 메모리 오버헤드가 최소화됩니다. 이 기술은 또한 “연결이 너무 많음” 오류의 가능성을 줄입니다.

RDS Proxy 보안

RDS Proxy는 TLS/SSL 및 AWS Identity and Access Management(IAM)와 같은 기존 RDS 보안 메커니즘을 사용합니다. 이러한 보안 기능에 대한 일반적인 내용은 Amazon Aurora의 보안 단원을 참조하십시오. 또한 Aurora가 인증, 권한 부여 및 기타 보안 영역에서 작동하는 방식을 숙지해야 합니다.

RDS Proxy는 클라이언트 애플리케이션과 기본 데이터베이스 간의 추가 보안 계층의 역할을 할 수 있습니다. 예를 들어 기본 DB 인스턴스가 이전 버전의 TLS를 지원하더라도 TLS 1.3을 사용하여 프록시에 연결할 수 있습니다. IAM 역할을 사용하여 프록시에 연결할 수 있습니다. 프록시가 기본 사용자 및 암호 인증 방법을 사용하여 데이터베이스에 연결하더라도 마찬가지입니다. 이 기술을 사용하면 많은 비용을 들여 DB 인스턴스 자체를 마이그레이션하지 않아도 데이터베이스 애플리케이션에 강력한 인증 요구 사항을 적용할 수 있습니다.

RDS Proxy가 사용하는 데이터베이스 자격 증명을 AWS Secrets Manager에 저장합니다. 프록시가 액세스하는 Aurora DB 클러스터의 각 데이터베이스 사용자는 Secrets Manager에 해당하는 보안 암호가 있어야 합니다. RDS Proxy의 사용자에 대한 IAM 인증을 설정할 수도 있습니다. 이렇게 하면 데이터베이스가 여전히 기본 암호 인증을 사용하는 경우에도 데이터베이스 액세스에 IAM 인증을 적용할 수 있습니다. 애플리케이션 코드에 데이터베이스 자격 증명을 포함하는 대신 이러한 보안 기능을 사용하는 것이 좋습니다.

RDS Proxy에서 TLS/SSL 사용

TLS/SSL 프로토콜을 사용하여 RDS Proxy에 연결할 수 있습니다.

> **Note**
>
> RDS Proxy는 AWS Certificate Manager(ACM)의 인증서를 사용합니다. RDS Proxy을 사용하는 경우 Amazon RDS 인증서를 다운로드하거나 RDS Proxy 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다.

프록시와 데이터베이스 간의 모든 연결에 TLS를 적용하려면 AWS Management Console에서 프록시를 생성하거나 수정할 때 전송 계층 보안 필요 설정을 지정할 수 있습니다.

RDS Proxy를 사용하면 세션에서 클라이언트와 RDS Proxy 엔드포인트 간에 TLS/SSL을 사용하도록 할 수 있습니다. RDS Proxy가 그렇게 하도록 하려면 클라이언트 측에서 요구 사항을 지정합니다. SSL 세션 변수는 RDS Proxy를 사용하는 데이터베이스에 대한 SSL 연결에 대해 설정되지 않습니다.

- Aurora MySQL의 경우 `mysql` 명령을 실행할 때 `--ssl-mode` 파라미터를 사용하여 클라이언트 측에서 요구 사항을 지정합니다.
- Aurora PostgreSQL의 경우 `psql` 명령을 실행할 때 `conninfo` 문자열의 일부로 `sslmode=require`를 지정합니다.

RDS Proxy는 TLS 프로토콜 버전 1.0, 1.1, 1.2 및 1.3을 지원합니다. 기본 데이터베이스에서 사용하는 것보다 더 높은 버전의 TLS를 사용하여 프록시에 연결할 수 있습니다.

기본적으로 클라이언트 프로그램은 RDS Proxy와의 암호화된 연결을 설정하며, `--ssl-mode` 옵션을 통해 추가 제어 기능도 제공합니다. 클라이언트 측에서 RDS Proxy는 모든 SSL 모드를 지원합니다.

클라이언트의 경우, SSL 모드는 다음과 같습니다.

PREFERRED

  SSL이 최우선 선택 사항이지만 필수는 아닙니다.

비활성화됨

  SSL이 허용되지 않습니다.

필수

  SSL을 설정합니다.

VERIFY_CA

SSL을 설정하고 인증 기관(CA)을 확인합니다.

VERIFY_IDENTITY

SSL을 설정하고 CA 및 CA 호스트 이름을 확인합니다.

`--ssl-mode`, `VERIFY_CA` 또는 `VERIFY_IDENTITY`과 함께 클라이언트를 사용하는 경우 `--ssl-ca` 형식의 CA를 가리키는 `.pem` 옵션을 지정합니다. 사용할 `.pem` 파일의 경우 [Amazon Trust Services](https://www.amazontrust.com/repository/)에서 모든 루트 CA PEM을 다운로드하고 하나의 `.pem` 파일에 배치하세요.

RDS 프록시에서는 도메인과 하위 도메인 모두에 적용되는 와일드카드 인증서를 사용합니다. `mysql` 클라이언트를 사용하여 SSL 모드 `VERIFY_IDENTITY`로 연결하는 경우 현재는 MySQL 8.0 호환 `mysql` 명령을 사용해야 합니다.

장애 조치

장애 조치는 원래 인스턴스를 사용할 수 없게 될 때 데이터베이스 인스턴스를 다른 인스턴스로 대체하는 고가용성 기능입니다. 장애 조치는 데이터베이스 인스턴스 문제로 인해 발생할 수 있습니다. 또한 데이터베이스 업그레이드를 수행하는 경우와 같이 일반적인 유지 관리 절차의 일부로 발생할 수 있습니다. 장애 조치는 라이터 인스턴스 이외에 하나 이상의 리더 인스턴스가 있는 Aurora DB 클러스터에 적용됩니다.

프록시를 통해 연결하면 애플리케이션이 데이터베이스 장애 조치를 보다 효율적으로 수행할 수 있습니다. 원래 DB 인스턴스를 사용할 수 없게 되면 RDS Proxy가 유휴 애플리케이션 연결을 끊지 않고 대기 데이터베이스에 연결합니다. 이렇게 하면 장애 조치 프로세스를 단축하고 간소화할 수 있습니다. 이는 일반적인 재부팅 또는 데이터베이스 문제보다 애플리케이션에 대한 운영 중단이 발생할 확률이 적습니다.

RDS Proxy를 사용하지 않으면 장애 조치에 잠시 중단이 필요합니다. 중단 중에 장애 조치 시 데이터베이스에 대한 쓰기 작업을 수행할 수 없습니다. 모든 기존 데이터베이스 연결이 중단되며 애플리케이션이 해당 연결을 다시 열어야 합니다. 사용할 수 없는 인스턴스를 대체하여 읽기 전용 DB 인스턴스가 승격되면 새 연결 및 쓰기 작업에 데이터베이스를 사용할 수 있게 됩니다.

DB 장애 조치 중에 RDS Proxy는 계속해서 동일한 IP 주소에서 연결을 수락하고 자동으로 연결을 새 기본 DB 인스턴스로 전달합니다. RDS Proxy를 통해 연결하는 클라이언트는 다음 사항에 영향을 받지 않습니다.

- 장애 조치 시 Domain Name System(DNS) 전파가 지연됩니다.

- 로컬 DNS 캐싱.
- 연결 시간 초과.
- 현재 라이터인 DB 인스턴스가 확실하지 않습니다.
- 연결을 닫지 않은 상태에서 사용할 수 없게 된 이전 라이터의 쿼리 응답을 기다립니다.

자체 연결 풀을 유지 관리하는 애플리케이션의 경우 RDS Proxy를 통해 연결한다는 것은 장애 조치 또는 기타 중단 중에 대부분의 연결이 활성 상태를 유지함을 의미합니다. 트랜잭션 도중 또는 SQL 문의 중간에 있는 연결만 취소됩니다. RDS Proxy는 새 연결을 즉시 수락합니다. 데이터베이스 라이터를 사용할 수 없는 경우 RDS Proxy는 들어오는 요청을 대기열에 넣습니다.

자체 연결 풀을 유지 관리하지 않는 애플리케이션의 경우 RDS Proxy는 더 빠른 연결 속도와 더 많은 열린 연결을 제공합니다. 그러므로 데이터베이스에서 자주 다시 연결하기 위한 비용이 많이 드는 오버헤드가 감소합니다. 이를 위해 RDS Proxy 연결 풀에서 유지 관리되는 데이터베이스 연결을 재사용합니다. 이 접근 방식은 설정 비용이 상당한 TLS 연결에 특히 중요합니다.

트랜잭션

단일 트랜잭션 내의 모든 명령문은 항상 동일한 기본 데이터베이스 연결을 사용합니다. 트랜잭션이 종료하면 다른 세션에서 연결을 사용할 수 있게 됩니다. 트랜잭션을 세분화 단위로 사용하면 다음과 같은 결과가 발생합니다.

- Aurora MySQL `autocommit` 설정을 활성화하면 개별 문 종료 후 연결 재사용이 발생할 수 있습니다.
- 반대로, `autocommit` 설정을 비활성화하면 세션에서 실행하는 첫 번째 문이 새 트랜잭션을 시작합니다. 예를 들어,`SELECT`, `INSERT`, `UPDATE` 및 기타 데이터 조작 언어(DML) 문의 시퀀스를 입력한다고 가정합시다. 이 경우, `COMMIT`, `ROLLBACK`을 실행하거나 기타 방법으로 트랜잭션을 종료할 때까지 연결을 재사용할 수 없습니다.
- DDL(데이터 정의 언어) 문을 입력하면 해당 문이 완료된 후 트랜잭션이 종료합니다.

RDS Proxy는 데이터베이스 클라이언트 애플리케이션에서 사용하는 네트워크 프로토콜을 통해 트랜잭션이 종료하는 시점을 감지합니다. 트랜잭션 감지는 SQL 문의 텍스트에 나타나는 `COMMIT` 또는 `ROLLBACK` 같은 키워드에 의존하지 않습니다.

경우에 따라 RDS Proxy에서 세션을 다른 연결로 이동하는 것이 비현실적인 데이터베이스 요청을 감지할 수 있습니다. 이러한 경우 세션의 나머지 부분에서 해당 연결에 대한 멀티플렉싱을 해제합니다. RDS Proxy가 세션에서 멀티플렉싱이 실용적이라는 것을 확신할 수 없는 경우에도 동일한 규칙이 적용

됩니다. 이 작업을 고정이라고 합니다. 고정을 탐지하고 최소화하는 방법은 RDS 프록시 고정 방지 단원을 참조하십시오.

RDS 프록시 시작하기

다음 페이지의 정보를 통해 관련 보안 옵션을 설정하고 Amazon RDS Proxy for Aurora를 관리하고 설정하세요. 보안 옵션은 각 프록시에 액세스할 수 있는 사용자 및 각 프록시가 DB 인스턴스에 연결하는 방법을 제어합니다.

RDS 프록시를 처음 사용하는 경우 제시된 순서대로 페이지를 따라가는 것이 좋습니다.

주제

- RDS 프록시에 대한 네트워크 사전 조건 설정
- AWS Secrets Manager에서 RDS 프록시용 데이터베이스 자격 증명 설정
- RDS 프록시에 대한 AWS Identity and Access Management(IAM) 정책 설정
- Amazon Aurora용 프록시 만들기
- 프록시 보기
- RDS Proxy를 통해 데이터베이스에 연결

RDS 프록시에 대한 네트워크 사전 조건 설정

RDS 프록시를 사용하려면 Aurora DB 클러스터와 RDS 프록시 간에 공통의 Virtual Private Cloud(VPC)가 있어야 합니다. 이 VPC에는 서로 다른 가용 영역에 있는 최소 2개의 서브넷이 있어야 합니다. 계정은 이러한 서브넷을 소유하거나 다른 계정과 공유할 수 있습니다. VPC 공유에 대한 자세한 내용은 공유 VPC 작업을 참조하세요.

Amazon EC2, Lambda 또는 Amazon ECS와 같은 클라이언트 애플리케이션 리소스는 프록시와 동일한 VPC에 있을 수 있습니다. 또는 프록시와 별도의 VPC에 있을 수도 있습니다. Aurora DB 클러스터에 성공적으로 연결했다면 필요한 네트워크 리소스가 이미 있는 것입니다.

주제

- 서브넷에 대한 정보 가져오기
- IP 주소 용량 계획

서브넷에 대한 정보 가져오기

Aurora를 막 시작하는 경우 Amazon Aurora 환경 설정의 프로시저에 따라 데이터베이스에 연결하는 기본 사항을 학습할 수 있습니다. Amazon Aurora 시작하기의 자습서를 따를 수도 있습니다.

프록시를 만들려면 프록시가 작동하는 서브넷과 VPC를 제공해야 합니다. 다음 Linux 예에서는 AWS 계정에서 소유한 VPC 및 서브넷을 검사하는 AWS CLI 명령을 보여줍니다. 특히 CLI를 사용하여 프록시를 생성할 경우 서브넷 ID를 파라미터로 전달합니다.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

다음 Linux 예시에서는 특정 Aurora DB 클러스터에 해당하는 서브넷 ID를 확인하는 AWS CLI 명령을 보여줍니다.

Aurora 클러스터의 경우 먼저, 연결된 DB 인스턴스 중 하나의 ID를 찾습니다. 해당 DB 인스턴스에서 사용하는 서브넷 ID를 추출할 수 있습니다. 이렇게 하려면 DB 인스턴스에 대한 describe 출력에서 `DBSubnetGroup` 및 `Subnets` 속성 내의 중첩된 필드를 검사합니다. 해당 데이터베이스 서버의 프록시를 설정할 때 이러한 서브넷 ID의 일부 또는 전부를 지정합니다.

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]|[0]|[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

DB 인스턴스 식별자를 찾은 다음 연결된 VPC를 검사하여 서브넷을 찾습니다. 다음 Linux 예제에서는 그 방법을 보여 줍니다.

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0]|[Subnets]|[0]|[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
```

```
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

IP 주소 용량 계획

RDS 프록시는 등록된 DB 인스턴스의 크기 및 수를 기준으로 필요에 따라 용량을 자동으로 조정합니다. 등록된 데이터베이스 또는 내부 RDS 프록시 유지 관리 작업의 크기를 늘리는 것과 같은 특정 작업의 경우 프록시 용량을 늘려야 할 수 있습니다. 이러한 작업 중에 프록시에 추가 용량을 프로비저닝하기 위해 더 많은 IP 주소가 필요할 수 있습니다. 이러한 추가 주소를 사용하면 워크로드에 영향을 주지 않고 프록시를 확장할 수 있습니다. 서브넷에 사용 가능한 IP 주소가 부족하면 프록시가 스케일 업되지 않을 수 있습니다. 이로 인해 쿼리 지연 시간이 길어지거나 클라이언트 연결이 실패할 수 있습니다. RDS는 서브넷에 사용 가능한 IP 주소가 충분하지 않을 경우 `RDS-EVENT-0243` 이벤트를 통해 알려줍니다. 이 이벤트에 대한 자세한 내용은 RDS 프록시 이벤트 작업 섹션을 참조하세요.

DB 인스턴스 클래스 크기를 기준으로 프록시의 서브넷에 다음과 같은 최소 개수의 사용 가능한 IP 주소를 예약합니다.

DB 인스턴스 클래스	여유 IP 주소 최소 개수
db.*.xlarge 이하	10
db.*.2xlarge	15

DB 인스턴스 클래스	여유 IP 주소 최소 개수
db.*.4xlarge	25
db.*.8xlarge	45
db.*.12xlarge	60
db.*.16xlarge	75
db.*.24xlarge	110

이러한 권장 IP 주소 수는 기본 엔드포인트만 있는 프록시에 대한 예상 개수입니다. 추가 엔드포인트 또는 읽기 전용 복제본이 있는 프록시에는 여유 IP 주소가 더 필요할 수 있습니다. 각 추가 엔드포인트에 대해 IP 주소를 세 개 더 예약하는 것이 좋습니다. 각 읽기 전용 복제본에 대해 해당 읽기 전용 복제본의 크기를 기준으로 테이블에 지정된 대로 추가 IP 주소를 예약하는 것이 좋습니다.

> **Note**
>
> RDS Proxy는 하나의 VPC 내에서 IP 주소를 215개보다 많이 소비하지 않습니다.

예를 들어, Aurora DB 클러스터에 연결된 프록시에 필요한 IP 주소를 추정하려는 상황을 가정해 보겠습니다.

이 경우 다음과 같이 가정합니다.

- Aurora DB 클러스터에 크기가 db.r5.8xlarge인 라이터 인스턴스 1개와 크기가 db.r5.2xlarge인 리더 인스턴스 1개가 있습니다.
- 이 DB 클러스터에 연결된 프록시에는 기본 엔드포인트와 읽기 전용 역할을 가진 사용자 지정 엔드포인트 1개가 있습니다.

이 경우 프록시에는 약 63개의 여유 IP 주소(라이터 인스턴스에 45개, 리더 인스턴스에 15개, 추가 사용자 지정 엔드포인트에 3개)가 필요합니다.

AWS Secrets Manager에서 RDS 프록시용 데이터베이스 자격 증명 설정

Amazon RDS의 RDS 프록시는 AWS Secrets Manager를 사용하여 데이터베이스 자격 증명을 안전하게 저장하고 관리합니다. 애플리케이션에 임베딩 자격 증명을 포함하는 대신 프록시를 필요한 인증 세부 정보가 포함된 Secrets Manager 보안 암호와 연결합니다. 프록시가 Aurora DB 클러스터에서 연결하는 각 데이터베이스 사용자 계정에 대해 별도의 Secrets Manager 암호를 만듭니다.

주제

- [RDS 프록시에 사용할 보안 암호 만들기](#)

RDS 프록시에 사용할 보안 암호 만들기

프록시를 만들기 전에 데이터베이스 자격 증명을 저장하는 보안 암호를 최소 하나 이상 만들어야 합니다.

콘솔

보안 암호 생성

1. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
2. Store a new secret(새 보안 암호 저장)을 선택합니다.
3. Amazon RDS 데이터베이스에 대한 자격 증명을 선택합니다.
4. 사용자 이름과 암호를 입력합니다. 입력한 자격 증명은 연결된 RDS 데이터베이스에 있는 데이터베이스 사용자의 자격 증명과 일치해야 합니다. RDS 프록시는 이러한 자격 증명을 사용하여 애플리케이션 대신 데이터베이스에 대한 연결을 인증하고 설정합니다.

   일치하지 않는 경우 데이터베이스 암호와 일치하도록 보안 암호를 업데이트할 수 있습니다. 보안 암호를 업데이트할 때까지는 해당 보안 암호를 사용하여 프록시를 통해 연결을 시도하는 작업이 실패할 것이지만, 다른 유효한 보안 암호를 사용하는 연결은 여전히 작동합니다.

   > **Note**
   >
   > RDS for SQL Server의 경우 RDS 프록시에는 DB 인스턴스 데이터 정렬 설정에 관계없이 Secrets Manager에 대/소문자를 구분하는 보안 암호가 필요합니다. 애플리케이션에서 관리자('Admin' 및 'admin')와 같이 대문자가 다른 사용자 이름을 허용하는 경우 각각에 대해 별도의 보안 암호를 만들어야 합니다. RDS 프록시는 클라이언트와 프록시 간에 대/소문자를 구분하지 않는 사용자 이름 인증을 지원하지 않습니다.

SQL Server의 콜레이션에 대한 자세한 내용은 Microsoft SQL Server 설명서를 참조하세요.

5. 데이터베이스에서 보안 암호가 액세스할 Amazon RDS 데이터베이스를 선택합니다.
6. 보안 암호의 다른 설정을 입력한 다음 저장을 선택합니다. 포괄적인 설명은 AWS Secrets Manager 사용 설명서에 나와 있는 Creating an AWS Secrets Manager secret를 참조하시기 바랍니다.

AWS CLI

AWS CLI를 통해 프록시를 만들 때 해당 보안 암호의 Amazon 리소스 이름(ARN)을 지정합니다. 프록시가 액세스할 수 있는 모든 DB 사용자 계정에 지정합니다. AWS Management Console에서는 설명하는 이름으로 비밀을 선택합니다.

- RDS 프록시와 함께 사용할 Secrets Manager 보안 암호를 만들려면 create-secret 명령을 사용합니다.

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- 또한 사용자 지정 키를 생성하여 Secrets Manager 보안 암호를 암호화할 수 있습니다. 다음 명령은 예시 키를 생성합니다.

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version":"2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
```

```
          "Principal":
            {
              "AWS":
                ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
            },
          "Action":
            [
              "kms:Create*",
              "kms:Describe*",
              "kms:Enable*",
              "kms:List*",
              "kms:Put*",
              "kms:Update*",
              "kms:Revoke*",
              "kms:Disable*",
              "kms:Get*",
              "kms:Delete*",
              "kms:TagResource",
              "kms:UntagResource",
              "kms:ScheduleKeyDeletion",
              "kms:CancelKeyDeletion"
            ],
          "Resource":"*"
        },
        {
          "Sid":"Allow use of the key",
          "Effect":"Allow",
          "Principal":{"AWS":"$ROLE_ARN"},
          "Action":["kms:Decrypt","kms:DescribeKey"],
          "Resource":"*"
        }
      ]
}'
```

예를 들어, 다음 명령은 두 데이터베이스 사용자에 대한 Secrets Manager 보안 암호를 생성합니다.

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
```

```
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

사용자 지정 AWS KMS 키로 암호화된 이러한 보안 암호를 만들려면 다음 명령을 사용하세요.

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

AWS 계정에서 소유한 보안 암호를 보려면 list-secrets 명령을 사용합니다.

```
aws secretsmanager list-secrets
```

CLI를 사용하여 프록시를 생성할 경우 하나 이상의 보안 정보에 대한 Amazon 리소스 이름(ARN)을 `--auth` 파라미터에 전달합니다. 다음 예제에서는 AWS 계정이 소유한 각 보안 암호의 이름과 ARN만 사용하여 보고서를 준비하는 방법을 보여 줍니다. 이 예에서는 `--output table` 버전 2에서 제공되는 AWS CLI 파라미터를 사용합니다. AWS CLI 버전 1을 사용하는 경우 `--output text`를 대신 사용합니다.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

보안 암호에 올바른 자격 증명이 올바른 형식으로 포함되어 있는지 확인하려면 get-secret-value 명령을 사용합니다. *`your_secret_name`*을 보안 암호의 간단한 이름 또는 ARN으로 바꿉니다.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

출력에는 다음과 유사한 JSON 인코딩 값이 있는 줄이 포함됩니다.

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

RDS 프록시에 대한 AWS Identity and Access Management(IAM) 정책 설정

Secrets Manager에서 비밀을 만든 후 해당 비밀에 액세스할 수 있는 IAM 정책을 생성합니다. IAM 사용에 대한 일반적인 정보는 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.

> **Tip**
>
> IAM 콘솔을 사용하는 경우 다음 절차가 적용됩니다. AWS Management Console for RDS를 사용하는 경우 RDS에서 자동으로 IAM 정책을 생성할 수 있습니다. 이 경우 다음 절차를 건너뛸 수 있습니다.

프록시와 함께 사용할 Secrets Manager 비밀에 액세스하는 IAM 정책을 생성하려면

1. IAM 콘솔에 로그인합니다. IAM 역할 생성에 설명된 대로 역할 생성 프로세스를 따르고, AWS 서비스에 대한 권한을 위임할 역할 생성을 선택합니다.

   신뢰할 수 있는 엔터티 유형에서 AWS 서비스를 선택합니다. 사용 사례 아래의 기타 AWS 서비스 사용 사례 드롭다운에서 RDS를 선택합니다. RDS – 데이터베이스에 역할 추가를 선택합니다.
2. 새 역할의 경우 인라인 정책 추가 단계를 수행합니다. IAM 정책 편집과 동일한 일반 절차를 사용합니다. 다음 JSON을 JSON 텍스트 상자에 붙여 넣습니다. 자신의 계정 ID를 대체합니다. AWS에 대한 `us-east-2` 리전을 대체합니다. 생성한 보안 암호에 대한 Amazon 리소스 이름(ARN)을 대체합니다. IAM 정책 설명에서 KMS 키 지정을 확인하세요. `kms:Decrypt` 작업을 수행하려면 기본 AWS KMS key 또는 자체 KMS 키를 대체하세요. 무엇을 사용할지는 Secrets Manager 암호를 암호화하는 데 사용할 때 무엇을 사용했는지에 따라 달라집니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
```

```
                "Effect": "Allow",
                "Action": "kms:Decrypt",
                "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
                "Condition": {
                    "StringEquals": {
                        "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                    }
                }
            }
        ]
    }
```

3. 이 IAM 역할에 대한 신뢰 정책을 편집합니다. 다음 JSON을 JSON 텍스트 상자에 붙여 넣습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

다음 명령은 AWS CLI를 통해 동일한 작업을 수행합니다.

```
PREFIX=my_identifier
USER_ARN=$(aws sts get-caller-identity --query "Arn" --output text)

aws iam create-role --role-name my_role_name \
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'

ROLE_ARN=arn:aws:iam::account_id:role/my_role_name

aws iam put-role-policy --role-name my_role_name \
  --policy-name $PREFIX-secret-reader-policy --policy-document '{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

Amazon Aurora용 프록시 만들기

Amazon RDS Proxy를 사용하면 연결을 풀링하고 데이터베이스 장애 조치를 더욱 효율적으로 관리하여 데이터베이스 애플리케이션의 확장성, 가용성 및 보안을 개선할 수 있습니다. 이 주제에서는 프록시를 만드는 절차를 살펴봅니다. 시작하기 전에 데이터베이스가 IAM 권한 및 VPC 구성을 포함하여 필요한 사전 조건을 충족하는지 확인합니다.

프록시를 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터와 연결할 수 있습니다.

콘솔

프록시를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. Create proxy(프록시 생성)를 선택합니다.

4. 프록시에 대한 다음 설정을 구성합니다.

설정	설명
엔진 패밀리.	프록시가 데이터베이스와 주고받는 네트워크 트래픽을 해석할 때 인식하는 데이터베이스 네트워크 프로토콜입니다. **Note** Aurora PostgreSQL을 사용하려면 인스턴스에서 `postgres` 데이터베이스를 유지해야 합니다. [삭제된 `postgres` 데이터베이스 문제 해결]을(를) 참조하세요.
프록시 식별자	AWS 계정 ID와 현재 AWS 리전 내에서 고유한 이름입니다.
유휴 클라이언트 연결 시간 초과	프록시는 설정된 기간 동안 유휴 상태로 유지되면 클라이언트 연결을 닫습니다. 기본값은 1,800초(30분)입니다. 애플리케이션이 이전 요청을 완료한 후 지정된 시간 내에 새 요청을 제출하지 않으면 연결은 유휴 상태입니다. 프록시는 기본 데이터베이스 연결을 열어 두고 연결 풀로 반환하여 새 클라이언트 연결에 사용할 수 있도록 합니다. 기간 경과 연결을 사전에 제거하려면 유휴 클라이언트 연결 제한 시간을 줄이세요. 워크로드가 급증하는 동안 연결 비용을 최소화하려면 제한 시간을 늘립니다.
데이터베이스	이 프록시를 통해 액세스할 Aurora DB 클러스터입니다. 이 목록에는 호환되는 데이터베이스 엔진, 엔진 버전 및 기타 설정이 있는 DB 인스턴스 또는 클러스터만 포함됩니다. 목록이 비어 있으면 RDS Proxy와 호환되는 새 DB 인스턴스 또는 클러스터를 생성합니다. 이 작업을 수행하려면 [Amazon Aurora DB 클러스터 생성]의 프로시저를 따르세요. 그런 다음 프록시를 다시 생성해 보세요.

설정	설명
연결 풀 최대 연결	RDS Proxy가 사용할 수 있는 max_connections 한도의 백분율을 정의하기 위한 1~100 사이의 값입니다. 이 DB 인스턴스 또는 클러스터에 하나의 프록시만 사용하려는 경우 이 값을 100으로 설정합니다. RDS Proxy가 이 설정을 사용하는 방법에 대한 자세한 내용은 the section called “MaxConnectionsPercent” 섹션을 참조하세요.
세션 고정 필터	RDS 프록시가 멀티플렉싱 연결에 대한 기본 안전 조치를 우회하는 감지된 특정 세션 상태를 고정하지 못하도록 합니다. 현재 PostgreSQL은 이 설정을 지원하지 않으며 사용 가능한 유일한 옵션은 EXCLUDE_VARIABLE_SETS 입니다. 이를 활성화하면 한 연결의 세션 변수가 다른 연결의 세션 변수에 영향을 줄 수 있으며, 쿼리가 현재 트랜잭션 외부에 설정된 세션 변수에 의존하는 경우 오류 또는 정확성 문제가 발생할 수 있습니다. 애플리케이션이 데이터베이스 연결을 안전하게 공유할 수 있는지 확인한 후에만 이 옵션을 사용합니다. 다음과 같은 패턴이 나타나면 안전한 상태로 간주될 수 있습니다. • 유효 세션 변수 값에 변경 사항이 없는 SET 문입니다. 즉, 세션 변수에 변경 사항이 없습니다. • 세션 변수 값을 변경하고 동일한 트랜잭션에서 명령문을 실행합니다. 자세한 내용은 RDS 프록시 고정 방지 섹션을 참조하세요.
연결 대여 시간 초과	프록시가 사용 가능한 모든 데이터베이스 연결을 사용할 것으로 예상되는 경우 제한 시간 오류가 반환되기 전에 대기 시간을 설정합니다. 최대 5분을 지정할 수 있습니다. 이 설정은 프록시가 최대 연결 수에 도달했으며 모두 사용 중인 경우에만 적용됩니다.

<table>
<tr><th>설정</th><th>설명</th></tr>
<tr><td>초기화 쿼리.</td><td>(선택 사항) 초기화 쿼리를 추가하거나 현재 쿼리를 수정합니다. 각 새 데이터베이스 접속을 열 때 실행할 프록시에 대한 하나 이상의 SQL 문을 지정할 수 있습니다. 이 설정은 일반적으로 각 접속에 동일한 설정이 있는지 확인하기 위해 SET 문과 함께 사용됩니다. 추가하는 쿼리가 유효한지 확인합니다. 단일 SET 문에 여러 변수를 포함하려면 쉼표 구분자를 사용합니다. 예시:
<pre><code>SET variable1 =value1, variable2 =value2</code></pre>여러 문의 경우 세미콜론을 구분 기호로 사용합니다.

Important
대상 그룹 구성의 일부로 초기화 쿼리에 액세스할 수 있으므로 이는 인증 또는 암호화 방법으로 보호되지 않습니다. 프록시 대상 그룹 구성을 보거나 관리할 수 있는 액세스 권한이 있는 사람은 누구나 초기화 쿼리를 볼 수 있습니다. 암호 또는 수명이 긴 암호화 키와 같은 민감한 데이터를 이 옵션에 추가해서는 안 됩니다.</td></tr>
<tr><td>AWS Identity and Access Management(IAM) 역할</td><td>프록시가 사용할 수 있는 데이터베이스 사용자 계정의 자격 증명을 나타내는 Secrets Manager 보안 암호에 액세스할 수 있는 권한이 있는 IAM 역할입니다. 또는 AWS Management Console에서 IAM 역할을 만들 수 있습니다.</td></tr>
<tr><td>Secrets Manager 보안 암호</td><td>프록시가 Aurora DB 클러스터에 액세스할 수 있는 데이터베이스 사용자 보안 인증 정보를 포함하는 Secrets Manager 보안 암호를 하나 이상 선택합니다.</td></tr>
<tr><td>클라이언트 인증 유형</td><td>프록시가 클라이언트로부터의 연결에 사용하는 인증 유형입니다. 선택 사항은 이 프록시와 연결된 모든 Secrets Manager 비밀에 적용됩니다. 비밀마다 다른 클라이언트 인증 유형을 지정해야 하는 경우 AWS CLI 또는 API를 대신 사용하여 프록시를 생성합니다.</td></tr>
</table>

설정	설명
IAM 인증.	프록시 연결에 대해 IAM 인증을 요구할지 아니면 허용하지 않을지를 나타냅니다. 선택 사항은 이 프록시와 연결된 모든 Secrets Manager 비밀에 적용됩니다. 암호마다 다른 IAM 인증을 지정해야 하는 경우 AWS CLI 또는 API를 대신 사용하여 프록시를 생성합니다.
전송 계층 보안 필요	모든 클라이언트 연결에 TLS/SSL을 적용합니다. 클라이언트 연결의 암호화 여부와는 상관없이 프록시는 기본 데이터베이스에 연결할 때 동일한 암호화 설정을 사용합니다.
서브넷	이 필드는 VPC와 연결된 모든 서브넷으로 미리 채워집니다. 프록시에 필요하지 않은 서브넷은 제거할 수 있지만 서브넷은 두 개 이상 남겨 두어야 합니다.

설정	설명
VPC 보안 그룹	기존 VPC 보안 그룹을 선택하거나 AWS Management Console 에서 새 보안 그룹을 만듭니다. 애플리케이션이 프록시에 액세스할 수 있도록 인바운드 규칙을 구성하고 데이터베이스 대상의 트래픽을 허용하도록 아웃바운드 규칙을 구성합니다. Note 보안 그룹은 프록시에서 데이터베이스로의 연결을 허용해야 합니다. 보안 그룹은 애플리케이션에서 프록시로 수신하고 프록시에서 데이터베이스로 발신하는 데 모두 사용됩니다. 예를 들어 데이터베이스와 프록시 모두에 동일한 보안 그룹을 사용하는 경우, 해당 보안 그룹 내의 리소스가 서로 통신할 수 있는지 확인합니다. 공유 VPC를 사용하는 경우 VPC에 대한 기본 보안 그룹이나 다른 계정과 연결된 보안 그룹을 사용하지 마세요. 대신 본인 계정에 속한 보안 그룹을 선택합니다. VPC가 없다면 만드세요. 자세한 내용은 공유 VPC 작업을 참조하세요. RDS는 고가용성을 보장하기 위해 여러 가용 영역에 프록시를 배포합니다. AZ 간 통신을 활성화하려면 프록시 서브넷의 네트워크 액세스 제어 목록(ACL)에서 엔진 포트의 발신 및 모든 포트의 수신을 허용해야 합니다. 네트워크 ACL에 대한 자세한 내용은 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요. 프록시와 대상의 네트워크 ACL이 동일한 경우 소스가 VPC CIDR로 설정된 TCP 프로토콜 수신 규칙을 추가해야 합니다. 또한 대상이 VPC CIDR로 설정된 엔진 포트별 TCP 프로토콜 송신 규칙을 추가해야 합니다.

설정	설명
향상된 로깅 활성화	프록시 호환성 또는 성능 문제를 해결하려면 이 설정을 활성화합니다. 활성화된 경우 RDS 프록시는 SQL 동작 또는 프록시 연결 성능 및 확장성을 디버깅하는 데 도움이 되는 자세한 성능 정보를 로깅합니다. 디버깅에 대해서만 이 설정을 활성화하고 로그의 민감한 정보를 보호하기 위한 적절한 보안 조치가 마련되어 있는지 확인합니다. 오버헤드를 최소화하기 위해 RDS 프록시는 활성화 후 24시간이 지나면 이 설정을 자동으로 비활성화합니다. 특정 문제를 해결하기 위해 일시적으로 사용합니다.

5. Create proxy(프록시 생성)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 프록시를 생성하려면 다음 필수 파라미터와 함께 create-db-proxy 명령을 호출합니다.

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--auth`
- `--vpc-subnet-ids`

`--engine-family` 값은 대소문자를 구분합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --auth ProxyAuthenticationConfig_JSON_string \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
```

```
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--tags comma_separated_list]
```

Windows의 경우:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --auth ProxyAuthenticationConfig_JSON_string ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--tags comma_separated_list]
```

다음은 `--auth` 옵션의 JSON 값 예시입니다. 이 예시는 각 보안 암호에 서로 다른 클라이언트 인증 유형을 적용합니다.

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },
```

```
  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

> **Tip**
>
> `--vpc-subnet-ids` 파라미터에 사용할 서브넷 ID를 아직 모르는 경우, [RDS 프록시에 대한 네트워크 사전 조건 설정]에서 서브넷 ID를 찾는 방법의 예를 참조하세요.

> **Note**
>
> 보안 그룹은 프록시가 연결하는 데이터베이스에 대한 액세스를 허용해야 합니다. 동일한 보안 그룹이 애플리케이션에서 프록시로 수신하고 프록시에서 데이터베이스로 발신하는 데 사용됩니다. 예를 들어 데이터베이스와 프록시에 대해 동일한 보안 그룹을 사용한다고 가정합니다. 이 경우 해당 보안 그룹의 리소스가 동일한 보안 그룹의 다른 리소스와 통신할 수 있도록 지정해야 합니다.
> 공유 VPC를 사용하는 경우 VPC에 대한 기본 보안 그룹이나 다른 계정에 속한 보안 그룹을 사용할 수 없습니다. 본인 계정에 속한 보안 그룹을 선택합니다. 없으면 새로 생성합니다. 이 제한 사항에 대한 자세한 내용은 [공유 VPC 작업]을 참조하세요.

프록시에 적합한 연결을 생성하려면 [register-db-proxy-targets] 명령을 사용합니다. 대상 그룹 이름 `default`을 지정합니다. RDS Proxy는 각 프록시를 생성할 때 이 이름으로 대상 그룹을 자동으로 생성합니다.

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS API

RDS 프록시를 생성하려면 Amazon RDS API 작업 CreateDBProxy를 호출합니다. AuthConfig 데이터 구조와 함께 파라미터를 전달합니다.

RDS Proxy는 각 프록시를 생성할 때 `default`라는 대상 그룹을 자동으로 생성합니다. RegisterDBProxyTargets 함수를 호출하여 Aurora DB 클러스터를 대상 그룹에 연결합니다.

프록시 보기

하나 이상의 RDS 프록시를 생성한 후 AWS Management Console, AWS CLI 또는 RDS API에서 해당 프록시를 보고 관리할 수 있습니다. 구성 세부 정보를 검토하고, 성능을 모니터링하고, 필요에 따라 수정하거나 삭제할 프록시를 결정할 수 있습니다.

데이터베이스 애플리케이션이 프록시를 통해 트래픽을 라우팅하도록 하려면 연결 문자열에 프록시 엔드포인트를 지정해야 합니다.

콘솔

콘솔에서 프록시를 확인하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 세부 정보를 볼 프록시 이름을 선택합니다.
4. 세부 정보 페이지의 대상 그룹 섹션에 프록시가 특정 Aurora DB 클러스터에 어떻게 연결되는지 표시됩니다. 기본 대상 그룹 페이지로 이동하여 프록시를 만드는 중에 정의된 구성 설정을 포함하여 이 연결을 더 자세히 볼 수 있습니다. 이러한 설정에는 최대 연결 비율, 연결 대여 시간 초과, 엔진 패밀리, 세션 고정 필터 등이 포함됩니다.

CLI

CLI를 사용하여 프록시를 보려면 describe-db-proxy 명령을 사용합니다. 기본적으로 요청은 AWS 계정이 소유한 모든 프록시를 반환합니다. 단일 프록시에 대한 세부 정보를 보려면 `--db-proxy-name` 파라미터와 함께 해당 이름을 지정합니다.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

프록시와 연결된 다른 정보를 보려면 다음 명령을 사용합니다.

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

프록시와 연결된 항목에 대한 자세한 내용을 보려면 다음 명령 시퀀스를 사용합니다.

1. 프록시 목록을 얻으려면 describe-db-proxies를 실행합니다.
2. 프록시에서 사용할 수 있는 최대 연결 비율과 같은 연결 파라미터를 표시하려면 describe-db-proxy-target-groups `--db-proxy-name`을 실행합니다. 프록시의 이름을 파라미터 값으로 사용합니다.
3. 반환된 대상 그룹과 연결된 Aurora DB 클러스터의 세부 정보를 보려면 describe-db-proxy-targets를 실행합니다.

RDS API

RDS API를 사용하여 프록시를 보려면 DescribeDBProxies 작업을 사용합니다. 이 작업은 DBProxy 데이터 형식의 값을 반환합니다.

프록시 연결 설정의 세부 정보를 보려면 DescribeDBProxyTargetGroups 작업을 사용하여 이 반환 값의 프록시 식별자를 사용합니다. 이 작업은 DBProxyTargetGroup 데이터 형식의 값을 반환합니다.

프록시와 연결된 RDS 인스턴스 또는 Aurora DB 클러스터를 보려면 DescribeDBProxyTargets 작업을 사용합니다. 이 작업은 DBProxyTarget 데이터 형식의 값을 반환합니다.

RDS Proxy를 통해 데이터베이스에 연결

일반적으로 데이터베이스에 직접 연결하는 것과 동일한 방식으로 프록시를 통해 Aurora DB 또는 Aurora Serverless v2를 사용하는 클러스터에 연결합니다. 가장 큰 차이점은 클러스터 엔드포인트 대신 프록시 엔드포인트를 지정한다는 것입니다. 기본적으로 모든 프록시 연결은 읽기/쓰기 기능이 있으며 라이터 인스턴스를 사용합니다. 읽기 전용 연결에 리더 엔드포인트를 일반적으로 사용하는 경우, 프록시를 위한 추가 읽기 전용 엔드포인트를 생성할 수 있습니다. 이 엔드포인트도 같은 방식으로 사용할 수 있습니다. 자세한 내용은 프록시 엔드포인트 개요 섹션을 참조하세요.

주제

- 기본 인증을 사용하여 프록시에 연결
- IAM 인증을 사용하여 프록시에 연결
- PostgreSQL을 사용하여 프록시에 연결할 때 고려할 사항

기본 인증을 사용하여 프록시에 연결

기본 인증을 사용하여 프록시에 연결하려면 다음 단계를 사용합니다.

1. 프록시 엔드포인트를 찾습니다. AWS Management Console에서는 해당 프록시의 세부 정보 페이지에서 엔드포인트를 찾을 수 있습니다. AWS CLI에서는 describe-db-proxies 명령을 사용할 수 있습니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다.

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. 클라이언트 애플리케이션의 연결 문자열에서 해당 엔드포인트를 호스트 파라미터로 지정합니다. 예를 들어 프록시 엔드포인트를 `mysql -h` 옵션 또는 `psql -h` 옵션의 값으로 지정합니다.

3. 평소와 동일한 데이터베이스 사용자 이름과 암호를 제공합니다.

IAM 인증을 사용하여 프록시에 연결

RDS Proxy에서 IAM 인증을 사용하는 경우 일반 사용자 이름 및 암호로 인증하도록 데이터베이스 사용자를 설정합니다. IAM 인증은 Secrets Manager에서 사용자 이름 및 암호 자격 증명을 검색하는 RDS Proxy에 적용됩니다. RDS 프록시에서 기본 데이터베이스로의 연결은 IAM을 거치지 않습니다.

IAM 인증을 사용하여 RDS 프록시에 연결하려면 IAM 인증을 통해 Aurora DB 클러스터에 연결하는 것과 동일한 일반 연결 프로시저를 따르면 됩니다. IAM 사용에 대한 일반적인 정보는 Amazon Aurora의 보안 섹션을 참조하세요.

RDS Proxy에 대한 IAM 사용의 주요 차이점은 다음과 같습니다.

- 권한 부여 플러그인으로 각 개별 데이터베이스 사용자를 구성하지 않습니다. 데이터베이스 사용자는 여전히 데이터베이스 내에서 일반 사용자 이름과 암호를 가지고 있습니다. 이러한 사용자 이름과 암호를 포함하는 Secrets Manager 비밀을 설정하고 RDS Proxy가 Secrets Manager에서 자격 증명을 검색할 수 있는 권한을 부여합니다.

  IAM 인증은 클라이언트 프로그램과 프록시 간의 연결에 적용됩니다. 그런 다음 프록시는 Secrets Manager에서 검색된 사용자 이름 및 암호 자격 증명을 사용하여 데이터베이스에 대해 인증합니다.
- 인스턴스, 클러스터 또는 리더 엔드포인트 대신 프록시 엔드포인트를 지정합니다. 프록시 엔드포인트에 대한 자세한 내용은 IAM 인증을 사용하여 DB 클러스터에 연결 단원을 참조하십시오.
- 직접 데이이터베이스 IAM 인증의 경우 데이터베이스 사용자를 선택적으로 선택하고 특수 인증 플러그인으로 식별되도록 구성합니다. 그런 다음 IAM 인증을 사용하여 해당 사용자에게 연결할 수 있습니다.

  프록시 사용 사례에서는 일부 사용자의 사용자 이름과 암호(기본 인증)가 포함된 암호를 프록시에 제공해야 합니다. 그런 다음 IAM 인증을 사용하여 프록시에 연결합니다. 여기서는 데이터베이스 엔드포인트가 아닌 프록시 엔드포인트로 인증 토큰을 생성하여 이를 수행합니다. 또한 제공한 암호의 사용자 이름 중 하나와 일치하는 사용자 이름을 사용합니다.
- IAM 인증을 사용하여 프록시에 연결할 때는 전송 계층 보안(TLS)/보안 소켓 계층(SSL)을 사용해야 합니다.

IAM 정책을 수정하여 특정 사용자에게 프록시에 대한 액세스 권한을 부여할 수 있습니다. 예를 들면 다음과 같습니다.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

PostgreSQL을 사용하여 프록시에 연결할 때 고려할 사항

RDS 프록시에 연결할 새 PostgreSQL 데이터베이스 사용자를 생성하는 경우 데이터베이스에 대해 사용자에게 CONNECT 권한을 부여해야 합니다. 이렇게 하지 않으면 사용자가 연결을 설정할 수 없습니다. 자세한 내용은 the section called “RDS 프록시 사용 시 PostgreSQL 데이터베이스에 새 데이터베이스 사용자 추가” 섹션을 참조하세요.

클라이언트가 PostgreSQL 데이터베이스에 대한 연결을 시작하면 시작 메시지가 전송됩니다. 이 메시지에는 파라미터 이름과 값 문자열의 쌍이 포함됩니다. 자세한 내용은 PostgreSQL 설명서에서 PostgreSQL 메시지 형식의 `StartupMessage`를 참조하십시오.

RDS 프록시를 통해 연결할 때 시작 메시지에는 현재 인식되는 다음과 같은 파라미터가 포함될 수 있습니다.

- `user`
- `database`

시작 메시지에는 다음과 같은 추가 런타임 파라미터가 포함될 수도 있습니다.

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`
- `extra_float_digits`
- `search_path`

PostgreSQL 메시징에 대한 자세한 내용은 PostgreSQL 설명서의 프런트 엔드/백엔드 프로토콜을 참조하십시오.

PostgreSQL의 경우 JDBC를 사용한다면 고정을 피하기 위해 다음을 사용하는 것이 좋습니다.

- 고정을 방지하려면 JDBC 연결 파라미터 `assumeMinServerVersion`을 `9.0` 이상으로 설정합니다. 이렇게 하면 `SET extra_float_digits = 3`을 실행할 때 JDBC 드라이버가 연결 시작 중에 추가 왕복을 수행하지 못합니다.
- 고정을 방지하려면 JDBC 연결 파라미터 `ApplicationName`을 *`any/your-application-name`*으로 설정합니다. 이렇게 하면 `SET application_name = "PostgreSQL JDBC`

Driver"을 실행할 때 JDBC 드라이버가 연결 시작 중에 추가 왕복을 수행하지 못합니다. JDBC 파라미터는 `ApplicationName`이지만 PostgreSQL `StartupMessage` 파라미터는 `application_name`입니다.

자세한 내용은 RDS 프록시 고정 방지 섹션을 참조하세요. JDBC를 사용한 연결에 대한 자세한 내용은 PostgreSQL 설명서의 데이터베이스에 연결을 참조하십시오.

RDS 프록시 관리

이 섹션에서는 RDS 프록시 작업 및 구성을 관리하는 방법에 대한 정보를 제공합니다. 이러한 절차는 애플리케이션이 데이터베이스 연결을 가장 효율적으로 사용하고 최대 연결 재사용을 달성하는 데 도움이 됩니다. 연결 재사용을 더 많이 활용할수록 CPU 및 메모리 오버헤드를 더 많이 절약할 수 있습니다. 이렇게 하면 애플리케이션의 대기 시간이 줄어들고 데이터베이스가 애플리케이션 요청을 처리하는 데 더 많은 리소스를 사용할 수 있습니다.

주제

- RDS 프록시 수정
- RDS 프록시 사용 시 새 데이터베이스 사용자 추가
- RDS 프록시 연결 고려 사항
- RDS 프록시 고정 방지
- RDS 프록시 삭제

RDS 프록시 수정

프록시를 생성한 후 프록시와 연결된 특정 설정을 변경할 수 있습니다. 프록시 자체, 연결된 대상 그룹 또는 둘 다 수정하면 됩니다. 각 프록시에는 연결된 대상 그룹이 있습니다.

AWS Management Console

> **Important**
>
> 클라이언트 인증 유형 및 IAM 인증 필드의 값은 이 프록시에 연결된 모든 Secrets Manager 보안 암호에 적용됩니다. 보안 암호마다 다른 값을 지정하려면 AWS CLI 또는 API를 대신 사용하여 프록시를 수정하세요.

프록시에 대한 설정을 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 프록시 목록에서 설정을 수정하려는 프록시를 선택하거나 세부 정보 페이지로 이동합니다.
4. 작업에서 수정을 선택합니다.
5. 수정할 속성을 입력하거나 선택합니다. 다음을 수정할 수 있습니다.
   - 프록시 식별자 – 새 식별자를 입력하여 프록시 이름을 변경합니다.
   - 유휴 클라이언트 연결 시간 초과 - 유휴 클라이언트 연결 시간 초과의 기간을 입력합니다.
   - IAM 역할 – Secrets Manager에서 보안 암호를 검색하는 데 사용되는 IAM 역할을 변경합니다.
   - Secrets Manager 보안 암호 - Secrets Manager 보안 암호를 추가하거나 제거합니다. 이러한 비밀은 데이터베이스 사용자 이름 및 암호에 해당합니다.
   - 클라이언트 인증 유형 - (PostgreSQL만 해당) 프록시에 대한 클라이언트 연결의 인증 유형을 변경합니다.
   - IAM 인증 - 프록시 연결에 대해 IAM 인증을 요구하거나 허용하지 않습니다.
   - 전송 계층 보안 필요 - 전송 계층 보안(TLS) 요구 사항을 설정하거나 해제합니다.
   - VPC 보안 그룹 - 프록시가 사용할 VPC 보안 그룹을 추가하거나 제거합니다.
   - 향상된 로깅 활성화 - 고급 로깅을 사용하거나 사용 중지하도록 설정합니다.
6. 수정을 선택합니다.

변경할 설정을 찾지 못한 경우 다음 절차에 따라 프록시의 대상 그룹을 업데이트합니다. 프록시와 연결된 대상 그룹은 물리적 데이터베이스 연결과 관련된 설정을 제어합니다. 각 프록시에는 `default`라는 하나의 연결된 대상 그룹이 있으며, 이 그룹은 프록시와 함께 자동으로 생성됩니다. 기본 대상 그룹의 이름은 바꿀 수 없습니다.

대상 그룹은 프록시 세부 정보 페이지에서만 수정할 수 있으며 Proxies(프록시) 페이지의 목록에서는 수정할 수 없습니다.

프록시 대상 그룹에 대한 설정을 수정하려면

1. [프록시(Proxies)] 페이지에서 프록시의 세부 정보 페이지로 이동합니다.
2. 대상 그룹에서 `default` 링크를 선택합니다. 현재 모든 프록시에는 `default`라는 단일 대상 그룹이 있습니다.

3. 기본 대상 그룹에 대한 세부 정보 페이지에서 수정을 선택합니다.
4. 수정할 수 있는 속성의 새 설정을 선택합니다.

   - 데이터베이스 - 다른 Aurora 클러스터를 선택합니다.
   - 연결 풀 최대 연결 – 프록시에서 사용할 수 있는 최대 연결 비율을 조정합니다.
   - 세션 고정 필터 - (선택 사항) 세션 고정 필터를 선택합니다. 이렇게 하면 클라이언트 연결 간 데이터베이스 연결을 멀티플렉싱하기 위한 기본 안전 조치를 우회할 수 있습니다. 현재 PostgreSQL에서는 설정이 지원되지 않습니다. `EXCLUDE_VARIABLE_SETS`만 선택할 수 있습니다.

     이 설정을 활성화하면 어떤 연결의 세션 변수가 다른 연결에 영향을 줄 수 있습니다. 쿼리가 현재 트랜잭션 외부에 설정된 세션 변수 값에 의존하는 경우 이로 인해 오류나 정확성 문제가 발생할 수 있습니다. 애플리케이션이 클라이언트 연결 간에 데이터베이스 연결을 공유해도 안전한지 확인한 후 이 옵션을 사용하는 것이 좋습니다.

     다음과 같은 패턴이 나타나면 안전한 상태로 간주될 수 있습니다.

     - 유효 세션 변수 값에 변경 사항이 없는 `SET` 명령문이 있습니다(즉 세션 변수에 변경 사항이 없는 경우).
     - 세션 변수 값을 변경하고 동일한 트랜잭션에서 명령문을 실행합니다.

     자세한 내용은 RDS 프록시 고정 방지 단원을 참조하십시오.
   - 연결 대여 시간 초과 - 연결 대여 시간 초과 간격을 조정합니다. 이 설정은 최대 연결 수가 프록시에 이미 사용되고 있는 경우에 적용됩니다. 이 설정은 시간 초과 오류를 반환하기 전에 프록시가 연결을 사용할 수 있을 때까지 기다리는 시간을 지정합니다.
   - 초기화 쿼리. (선택 사항) 초기화 쿼리를 추가하거나 현재 쿼리를 수정합니다. 각 새 데이터베이스 접속을 열 때 실행할 프록시에 대한 하나 이상의 SQL 문을 지정할 수 있습니다. 이 설정은 일반적으로 각 접속에 동일한 설정이 있는지 확인하기 위해 `SET` 문과 함께 사용됩니다. 추가하는 쿼리가 유효한지 확인합니다. 단일 `SET` 문에 여러 변수를 포함하려면 쉼표 구분자를 사용합니다. 예시:

     ```
     SET variable1=value1, variable2=value2
     ```

     여러 문의 경우 세미콜론을 구분 기호로 사용합니다.

   대상 그룹 식별자, 데이터베이스 엔진과 같은 특정 속성은 변경할 수 없습니다.
5. Modify target group(대상 그룹 수정)을 선택합니다.

AWS CLI

AWS CLI를 사용하여 프록시를 수정하려면 modify-db-proxy, modify-db-proxy-target-group, deregister-db-proxy-targets 및 register-db-proxy-targets 명령을 사용합니다.

`modify-db-proxy` 명령을 사용하여 다음과 같은 속성을 변경할 수 있습니다.

- 프록시에서 사용하는 Secrets Manager 보안 암호 집합입니다.
- TLS가 필요한지 여부입니다.
- 유휴 클라이언트 시간 초과.
- 디버깅을 위해 SQL 문에서 추가 정보를 로깅할지 여부입니다.
- Secrets Manager 보안 암호를 검색하는 데 사용되는 IAM 역할입니다.
- 프록시에서 사용하는 보안 그룹입니다.

다음 예제에서는 기존 프록시의 이름을 바꾸는 방법을 보여 줍니다.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

연결 관련 설정을 수정하거나 대상 그룹의 이름을 변경하려면 `modify-db-proxy-target-group` 명령을 사용합니다. 현재 모든 프록시에는 `default`라는 단일 대상 그룹이 있습니다. 이 대상 그룹으로 작업하는 경우 프록시 이름을 지정하고 대상 그룹 이름에 `default`를 지정합니다. 기본 대상 그룹의 이름은 바꿀 수 없습니다.

다음 예제에서는 대상 그룹을 사용하여 프록시에 대한 `MaxIdleConnectionsPercent` 설정을 먼저 확인한 다음 변경하는 방법을 보여 줍니다.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
```

```
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

`deregister-db-proxy-targets` 및 `register-db-proxy-targets` 명령을 사용하여 대상 그룹을 통해 프록시가 연결된 Aurora DB 클러스터를 변경합니다. 현재 각 프록시는 하나의 Aurora DB 클러스터에 연결할 수 있습니다. 대상 그룹은 모든 Aurora 클러스터의 전체 DB 인스턴스에 대한 연결 세부 정보를 추적합니다.

다음 예제는 프록시가 `cluster-56-2020-02-25-1399`라는 Aurora MySQL 클러스터와 연결된 상태에서 시작합니다. 이 예제에서는 `provisioned-cluster`라는 다른 클러스터에 연결할 수 있도록 프록시를 변경하는 방법을 보여 줍니다.

Aurora DB 클러스터로 작업하는 경우 `--db-cluster-identifier` 옵션을 지정합니다.

다음 예제에서는 Aurora MySQL 프록시를 수정합니다. Aurora PostgreSQL 프록시는 포트 5432가 있습니다.

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy
```

```
{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}
```

```
aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

RDS API를 사용하여 프록시를 수정하려면 ModifyDBProxy, ModifyDBProxyTargetGroup, DeregisterDBProxyTargets 및 RegisterDBProxyTargets 작업을 사용합니다.

`ModifyDBProxy`를 사용하면 다음과 같은 속성을 변경할 수 있습니다.

- 프록시에서 사용하는 Secrets Manager 보안 암호 집합입니다.
- TLS가 필요한지 여부입니다.
- 유휴 클라이언트 시간 초과.
- 디버깅을 위해 SQL 문에서 추가 정보를 로깅할지 여부입니다.
- Secrets Manager 보안 암호를 검색하는 데 사용되는 IAM 역할입니다.
- 프록시에서 사용하는 보안 그룹입니다.

ModifyDBProxyTargetGroup을 사용하여 연결 관련 설정을 수정할 수 있습니다. 현재 모든 프록시에는 default라는 단일 대상 그룹이 있습니다. 이 대상 그룹으로 작업하는 경우 프록시 이름을 지정하고 대상 그룹 이름에 default를 지정합니다. 기본 대상 그룹의 이름은 바꿀 수 없습니다.

DeregisterDBProxyTargets 및 RegisterDBProxyTargets를 사용하여 대상 그룹을 통해 프록시가 연결된 Aurora 클러스터를 변경합니다. 현재 각 프록시는 하나의 Aurora DB 클러스터에 연결할 수 있습니다. 대상 그룹은 Aurora 클러스터의 DB 인스턴스에 대한 연결 세부 정보를 추적합니다.

RDS 프록시 사용 시 새 데이터베이스 사용자 추가

프록시와 연결된 Aurora 클러스터에 새 데이터베이스 사용자를 추가해야 할 경우가 있습니다. 그렇다면 Secrets Manager 비밀을 추가하거나 재사용하여 해당 사용자의 자격 증명을 저장합니다. 이를 수행하려면 다음 단계를 수행합니다.

1. AWS Secrets Manager에서 RDS 프록시용 데이터베이스 자격 증명 설정에 설명된 절차를 사용하여 새 Secrets Manager 비밀을 만듭니다.
2. IAM 역할을 업데이트하여 RDS Proxy에 새 Secrets Manager 비밀에 대한 액세스 권한을 부여합니다. 이렇게 하려면 IAM 역할 정책의 리소스 섹션을 업데이트합니다.
3. Secrets Manager 보안 암호 아래에서 RDS 프록시를 수정하여 새 Secrets Manager 보안 암호를 추가합니다.
4. 새 사용자가 기존 사용자를 대신하는 경우 기존 사용자의 프록시 Secrets Manager 비밀에 저장된 자격 증명을 업데이트합니다.

RDS 프록시 사용 시 PostgreSQL 데이터베이스에 새 데이터베이스 사용자 추가

새 사용자를 PostgreSQL 데이터베이스에 추가할 때 다음 명령을 실행한 경우:

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

대상 데이터베이스의 연결을 모니터링할 수 있도록 rdsproxyadmin 사용자에게 CONNECT 권한을 부여합니다.

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

위 명령에서 rdsproxyadmin을 데이터베이스 사용자로 변경하여 다른 대상 데이터베이스 사용자가 상태 확인을 수행하도록 허용할 수도 있습니다.

RDS 프록시 사용 시 데이터베이스 사용자 암호 변경

프록시와 연결된 Aurora 클러스터에서 데이터베이스 사용자의 암호를 변경해야 할 경우가 있습니다. 그렇다면 해당 Secrets Manager 비밀을 새 암호로 업데이트합니다.

RDS 프록시 연결 고려 사항

연결 설정 구성

RDS 프록시의 연결 풀링을 조정하려면 다음 설정을 수정합니다.

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

프록시에 의해 종료되기 전에 클라이언트 연결이 유휴 상태일 수 있는 시간을 지정합니다. 기본값은 1,800초(30분)입니다.

애플리케이션이 이전 요청이 완료된 후 지정된 시간 내에 새 요청을 제출하지 않으면 클라이언트 연결이 유휴 상태로 간주됩니다. 기본 데이터베이스 연결은 열린 상태를 유지하고 연결 풀로 반환됩니다. 따라서 새 클라이언트 연결에 다시 사용할 수 있습니다. 프록시가 기간 경과 연결을 사전에 제거하도록 하려면 유휴 클라이언트 연결 제한 시간을 줄이는 것이 좋습니다. 워크로드가 프록시와 자주 연결하는 경우 연결 설정 비용을 절약하기 위해 유휴 클라이언트 연결 제한 시간을 늘리세요.

이 설정은 RDS 콘솔의 유휴 클라이언트 연결 제한 시간(Idle client connection timeout) 필드와 AWS CLI 및 API의 `IdleClientTimeout` 설정으로 표시됩니다. RDS 콘솔에서 유휴 클라이언트 연결 제한 시간(Idle client connection timeout) 필드의 값을 변경하는 방법을 알아보려면 AWS Management Console 섹션을 참조하세요. `IdleClientTimeout` 설정의 값을 변경하는 방법을 알아보려면 CLI 명령 modify-db-proxy 또는 API 작업 ModifyDBProxy를 참조하세요.

MaxConnectionsPercent

RDS 프록시가 대상 데이터베이스와 설정할 수 있는 연결 수를 제한할 수 있습니다. 데이터베이스에 사용할 수 있는 최대 연결의 백분율로 제한을 지정합니다. 이 설정은 RDS 콘솔의 연결 풀 최대 연결(Connection pool maximum connections) 필드와 AWS CLI 또는 API의 `MaxConnectionsPercent` 설정으로 표시됩니다.

MaxConnectionsPercent 값은 대상 그룹에서 사용하는 Aurora DB 클러스터에 대한 max_connections 설정의 백분율로 표시됩니다. 프록시가 이러한 연결을 모두 미리 생성하지는 않습니다. 이 설정을 통해 프록시는 워크로드에 필요한 연결을 설정할 수 있습니다.

예를 들어 등록된 데이터베이스 대상이 max_connections가 1000으로 설정되어 있고 MaxConnectionsPercent가 95로 설정된 경우, RDS 프록시는 950개의 연결을 해당 데이터베이스 대상에 대한 동시 연결의 상한선으로 설정합니다.

워크로드가 허용된 최대 데이터베이스 연결 수에 도달할 경우 발생하는 일반적인 부작용은 전체 쿼리 지연 시간이 늘어나고, 그와 함께 DatabaseConnectionsBorrowLatency 지표도 증가한다는 점입니다. DatabaseConnections 및 MaxDatabaseConnectionsAllowed 지표를 비교하여 현재 사용된 데이터베이스 연결 수와 허용된 총 데이터베이스 연결 수를 모니터링할 수 있습니다.

이 파라미터를 설정할 때는 다음과 같은 모범 사례를 고려하세요.

- 워크로드 패턴의 변동에 대비하여 충분한 연결 여유 용량을 확보하세요. 파라미터를 최근에 모니터링한 최대 사용량보다 30% 이상 높게 설정하는 것이 좋습니다. RDS 프록시는 데이터베이스 연결 할당량을 여러 노드에 재분배하므로, 차용 지연 시간이 늘어나는 것을 방지하기 위해 내부 용량 변경 시 추가 연결을 위한 최소 30%의 여유 용량이 필요할 수 있습니다.
- RDS 프록시는 빠른 장애 조치, 트래픽 라우팅, 내부 작업을 지원하기 위해 활성 모니터링을 위한 특정 수의 연결을 예약합니다. MaxDatabaseConnectionsAllowed 지표에는 이러한 예약된 연결이 포함되지 않습니다. 이는 워크로드를 처리하는 데 사용 가능한 연결 수를 나타내며 MaxConnectionsPercent 설정에서 파생된 값보다 낮을 수 있습니다.

  최소 권장 MaxConnectionsPercent 값은 다음과 같습니다.

  - db.t3.small: 100
  - db.t3.medium: 55
  - db.t3.large: 35
  - db.r3.large 이상: 20

  리더 노드가 있는 Aurora 클러스터 같은 여러 대상 인스턴스가 RDS 프록시에 등록되어 있는 경우, 가장 작은 등록 인스턴스를 기준으로 최소값을 설정합니다.

RDS 콘솔에서 연결 풀 최대 연결(Connection pool maximum connections) 필드의 값을 변경하는 방법을 알아보려면 AWS Management Console 섹션을 참조하세요. MaxConnectionsPercent 설정의 값을 변경하는 방법을 알아보려면 CLI 명령 modify-db-proxy-target-group 또는 API 작업 ModifyDBProxyTargetGroup을 참조하세요.

> **Important**
>
> DB 클러스터가 쓰기 전달이 켜져 있는 글로벌 데이터베이스의 일부인 경우, 쓰기 전달에 할당된 할당량만큼 프록시의 `MaxConnectionsPercent` 값을 줄입니다. 쓰기 전달 할당량은 DB 클러스터 파라미터 `aurora_fwd_writer_max_connections_pct`에 설정됩니다. 쓰기 전달에 대한 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용](#) 단원을 참조하십시오.

데이터베이스 연결 한도에 대한 자세한 내용은 [Aurora MySQL DB 인스턴스에 대한 최대 연결](#)과 [Aurora PostgreSQL DB 인스턴스에 대한 최대 연결](#)을 참조하세요.

MaxIdleConnectionsPercent

RDS 프록시가 연결 풀에서 유지할 수 있는 유휴 데이터베이스 연결 수를 제어할 수 있습니다. 기본적으로, RDS 프록시는 5분 동안 연결에 대한 활동이 없으면 풀의 데이터베이스 연결을 유휴 상태로 간주합니다.

`MaxIdleConnectionsPercent` 값은 RDS DB 인스턴스 대상 그룹에 대한 `max_connections` 설정의 백분율로 표시됩니다. 기본값은 `MaxConnectionsPercent`의 50%이고 상한은 `MaxConnectionsPercent` 값입니다. 예를 들어, `MaxConnectionsPercent`가 80인 경우 `MaxIdleConnectionsPercent`의 기본값은 40입니다.

값이 높으면 프록시가 유휴 데이터베이스 연결 비율을 높게 유지할 수 있습니다. 값이 낮으면 프록시가 높은 유휴 데이터베이스 연결 비율을 닫습니다. 워크로드를 예측할 수 없는 경우 `MaxIdleConnectionsPercent`에 높은 값을 설정하는 것을 고려해 보세요. 이렇게 하면 RDS 프록시가 새 데이터베이스 연결을 많이 열지 않고도 급증하는 활동을 수용할 수 있습니다.

이 설정은 AWS CLI 또는 API에서 `DBProxyTargetGroup`의 `MaxIdleConnectionsPercent` 설정으로 표시됩니다. `MaxIdleConnectionsPercent` 설정의 값을 변경하는 방법을 알아보려면 CLI 명령 [modify-db-proxy-target-group](#) 또는 API 작업 [ModifyDBProxyTargetGroup](#)을 참조하세요.

데이터베이스 연결 한도에 대한 자세한 내용은 [Aurora MySQL DB 인스턴스에 대한 최대 연결](#)과 [Aurora PostgreSQL DB 인스턴스에 대한 최대 연결](#)을 참조하세요.

ConnectionBorrowTimeout

시간 초과 오류를 반환하기 전에 RDS 프록시가 연결 풀의 데이터베이스 연결을 사용할 수 있을 때까지 기다리는 시간을 선택할 수 있습니다. 기본값은 120초입니다. 이 설정은 연결 수가 최대일 때, 즉 연결

풀에서 사용할 수 있는 연결이 없을 때 적용됩니다. 이는 또한 장애 조치 작업이 진행 중인 경우와 같이 요청을 처리할 수 있는 적절한 데이터베이스 인스턴스가 없는 경우에도 적용됩니다. 이 설정을 사용하면 애플리케이션 코드에서 쿼리 시간 초과를 변경하지 않고도 애플리케이션에 가장 적합한 대기 기간을 설정할 수 있습니다.

이 설정은 RDS 콘솔의 연결 차용 제한 시간(Connection borrow timeout) 필드나 AWS CLI 또는 API의 `DBProxyTargetGroup`의 `ConnectionBorrowTimeout` 설정으로 표시됩니다. RDS 콘솔에서 연결 차용 제한 시간(Connection borrow timeout) 필드의 값을 변경하는 방법을 알아보려면 AWS Management Console 섹션을 참조하세요. `ConnectionBorrowTimeout` 설정의 값을 변경하는 방법을 알아보려면 CLI 명령 modify-db-proxy-target-group 또는 API 작업 ModifyDBProxyTargetGroup을 참조하세요.

클라이언트 및 데이터베이스 연결

애플리케이션에서 RDS 프록시로의 연결을 클라이언트 연결이라고 합니다. 프록시에서 데이터베이스로의 연결은 데이터베이스 연결입니다. RDS 프록시를 사용하는 경우 데이터베이스 연결은 RDS 프록시 내에서 관리되는 반면 클라이언트 연결은 프록시에서 종료됩니다.

애플리케이션 측 연결 풀링은 애플리케이션과 RDS 프록시 간의 반복적인 연결 설정을 줄이는 이점을 제공할 수 있습니다.

애플리케이션 측 연결 풀을 구현하기 전에 다음과 같은 구성 측면을 고려하세요.

- 클라이언트 연결 최대 수명: RDS 프록시는 클라이언트 연결의 최대 수명을 24시간으로 제한합니다. 이 값은 구성할 수 없습니다. 예상치 못한 클라이언트 연결 끊김을 방지하려면 최대 연결 수명을 24시간 미만으로 설정하여 풀을 구성하세요.
- 클라이언트 연결 유휴 제한 시간: RDS 프록시는 클라이언트 연결에 최대 유휴 시간을 적용합니다. 예상치 못한 연결 끊김을 방지하려면 RDS 프록시의 클라이언트 연결 유휴 제한 시간 설정보다 낮은 값으로 풀의 유휴 연결 제한 시간을 구성하세요.

애플리케이션 측 연결 풀에 구성된 최대 클라이언트 연결 수를 RDS 프록시의 max_connections 설정으로 제한할 필요는 없습니다.

클라이언트 연결 풀링을 사용하면 클라이언트 연결 수명이 길어집니다. 연결이 고정되는 경우 클라이언트 연결을 풀링하면 멀티플렉싱 효율성이 저하될 수 있습니다. 고정되어 있지만 애플리케이션 측 연결 풀에서 유휴 상태인 클라이언트 연결은 데이터베이스 연결을 계속 유지하고 다른 클라이언트 연결에서 데이터베이스 연결을 재사용하지 못하게 합니다. 프록시 로그를 검토하여 연결 고정이 발생하는지 확인하세요.

> **Note**
>
> RDS 프록시는 더 이상 사용되지 않을 경우 24시간 후에 데이터베이스 연결을 닫습니다. 프록시는 최대 유휴 연결 설정 값에 관계없이 이 작업을 수행합니다.

RDS 프록시 고정 방지

멀티플렉싱은 데이터베이스 요청이 이전 요청의 상태 정보에 의존하지 않을 때 더 효율적입니다. 이 경우 RDS Proxy는 각 트랜잭션이 완료될 때 연결을 다시 사용할 수 있습니다. 이러한 상태 정보의 예로는 SET 또는 SELECT 문을 통해 변경할 수 있는 대부분의 변수 및 구성 파라미터가 있습니다. 클라이언트 연결에 대한 SQL 트랜잭션은 기본적으로 기본 데이터베이스 연결 간에 멀티플렉싱할 수 있습니다.

프록시에 대한 연결은 고정이라는 상태로 들어갈 수 있습니다. 연결이 고정되면 이후의 각 트랜잭션은 세션이 끝날 때까지 동일한 기본 데이터베이스 연결을 사용합니다. 다른 클라이언트 연결도 세션이 끝날 때까지 해당 데이터베이스 연결을 다시 사용할 수 없습니다. 클라이언트 연결이 끊어지면 세션이 종료됩니다.

RDS Proxy는 다른 세션에 적합하지 않은 세션 상태 변경을 감지하면 클라이언트 연결을 특정 DB 연결에 자동으로 고정합니다. 고정은 연결 재사용의 효과를 줄입니다. 모든 또는 거의 모든 연결이 고정되는 경우 애플리케이션 코드 또는 워크로드를 수정하여 고정을 유발하는 조건을 줄이는 것이 좋습니다.

예를 들어, 애플리케이션이 세션 변수 또는 구성 파라미터를 변경한다고 가정하겠습니다. 이 경우 이후의 문이 효력을 발휘하기 위해 새 변수 또는 파라미터를 사용할 수 있습니다. 따라서 RDS Proxy는 세션 변수 또는 구성 설정을 변경하라는 요청을 처리할 때 해당 세션을 DB 연결에 고정합니다. 이렇게 하면 동일한 세션의 모든 이후 트랜잭션에 대해 세션 상태가 유효하게 유지됩니다.

일부 데이터베이스의 경우 이 규칙은 설정할 수 있는 모든 파라미터에 적용되지는 않습니다. RDS 프록시는 특정 문과 변수를 추적합니다. 따라서 RDS 프록시는 이들을 수정할 때 세션을 고정하지 않습니다. 이 경우 RDS 프록시는 해당 설정에 대해 동일한 값을 가진 다른 세션에 대해서만 연결을 재사용합니다. Aurora MySQL 추적된 문 및 변수 목록은 RDS 프록시가 Aurora MySQL 데이터베이스에서 추적하는 대상 섹션을 참조하세요.

RDS 프록시가 Aurora MySQL 데이터베이스에서 추적하는 대상

다음은 RDS 프록시가 추적하는 MySQL 문입니다.

- DROP DATABASE
- DROP SCHEMA

- USE

다음은 RDS 프록시가 추적하는 MySQL 변수입니다.

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)
- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

고정 최소화

RDS 프록시 성능 튜닝에는 고정을 최소화하여 트랜잭션 수준 연결 재사용(멀티플렉싱)을 최대화하려는 시도가 포함됩니다.

다음 작업을 수행하여 고정을 최소화할 수 있습니다.

- 고정을 야기할 수 있는 불필요한 데이터베이스 요청을 방지합니다.
- 모든 연결에서 변수와 구성 설정을 일관되게 설정합니다. 그렇게 하면 이후 세션에서 특정 설정이 있는 연결을 재사용할 가능성이 증가합니다.

  그러나 PostgreSQL의 경우 변수를 설정하면 세션이 고정됩니다.
- MySQL 엔진 패밀리 데이터베이스의 경우 프록시에 세션 고정 필터를 적용합니다. 이렇게 해도 애플리케이션의 올바른 작동에 영향을 주지 않음이 확인될 경우 특정 유형의 작업을 세션 고정에서 제외할 수 있습니다.
- Amazon CloudWatch 지표 중 `DatabaseConnectionsCurrentlySessionPinned`를 모니터링하여 고정이 얼마나 자주 발생하는지 확인합니다. 이 지표와 기타 CloudWatch 지표에 대한 자세한 내용은 [Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링](#) 단원을 참조하십시오.
- `SET` 문을 사용하여 각 클라이언트 연결에 대해 동일한 초기화를 수행하는 경우 트랜잭션 수준 멀티플렉싱을 유지하면서 그렇게 할 수 있습니다. 이 경우 초기 세션 상태를 설정하는 문을 프록시에서 사용하는 초기화 쿼리로 이동합니다. 이 속성은 세미콜론으로 구분된 하나 이상의 SQL 문을 포함하는 문자열입니다.

  예를 들어, 특정 구성 파라미터를 설정하는 프록시에 대한 초기화 질의를 정의할 수 있습니다. 그러면 RDS Proxy는 해당 프록시에 대해 새 연결을 설정할 때마다 해당 설정을 적용합니다. 애플리케이션 코드에서 해당 `SET` 문을 제거하여 트랜잭션 수준 멀티플렉싱을 방해하지 않도록 할 수 있습니다.

  특정 프록시에서 고정이 발생하는 빈도에 대한 지표를 보려면 [Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링](#) 단원을 참조하십시오.

모든 엔진 패밀리에 대해 고정을 유발하는 조건

다음과 같이 멀티플렉싱으로 예기치 않은 동작이 발생할 수 있는 상황에서 프록시는 세션을 현재 연결에 고정합니다.

- 텍스트 크기가 16KB보다 큰 명령문을 사용하면 프록시가 세션을 고정합니다.

Aurora MySQL에서 고정이 발생하는 조건

MySQL의 경우 다음 상호 작용으로 인한 고정도 발생합니다.

- 명시적 테이블 잠금 문인 `LOCK TABLE`, `LOCK TABLES` 또는 `FLUSH TABLES WITH READ LOCK`을 사용하면 프록시가 세션을 고정합니다.

- GET_LOCK을 사용하여 명명된 잠금을 만들면 프록시가 세션을 고정합니다.
- 사용자 변수 또는 시스템 변수(일부 예외)를 설정하면 프록시가 세션을 고정합니다. 이 경우 연결 재사용이 너무 줄어들면 SET 작업이 고정되지 않도록 선택합니다. 세션 고정 필터 속성을 설정하여 이를 수행하는 방법에 대한 자세한 내용은 [Amazon Aurora용 프록시 만들기](#) 및 [RDS 프록시 수정](#) 섹션을 참조하세요.
- 임시 테이블을 생성하면 프록시가 세션을 고정합니다. 이렇게 하면 트랜잭션 경계에 관계없이 임시 테이블의 내용이 세션 전체에서 보존됩니다.
- 함수 ROW_COUNT, FOUND_ROWS 및 LAST_INSERT_ID를 호출하면 고정을 야기하는 경우가 있습니다.

  이러한 함수가 고정을 야기하는 정확한 상황은 MySQL 5.7과 호환되는 Aurora MySQL 버전에 따라 다를 수 있습니다.
- 준비된 문을 사용하면 프록시가 세션을 고정합니다. 이 규칙은 준비된 문이 SQL 텍스트를 사용하는지 이진 프로토콜을 사용하는지 여부를 적용합니다.
- SET LOCAL을 사용할 경우 RDS 프록시는 연결을 고정하지 않습니다.
- 저장 프로시저 및 저장 함수를 호출해도 고정이 발생하지 않습니다. RDS 프록시는 이러한 호출로 인한 세션 상태 변경을 감지하지 못합니다. 트랜잭션 전반에 걸쳐 세션 상태를 유지하려고 해당 세션 상태에 의존하는 경우 저장된 루틴 내에서 애플리케이션이 세션 상태를 변경하지 않는지 확인합니다. 예를 들어, RDS 프록시는 현재 모든 트랜잭션에 걸쳐 지속되는 임시 테이블을 생성하는 저장 프로시저와 호환되지 않습니다.

애플리케이션 동작에 대한 전문 지식이 있는 경우 특정 애플리케이션 문에서 고정 동작을 건너뛸 수 있습니다. 그러려면 프록시를 생성할 때 세션 고정 필터 옵션을 선택하면 됩니다. 현재 세션 변수 및 구성 설정을 설정하기 위해 세션 고정을 옵트아웃할 수 있습니다.

Aurora PostgreSQL에서 고정이 발생하는 조건

PostgreSQL의 경우 다음 상호 작용으로 인해 고정도 발생합니다.

- SET 명령 사용.
- PREPARE, DISCARD, DEALLOCATE 또는 EXECUTE 명령을 사용하여 준비된 문 관리.
- 임시 시퀀스, 테이블 또는 뷰 생성.
- 커서 선언.
- 세션 상태 무시.
- 알림 채널에서 수신.

- auto_explain과 같은 라이브러리 모듈 로드.
- nextval 및 setval과 같은 함수를 사용하여 시퀀스 조작.
- pg_advisory_lock 및 pg_try_advisory_lock과 같은 함수를 사용하여 잠금과 상호 작용.

> **Note**
>
> RDS 프록시는 트랜잭션 수준 권고 잠금(특히 pg_advisory_xact_lock, pg_advisory_xact_lock_shared, pg_try_advisory_xact_lock, pg_try_advisory_xact_lock_shared)을 고정하지 않습니다.

- 파라미터 설정 또는 파라미터를 기본값으로 재설정. 특히 SET 및 set_config 명령을 사용하여 세션 변수에 기본값 할당.
- 저장 프로시저 및 저장 함수를 호출해도 고정이 발생하지 않습니다. RDS 프록시는 이러한 호출로 인한 세션 상태 변경을 감지하지 못합니다. 트랜잭션 전반에 걸쳐 세션 상태를 유지하려고 해당 세션 상태에 의존하는 경우 저장된 루틴 내에서 애플리케이션이 세션 상태를 변경하지 않는지 확인합니다. 예를 들어, RDS 프록시는 현재 모든 트랜잭션에 걸쳐 지속되는 임시 테이블을 생성하는 저장 프로시저와 호환되지 않습니다.

RDS 프록시 삭제

더 이상 필요하지 않은 프록시는 삭제할 수 있습니다. 또는 프록시와 연결된 DB 인스턴스 또는 클러스터를 서비스 중단 상태로 전환하면 프록시를 삭제할 수 있습니다.

AWS Management Console

프록시를 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 목록에서 삭제할 프록시를 선택합니다.
4. Delete Proxy(프록시 삭제)를 선택합니다.

AWS CLI

DB 프록시를 삭제하려면 AWS CLI 명령 delete-db-proxy를 사용합니다. 관련 연결을 제거하려면 deregister-db-proxy-targets 명령도 사용합니다.

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

DB 프록시를 삭제하려면 Amazon RDS API 함수 DeleteDBProxy를 호출합니다. 관련 항목 및 연결을 삭제하려면 DeleteDBProxyTargetGroup 및 DeregisterDBProxyTargets 함수를 호출합니다.

Amazon RDS 프록시 엔드포인트 작업

RDS Proxy 엔드포인트는 데이터베이스 연결을 관리하는 유연하고 효율적인 방법을 제공하여 확장성, 가용성 및 보안을 개선합니다. 프록시 엔드포인트를 사용하면 다음을 수행할 수 있습니다.

- 모니터링 및 문제 해결 간소화 – 여러 엔드포인트를 사용하여 여러 애플리케이션의 연결을 독립적으로 모니터링하고 문제를 해결합니다.
- 읽기 확장성 향상 - Aurora DB 클러스터와 함께 리더 엔드포인트를 활용하여 읽기 트래픽을 효율적으로 분산함으로써 쿼리가 많은 워크로드의 성능을 최적화합니다.
- 크로스 VPC 액세스 활성화 - 크로스 VPC 엔드포인트를 사용하여 VPC 간에 데이터베이스를 연결합니다. 이를 통해 한 VPC의 Amazon EC2 인스턴스와 같은 리소스가 다른 VPC의 데이터베이스에 액세스할 수 있습니다.

주제

- 프록시 엔드포인트 삭제

프록시 엔드포인트 개요

RDS 프록시 엔드포인트 작업을 수행하려면 Aurora DB 클러스터 및 리더 엔드포인트와 동일한 절차를 따릅니다. Aurora 엔드포인트에 익숙하지 않은 경우 Amazon Aurora 엔드포인트 연결에서 자세한 내용을 참조하세요.

RDS Proxy를 Aurora 클러스터와 함께 사용하는 경우 기본 엔드포인트는 읽기 및 쓰기 작업을 모두 지원합니다. 즉, 모든 요청을 클러스터의 라이터 인스턴스로 라우팅하여 `max_connections` 제한에 기여합니다. 트래픽을 더 잘 분산하기 위해 프록시에 대한 추가 읽기/쓰기 또는 읽기 전용 엔드포인트를 생성하여 더욱 효율적인 워크로드 관리와 향상된 확장성을 제공할 수 있습니다.

Aurora 프로비저닝 클러스터의 리더 엔드포인트와 마찬가지로 프록시와 함께 읽기 전용 엔드포인트를 사용하여 읽기 쿼리를 처리합니다. 이 접근 방식은 하나 이상의 리더 DB 인스턴스에 쿼리를 분산하여 Aurora의 읽기 확장성을 극대화합니다. 읽기 전용 엔드포인트를 사용하고 필요에 따라 리더 인스턴스를 추가하면 클러스터가 처리할 수 있는 동시 쿼리 및 연결 수를 늘릴 수 있습니다.

> **Tip**
>
> AWS Management Console을 사용하여 Aurora 클러스터에 대한 프록시를 생성할 때 RDS 프록시를 선택하여 리더 엔드포인트를 자동으로 생성할 수 있습니다. 리더 엔드포인트의 이점에 대한 자세한 내용은 Aurora 클러스터에 리더 엔드포인트 사용 섹션을 참조하세요.

프록시 엔드포인트를 만들면 엔드포인트를 프록시의 VPC와 다른 가상 프라이빗 클라우드(VPC)와 연결할 수 있습니다. 이렇게 하면 조직 내 다른 애플리케이션에서 사용하는 VPC와 같은 다른 VPC에서 프록시에 연결할 수 있습니다.

프록시 엔드포인트과 관련한 제한 사항에 대한 자세한 내용은 프록시 엔드포인트에 대한 제한 사항 섹션을 참조하세요.

RDS Proxy 로그에서 각 항목에는 연결된 프록시 엔드포인트의 이름이 접두사로 붙습니다. 이 이름은 사용자 정의 엔드포인트에 대해 지정한 이름이거나, 프록시의 기본 읽기/쓰기 엔드포인트에 대한 특수한 이름 `default`일 수 있습니다.

각 프록시 엔드포인트에는 일련의 자체 CloudWatch 지표가 있습니다. 모든 프록시 엔드포인트, 특정 엔드포인트 또는 프록시의 모든 읽기/쓰기 또는 읽기 전용 엔드포인트에 대한 지표를 모니터링합니다. 자세한 내용은 Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링 섹션을 참조하세요.

프록시 엔드포인트는 연결된 프록시와 동일한 인증 메커니즘을 사용합니다. RDS Proxy는 연결된 프록시의 속성과 일치하는 사용자 정의 엔드포인트에 대한 권한 및 승인을 자동으로 설정합니다.

Aurora Global Database의 DB 클러스터에서 프록시 엔드포인트가 작동하는 방법을 알아보려면 RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식 섹션을 참조하세요.

프록시 엔드포인트에 대한 제한 사항

RDS 프록시 엔드포인트에는 다음과 같은 제한 사항이 있습니다.

- RDS 프록시 기본 엔드포인트는 수정할 수 없습니다.
- 프록시의 사용자 정의 엔드포인트의 최대 개수는 20개입니다. 따라서 프록시는 최대 21개의 엔드포인트(기본 엔드포인트와 사용자가 생성하는 20개)를 가질 수 있습니다.
- 추가 엔드포인트를 프록시와 연결할 때 RDS Proxy는 클러스터에서 각 엔드포인트에 사용할 DB 인스턴스를 자동으로 결정합니다. Aurora 사용자 지정 엔드포인트처럼 특정 인스턴스를 지정할 수는 없습니다.

프록시를 생성하면 RDS는 애플리케이션과 데이터베이스 간의 보안 통신을 위해 VPC 엔드포인트를 자동으로 만듭니다. VPC 엔드포인트는 Amazon VPC 콘솔에서 보고 액세스할 수 있습니다.

새 프록시 엔드포인트를 추가하면 AWS PrivateLink 인터페이스 엔드포인트가 프로비저닝됩니다. 프록시에 엔드포인트를 하나 이상 추가하면 추가 요금이 발생합니다. 자세한 내용은 RDS Proxy Pricing을 참조하세요.

Aurora 클러스터에 리더 엔드포인트 사용

Aurora 클러스터에 RDS Proxy를 사용할 경우 리더 엔드포인트라는 읽기 전용 엔드포인트를 생성하고 연결할 수 있습니다. 이러한 리더 엔드포인트는 쿼리 집약적 애플리케이션의 읽기 확장성을 개선하는 데 도움이 됩니다. 또한 리더 엔드포인트는 클러스터의 리더 DB 인스턴스를 사용할 수 없게 될 경우 연결 가용성을 높이는 데에도 도움이 됩니다.

> **Note**
>
> 새 엔드포인트를 읽기 전용으로 지정할 경우, RDS Proxy를 사용하려면 Aurora 클러스터에 리더 DB 인스턴스가 하나 이상 있어야 합니다. 경우에 따라 프록시의 대상을 단일 라이터만 포함하는 Aurora 클러스터로 변경할 수 있습니다. 그렇게 하면 오류로 인해 읽기 전용 엔드포인트에 대한 요청에 실패합니다. 프록시의 대상이 Aurora 클러스터가 아니라 RDS 인스턴스인 경우에도 요청이 실패합니다.

Aurora 클러스터에 리더 인스턴스가 있지만 해당 인스턴스를 사용할 수 없는 경우, RDS Proxy는 즉시 오류를 반환하는 대신 요청을 전송할 때까지 기다립니다. 연결 대여 제한 시간 내에 리더 인스턴스가 사용 가능한 상태가 되지 않으면 오류가 발생하여 요청이 실패합니다.

리더 엔드포인트가 애플리케이션 가용성을 높이는 방법

경우에 따라 클러스터에 있는 하나 이상의 리더 인스턴스를 사용할 수 없게 될 수 있습니다. 이 경우 DB 프록시의 리더 엔드포인트를 사용하는 연결은 Aurora 리더 엔드포인트를 사용하는 연결보다 더 빠르게 복구됩니다. RDS Proxy는 클러스터에서 사용 가능한 리더 인스턴스로만 연결을 라우팅합니다. 인스턴스를 사용할 수 없게 될 때 DNS 캐싱으로 인해 지연이 발생하지 않습니다.

연결이 다중화된 경우 RDS Proxy는 애플리케이션을 중단하지 않고 후속 쿼리를 다른 리더 DB 인스턴스로 보냅니다. 새 리더 인스턴스로 자동 전환하는 동안 RDS Proxy는 이전 및 새 리더 인스턴스의 복제 지연을 확인합니다. RDS Proxy는 새 리더 인스턴스가 이전 리더 인스턴스와 동일한 변경 사항이 적용된 최신 버전인지 확인합니다. 따라서 RDS Proxy가 리더 DB 인스턴스 간에 전환할 때 애플리케이션에서 오래된 데이터 문제가 발생하지 않습니다.

연결이 고정된 경우 연결에 대한 다음 쿼리에서 오류가 반환됩니다. 그러나 애플리케이션은 동일한 엔드포인트에 즉시 다시 연결할 수 있습니다. RDS Proxy는 `available` 상태에 있는 다른 리더 DB 인스턴스로 연결을 라우팅합니다. 수동으로 다시 연결하면 RDS Proxy가 이전 리더 인스턴스와 새 리더 인스턴스 간의 복제 지연을 확인하지 않습니다.

Aurora 클러스터에 사용 가능한 리더 인스턴스가 없는 경우 RDS Proxy는 이 조건이 일시적인지 영구적인지 여부를 확인합니다. 각 경우의 동작은 다음과 같습니다.

- 클러스터에 하나 이상의 리더 DB 인스턴스가 있지만 그 중 어느 것도 `Available` 상태가 아닙니다. 예를 들어 모든 리더 인스턴스가 재부팅되거나 문제가 발생했을 수 있습니다. 이 경우 리더 엔드포인트에 대한 연결을 시도하면 리더 인스턴스를 사용할 수 있을 때까지 기다립니다. 연결 대여 제한 시간 내에 리더 인스턴스가 사용 가능한 상태가 되지 않으면 연결 시도가 실패합니다. 리더 인스턴스를 사용할 수 있게 되면 연결 시도가 성공합니다.
- 클러스터에 리더 DB 인스턴스가 없다고 가정합니다. 이 경우 리더 엔드포인트에 연결하려고 하면 RDS Proxy가 즉시 오류를 반환합니다. 이 문제를 해결하려면 리더 엔드포인트에 연결하기 전에 하나 이상의 리더 인스턴스를 클러스터에 추가합니다.

리더 엔드포인트가 쿼리 확장성을 높이는 방법

프록시용 리더 엔드포인트는 다음과 같은 방법으로 Aurora 쿼리 확장성을 높입니다.

- 리더 인스턴스를 Aurora 클러스터에 추가하면 RDS Proxy가 리더 엔드포인트에 대한 새 연결을 다른 리더 인스턴스로 라우팅할 수 있습니다. 이렇게 하면 특정 리더 엔드포인트 연결을 사용하여 수행되는 쿼리가 다른 리더 엔드포인트 연결을 사용하여 수행되는 쿼리 속도를 저하시키지 않습니다. 쿼리는 별도의 DB 인스턴스에서 실행됩니다. 각 DB 인스턴스에는 자체 컴퓨팅 리소스, 버퍼 캐시 등이 있습니다.
- 가능한 경우 RDS Proxy는 특정 리더 엔드포인트 연결을 사용하는 모든 쿼리 문제에 대해 동일한 리더 DB 인스턴스를 사용합니다. 이렇게 하면 동일한 테이블에 있는 일련의 관련 쿼리가 특정 DB 인스턴스에서 캐싱, 계획 최적화 등을 활용할 수 있습니다.
- Reader DB 인스턴스를 사용할 수 없게 되면 세션이 멀티플렉싱되었는지 고정되어 있는지에 따라 애플리케이션에 미치는 영향이 달라집니다. 세션이 멀티플렉싱된 경우 RDS Proxy는 후속 쿼리를 사용자의 작업 없이 다른 리더 DB 인스턴스로 라우팅합니다. 세션이 고정되어 있으면 애플리케이션에 오류가 발생하여 다시 연결해야 합니다. 리더 엔드포인트에 즉시 다시 연결할 수 있으며 RDS Proxy는 연결을 사용 가능한 리더 DB 인스턴스로 라우팅합니다. 프록시 세션의 멀티플렉싱 및 고정에 대한 자세한 내용은 RDS Proxy 개념 개요 섹션을 참조하세요.
- 클러스터에 리더 DB 인스턴스가 많을수록 리더 엔드포인트를 사용하여 더 많은 동시 연결을 생성할 수 있습니다. 예를 들어 클러스터에 각각 200개의 동시 연결을 지원하도록 구성된 4개의 리더 DB 인스턴스가 있다고 가정합니다. 또한 프록시가 최대 연결의 50%를 사용하도록 구성되어 있다고 가정합니다. 여기서 프록시의 리더 엔드포인트를 통해 생성할 수 있는 최대 연결 수는 리더 1의 경우 100개(200개의 50%)입니다. 또한 리더 2에 대해 100개와 같은 식으로 총 400개까지 생성할 수 있습니다. 클러스터 리더 DB 인스턴스의 수를 8개로 두 배로 늘리면 리더 엔드포인트를 통한 최대 연결 수도 800개로 두 배가 됩니다.

리더 엔드포인트 사용의 예

다음 Linux 예제는 사용자가 리더 엔드포인트를 통해 Aurora MySQL 클러스터에 연결되었는지 확인하는 방법을 보여 줍니다. `innodb_read_only` 구성 설정이 활성화되어 있습니다. `CREATE DATABASE` 문과 같은 쓰기 작업을 수행하려고 하면 오류가 발생하여 실패합니다. 또한 `aurora_server_id` 변수를 확인하여, 리더 DB 인스턴스에 연결되어 있는지 확인할 수 있습니다.

> **Tip**
>
> DB 인스턴스 이름을 확인해야만 연결이 읽기/쓰기인지 아니면 읽기 전용인지 확인할 수 있습니다. Aurora 클러스터의 DB 인스턴스는 장애 조치가 발생할 때 라이터 및 리더 사이에서 역할을 바꿀 수 있습니다.

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+----------------------------------------+
| @@aurora_server_id                     |
+----------------------------------------+
| proxy-reader-endpoint-demo-instance-3  |
+----------------------------------------+
```

다음 예에서는 리더 DB 인스턴스가 삭제되더라도 프록시 리더 엔드포인트에 대한 연결이 어떻게 계속 작동하는지 보여 줍니다. 이 예에서 Aurora 클러스터에는 두 개의 리더 인스턴스, `instance-5507` 및 `instance-7448`이 있습니다. 리더 엔드포인트에 대한 연결이 리더 인스턴스 중 하나를 사용하기 시작합니다. 예를 들어 이 리더 인스턴스는 `delete-db-instance` 명령에 의해 삭제됩니다. RDS Proxy는 후속 쿼리를 위해 다른 리더 인스턴스로 전환합니다.

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

```
mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

mysql 세션이 계속 실행 중이지만, 다음 명령은 리더 엔드포인트가 연결된 리더 인스턴스를 삭제합니다.

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

mysql 세션의 쿼리는 다시 연결할 필요 없이 계속 작동합니다. RDS Proxy가 자동으로 다른 리더 DB 인스턴스로 전환합니다.

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

VPC 간에 Aurora 데이터베이스 액세스

기본적으로 Aurora 기술 스택의 구성 요소는 모두 동일한 Amazon VPC에 있습니다. 예를 들어 Amazon EC2 인스턴스에서 실행되는 애플리케이션이 Aurora DB 클러스터에 연결한다고 가정합니다. 이 경우 애플리케이션 서버와 데이터베이스가 모두 동일한 VPC 내에 있어야 합니다.

RDS 프록시를 사용하면 EC2 인스턴스와 같은 다른 VPC의 리소스에서 한 VPC에 있는 Aurora DB 클러스터에 대한 액세스를 설정할 수 있습니다. 예를 들어 조직에 동일한 데이터베이스 리소스에 액세스하는 애플리케이션이 여러 개 있을 수 있습니다. 각 애플리케이션은 자체 VPC에 있을 수 있습니다.

VPC 간 액세스를 활성화하려면 프록시를 위한 새 엔드포인트를 생성합니다. 프록시 자체는 Aurora DB 클러스터와 동일한 VPC에 상주합니다. 하지만 VPC 간 엔드포인트는 EC2 인스턴스 등의 다른 리소스와 함께 다른 VPC에 상주합니다. VPC 간 엔드포인트는 EC2 및 기타 리소스와 동일한 VPC의 서브넷 및 보안 그룹과 연결됩니다. 이러한 연결을 사용하면 VPC 제한으로 인해 데이터베이스에 액세스할 수 없는 애플리케이션이 엔드포인트에 연결할 수 있습니다.

다음 단계에서는 RDS Proxy를 통해 VPC 간 엔드포인트를 생성하고 액세스하는 방법을 설명합니다.

1. VPC 2개를 생성하거나 Aurora 작업에 이미 사용 중인 VPC 2개를 선택합니다. 각 VPC에는 인터넷 게이트웨이, 라우팅 테이블, 서브넷 및 보안 그룹과 같은 연결된 자체 네트워크 리소스가 있어야 합니다. VPC가 하나만 있는 경우 Amazon Aurora 시작하기에서 Aurora를 성공적으로 사용하기 위해 다른 VPC를 설정하는 단계를 참조하세요. 또한 Amazon EC2 콘솔에서 기존 VPC를 조사하여 함께 연결할 리소스의 종류를 확인할 수 있습니다.
2. 연결할 Aurora DB 클러스터에 연결된 DB 프록시를 생성합니다. Amazon Aurora용 프록시 만들기의 절차를 따르십시오.
3. RDS 콘솔에서 프록시의 [세부 정보(Details)] 페이지에 있는 [프록시 엔드포인트(Proxy endpoints)] 섹션에서 [엔드포인트 생성(Create endpoint)]을 선택합니다. 프록시 엔드포인트 생성의 절차를 따르십시오.
4. VPC 간 엔드포인트를 읽기/쓰기와 읽기 전용 중 무엇으로 설정할지 선택합니다.
5. Aurora DB 클러스터와 동일한 VPC인 기본값을 사용하지 않고, 다른 VPC를 선택합니다. 이 VPC는 프록시가 상주하는 VPC와 동일한 AWS 리전에 있어야 합니다.
6. 이제 Aurora DB 클러스터와 동일한 VPC의 서브넷 및 보안 그룹에 대한 기본값을 사용하지 않고 새로 선택합니다. 선택한 VPC의 서브넷과 보안 그룹을 기준으로 선택합니다.
7. Secrets Manager 보안 정보에 대한 설정은 변경할 필요가 없습니다. 각 엔드포인트가 속한 VPC에 관계없이 프록시의 모든 엔드포인트에 대해 동일한 자격 증명이 작동합니다.
8. 새 엔드포인트가 [사용 가능(Available)] 상태로 바뀔 때까지 기다립니다.
9. 전체 엔드포인트 이름을 기록해 둡니다. 이 값은 데이터베이스 애플리케이션의 연결 문자열의 일부로 제공하는 `Region_name.rds.amazonaws.com`으로 끝납니다.
10. 엔드포인트와 동일한 VPC에 있는 리소스에서 새 엔드포인트에 액세스합니다. 이 프로세스를 테스트하는 간단한 방법은 이 VPC에 새 EC2 인스턴스를 생성하는 것입니다. 그런 다음 EC2 인스턴스에 로그인하고 `mysql` 또는 `psql` 명령을 실행하여 연결 문자열의 엔드포인트 값을 통해 연결합니다.

프록시 엔드포인트 생성

프록시 엔드포인트를 생성하려면 다음 지침을 따르세요.

콘솔

프록시 엔드포인트를 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 새 엔드포인트를 생성하려는 프록시의 이름을 클릭합니다.

   해당 프록시에 대한 세부 정보 페이지가 나타납니다.
4. [프록시 엔드포인트(Proxy endpoints)] 섹션에서 [프록시 엔드포인트 생성(Create proxy endpoint)]을 선택합니다.

   [프록시 엔드포인트 생성(Create proxy endpoint)] 창이 나타납니다.
5. [프록시 엔드포인트 이름(Proxy endpoint name)]에 원하는 알기 쉬운 이름을 입력합니다.
6. [대상 역할(Target role)]에서 엔드포인트를 읽기/쓰기와 읽기 전용 중 무엇으로 설정할지 선택합니다.

   읽기/쓰기 엔드포인트를 사용하는 연결은 데이터 정의 언어(DDL) 문, 데이터 조작 언어(DML) 문, 쿼리 등 모든 종류의 작업을 수행할 수 있습니다. 이러한 엔드포인트는 항상 Aurora 클러스터의 기본 인스턴스에 연결합니다. 애플리케이션에서 단일 엔드포인트만 사용하는 경우 일반 데이터베이스 작업에 읽기/쓰기 엔드포인트를 사용할 수 있습니다. 관리 작업, OLTP(온라인 트랜잭션 처리) 애플리케이션 및 ETL(추출 변환 로드) 작업에 읽기/쓰기 엔드포인트를 사용할 수도 있습니다.

   읽기 전용 엔드포인트를 사용하는 연결은 쿼리만 수행할 수 있습니다. Aurora 클러스터에 여러 리더 인스턴스가 있는 경우, RDS 프록시는 엔드포인트에 대한 각 연결마다 서로 다른 리더 인스턴스를 사용할 수 있습니다. 이렇게 하면 쿼리 집약적인 애플리케이션이 Aurora의 클러스터링 기능을 활용할 수 있습니다. 더 많은 리더 DB 인스턴스를 추가하여 클러스터의 쿼리 용량을 늘릴 수 있습니다. 이러한 읽기 전용 연결은 클러스터의 기본 인스턴스에 오버헤드를 발생시키지 않습니다. 이렇게 하면 보고 및 분석 쿼리가 OLTP 애플리케이션의 쓰기 작업 속도를 저하시키지 않습니다.
7. Virtual Private Cloud(VPC)의 경우, 일반적으로 프록시 또는 연결된 데이터베이스에 액세스하는 데 사용하는 것과 동일한 EC2 인스턴스 또는 기타 리소스에서 엔드포인트에 액세스하려면 기본값을 선택합니다. 이 프록시에 대해 VPC 간 액세스를 설정하려면 기본값 이외의 VPC를 선택합니다. VPC 간 액세스에 대한 자세한 내용은 [VPC 간에 Aurora 데이터베이스 액세스](#) 섹션을 참조하세요.

8. [서브넷(Subnets)]의 경우 RDS Proxy는 기본적으로 연결된 프록시와 동일한 서브넷을 채웁니다. VPC의 주소 범위 중 일부만 연결할 수 있도록 엔드포인트에 대한 액세스를 제한하려면 하나 이상의 서브넷을 제거합니다.
9. VPC 보안 그룹의 경우 기존 보안 그룹을 선택하거나 새 보안 그룹을 생성할 수 있습니다. RDS Proxy는 기본적으로 연결된 프록시와 동일한 보안 그룹 또는 그룹을 채웁니다. 프록시의 인바운드 및 아웃바운드 규칙이 이 엔드포인트에 적합한 경우, 기본 선택 항목을 그대로 두면 됩니다.

   새 보안 그룹을 생성하도록 선택한 경우 이 페이지에서 보안 그룹의 이름을 지정합니다. 그런 다음 EC2 콘솔에서 보안 그룹 설정을 편집합니다.
10. [프록시 엔드포인트 생성(Create proxy endpoint)]을 선택합니다.

AWS CLI

프록시 엔드포인트를 생성하려면 AWS CLI create-db-proxy-endpoint 명령을 사용합니다.

다음 필수 파라미터를 포함합니다.

- `--db-proxy-name` *value*
- `--db-proxy-endpoint-name` *value*
- `--vpc-subnet-ids` *list_of_ids*. 서브넷 ID를 공백으로 구분합니다. VPC 자체의 ID는 지정하지 않습니다.

다음과 같은 선택적 파라미터도 포함할 수 있습니다.

- `--target-role { READ_WRITE | READ_ONLY }`이 파라미터의 기본값은 입니다.`READ_WRITE` `READ_ONLY` 값은 하나 이상의 리더 DB 인스턴스가 포함되어 있는 Aurora 프로비저닝된 클러스터에만 영향을 미칩니다. 프록시가 라이터 DB 인스턴스만 포함된 Aurora 클러스터에 연결되어 있는 경우 `READ_ONLY`를 지정할 수 없습니다. Aurora 클러스터에서 읽기 전용 엔드포인트의 용도에 대한 자세한 내용은 Aurora 클러스터에 리더 엔드포인트 사용 섹션을 참조하세요.
- `--vpc-security-group-ids` *value*. 보안 그룹 ID를 공백으로 구분합니다. 이 파라미터를 생략하면 RDS Proxy가 VPC에 기본 보안 그룹을 사용합니다. RDS Proxy는 `--vpc-subnet-ids` 파라미터에 대해 지정한 서브넷 ID를 기반으로 VPC를 확인합니다.

Example

다음 예에서는 `my-endpoint`라는 프록시 엔드포인트를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id ]
```

Windows의 경우:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id
```

RDS API

프록시 엔드포인트를 생성하려면 RDS API CreateDBProxyEndpoint 작업을 사용합니다.

프록시 엔드포인트 보기

기존 프록시 엔드포인트를 보려면 다음 지침을 따르세요.

콘솔

프록시 엔드포인트에 대한 세부 정보를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 목록에서 엔드포인트를 보려는 프록시를 선택합니다. 프록시 이름을 클릭하여 세부 정보 페이지를 봅니다.
4. [프록시 엔드포인트(Proxy endpoints)] 섹션에서 보려는 엔드포인트를 선택합니다. 세부 정보 페이지를 보려면 이름을 클릭합니다.
5. 값을 확인하고 싶은 파라미터를 조사합니다. 다음과 같은 속성을 확인할 수 있습니다.

- 엔드포인트가 읽기/쓰기인지 아니면 읽기 전용인지 여부.
- 데이터베이스 연결 문자열에 사용하는 엔드포인트 주소.
- 엔드포인트와 연결된 VPC, 서브넷 및 보안 그룹.

AWS CLI

하나 이상의 프록시 엔드포인트를 보려면 AWS CLI [describe-db-proxy-endpoints](#) 명령을 사용합니다.

다음과 같은 선택적 파라미터를 포함할 수 있습니다.

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

다음 예에서는 `my-endpoint` 프록시 엔드포인트를 설명합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

Windows의 경우:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

하나 이상의 프록시 엔드포인트를 설명하려면 RDS API [DescribeDBProxyEndpoints](#) 작업을 사용합니다.

프록시 엔드포인트 수정

프록시 엔드포인트를 수정하려면 다음 지침을 따르세요.

콘솔

하나 이상의 프록시 엔드포인트를 수정하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Proxies(프록시)를 선택합니다.
3. 목록에서 엔드포인트를 수정할 프록시를 선택합니다. 프록시 이름을 클릭하여 세부 정보 페이지를 봅니다.
4. [프록시 엔드포인트(Proxy endpoints)] 섹션에서 수정할 엔드포인트를 선택합니다. 목록에서 선택하거나 이름을 클릭하여 세부 정보 페이지를 볼 수 있습니다.
5. 프록시 세부 정보 페이지의 [프록시 엔드포인트(Proxy endpoints)] 섹션에서 [편집(Edit)]을 선택합니다. 또는 프록시 엔드포인트 세부 정보 페이지의 작업에서 편집을 선택합니다.
6. 수정할 파라미터의 값을 변경합니다.
7. Save changes(변경 사항 저장)를 선택합니다.

AWS CLI

프록시 엔드포인트를 수정하려면 AWS CLI modify-db-proxy-endpoint 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-proxy-endpoint-name`

다음 파라미터 중 하나 이상을 사용하여 엔드포인트 속성에 대한 변경 사항을 지정합니다.

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. 보안 그룹 ID를 공백으로 구분합니다.

다음 예에서는 `my-endpoint` 프록시 엔드포인트의 이름을 `new-endpoint-name`으로 변경합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

Windows의 경우:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

프록시 엔드포인트를 수정하려면 RDS API ModifyDBProxyEndpoint 작업을 사용합니다.

프록시 엔드포인트 삭제

프록시의 엔드포인트를 삭제하려면 다음 지침을 따르세요.

> **Note**
> RDS 프록시가 각 프록시에 대해 자동으로 생성하는 기본 프록시 엔드포인트는 삭제할 수 없습니다.
> 프록시를 삭제하면 RDS Proxy가 모든 연결된 엔드포인트를 자동으로 삭제합니다.

콘솔

AWS Management Console을 사용하여 프록시 엔드포인트를 삭제하려면

1. 탐색 창에서 Proxies(프록시)를 선택합니다.
2. 목록에서 엔드포인트를 삭제할 프록시를 선택합니다. 프록시 이름을 클릭하여 세부 정보 페이지를 봅니다.
3. [프록시 엔드포인트(Proxy endpoints)] 섹션에서 삭제할 엔드포인트를 선택합니다. 목록에서 하나 이상의 엔드포인트를 선택하거나, 단일 엔드포인트의 이름을 클릭하여 세부 정보 페이지를 볼 수 있습니다.
4. 프록시 세부 정보 페이지의 [프록시 엔드포인트(Proxy endpoints)] 섹션에서 [삭제(Delete)]를 선택합니다. 또는 프록시 엔드포인트 세부 정보 페이지의 작업에서 삭제를 선택합니다.

AWS CLI

프록시 엔드포인트를 삭제하려면 다음 필수 파라미터를 사용하여 delete-db-proxy-endpoint 명령을 실행합니다.

- `--db-proxy-endpoint-name`

다음 명령은 `my-endpoint`라는 프록시 엔드포인트를 삭제합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

Windows의 경우:

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

RDS API를 사용하여 프록시 엔드포인트를 삭제하려면 DeleteDBProxyEndpoint 작업을 실행합니다. `DBProxyEndpointName` 파라미터에 프록시 엔드포인트의 이름을 지정합니다.

Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링

Amazon CloudWatch를 사용하여 RDS Proxy를 모니터링할 수 있습니다. CloudWatch는 프록시에서 원시 데이터를 수집하여 읽기 쉬운 실시간 지표로 처리합니다. CloudWatch 콘솔에서 이러한 지표를 찾으려면 지표를 선택한 다음 RDS, Per-Proxy Metrics(프록시별 지표)를 차례로 선택합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 지표 사용을 참조하세요.

> **Note**
>
> RDS는 프록시와 연결된 각 기본 Amazon EC2 인스턴스에 대해 이러한 지표를 게시합니다. 하나의 프록시가 둘 이상의 EC2 인스턴스에서 사용될 수 있습니다. CloudWatch 통계를 사용하여 연관된 모든 인스턴스에서 프록시 값을 집계합니다.
> 이러한 지표 중 일부는 프록시에 의한 첫 번째 연결이 성공하기 전까지 표시되지 않을 수 있습니다.

RDS Proxy 로그에서 각 항목에는 연결된 프록시 엔드포인트의 이름이 접두사로 붙습니다. 이 이름은 사용자 정의 엔드포인트에 대해 지정한 이름이거나 읽기/쓰기 요청을 수행하는 프록시의 기본 엔드포인트에 대한 특수 이름(`default`)일 수 있습니다.

모든 RDS Proxy 지표는 proxy 그룹에 있습니다.

각 프록시 엔드포인트에는 자체 CloudWatch 지표가 있습니다. 각 프록시 엔드포인트의 사용을 독립적으로 모니터링할 수 있습니다. 프록시 엔드포인트에 대한 자세한 내용은 Amazon RDS 프록시 엔드포인트 작업 섹션을 참조하세요.

다음 측정 기준 세트 중 하나를 사용하여 각 지표의 값을 집계할 수 있습니다. 예를 들어 ProxyName 측정 기준 세트를 사용하여 특정 프록시의 모든 트래픽을 분석할 수 있습니다. 다른 차원 세트를 사용하면 여러 가지 방식으로 지표를 분할할 수 있습니다. 각 프록시의 여러 엔드포인트 또는 대상 데이터베이스나, 읽기/쓰기 및 읽기 전용 트래픽을 기준으로 지표를 분할할 수 있습니다.

- 차원 세트 1: ProxyName
- 차원 세트 2: ProxyName, EndpointName
- 차원 세트 3: ProxyName, TargetGroup, Target
- 차원 세트 4: ProxyName, TargetGroup, TargetRole

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
AvailabilityPercentage	차원이 나타내는 역할에서 대상 그룹을 사용할 수 있었던 시간의 비율입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 4
ClientConnections	현재 클라이언트 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 2
ClientConnectionsClosed	닫은 클라이언트 연결 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
ClientConnectionsInSetup	현재 열려 있지만 설정을 완료하지 않은 클라이언트 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.	1분	Dimension set 1, Dimension set 2
ClientConnectionsNoTLS	TLS(전송 계층 보안)가 없는 현재 클라이언트 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 2
ClientConnectionsReceived	수신된 클라이언트 연결 요청 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
ClientConnectionsSetupFailedAuth	잘못 구성된 인증 또는 TLS로 인해 실패한 클라이언트 연결 시도 횟수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
ClientConnectionsSetupSucceeded	TLS를 사용하거나 사용하지 않는 인증 메커니즘을 통해 성공적으로 설정된 클라이언트 연결 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
ClientConnectionsTLS	현재 TLS를 사용하는 클라이언트 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 2
DatabaseConnectionRequests	데이터베이스 연결을 생성하기 위한 요청 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionRequestsWithTLS	TLS를 사용하는 데이터베이스 연결을 생성하기 위한 요청 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnections	현재 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
`DatabaseConnectionsBorrowLatency`	모니터링되는 프록시가 데이터베이스 연결을 가져오는 데 걸리는 시간(마이크로초)입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
`DatabaseConnectionsCurrentlyBorrowed`	현재 대여 상태인 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsCurrentlyInTransaction`	현재 트랜잭션의 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsCurrentlySessionPinned`	세션 상태를 변경하는 클라이언트 요청의 작업으로 인해 현재 고정된 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
DatabaseConnectionsSetupFailed	실패한 데이터베이스 연결 요청 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupSucceeded	TLS를 사용하거나 사용하지 않고 성공적으로 설정된 데이터베이스 연결 수입니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsWithTLS	현재 TLS를 사용하는 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4
MaxDatabaseConnectionsAllowed	허용되는 최대 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분	Dimension set 1, Dimension set 3, Dimension set 4
QueryDatabaseResponseLatency	데이터베이스가 쿼리에 응답하는 데 걸린 시간(마이크로초)입니다. 이 지표에 가장 유용한 통계는 Average입니다.	1분 이상	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4

측정치	설명	유효 기간	CloudWatch 측정 기준 세트
QueryRequests	수신된 쿼리 수입니다. 여러 문을 포함하는 쿼리는 하나의 쿼리로 계산됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
QueryRequestsNoTLS	TLS 이외의 연결에서 받은 쿼리 수입니다. 여러 문을 포함하는 쿼리는 하나의 쿼리로 계산됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
QueryRequestsTLS	TLS 연결에서 받은 쿼리 수입니다. 여러 문을 포함하는 쿼리는 하나의 쿼리로 계산됩니다. 이 지표에 가장 유용한 통계는 Sum입니다.	1분 이상	Dimension set 1, Dimension set 2
QueryResponseLatency	쿼리 요청을 받는 시간과 프록시가 응답하는 시간 사이의 시간(마이크로 초)입니다. 이 지표에 가장 유용한 통계는 Average입니다.	1분 이상	Dimension set 1, Dimension set 2

AWS Management Console의 CloudWatch 아래에서 RDS Proxy 활동 로그를 찾을 수 있습니다. 각 프록시는 로그 그룹 페이지에 항목이 있습니다.

⚠ Important

이러한 로그는 프로그래밍 방식 액세스가 아니라 사용자가 문제 해결을 위해 사용하기 위한 로그입니다. 로그의 형식과 내용은 변경될 수 있습니다.
특히 이전 로그에는 각 요청의 엔드포인트를 나타내는 접두사가 포함되어 있지 않습니다. 최신 로그에서는 각 항목에 연결된 프록시 엔드포인트의 이름이 접두사로 붙습니다. 이 이름은 사용자 정의 엔드포인트에 대해 지정한 이름이거나, 프록시의 기본 엔드포인트를 사용한 요청을 위한 `default`라는 특수한 이름일 수 있습니다.

RDS 프록시 이벤트 작업

이벤트는 서비스형 소프트웨어(SaaS) 파트너로부터 AWS 환경, 서비스 또는 애플리케이션과 같은 환경의 변화를 나타냅니다. 또는 사용자 지정 애플리케이션이나 서비스 중 하나일 수 있습니다. 예를 들어 Amazon Aurora에서는 RDS 프록시를 생성하거나 수정할 때 이벤트를 생성합니다. Amazon Aurora는 거의 실시간으로 Amazon EventBridge에 이벤트를 전송합니다. 다음에서 구독할 수 있는 RDS 프록시 이벤트 목록과 RDS Proxy 이벤트의 예를 찾을 수 있습니다.

이벤트 작업에 대한 자세한 내용은 다음을 참조하세요.

- AWS Management Console, AWS CLI 또는 RDS API를 사용하여 이벤트를 보는 방법에 대한 지침은 Amazon RDS 이벤트 보기 섹션을 참조하세요.
- EventBridge로 이벤트를 전송하도록 Amazon Aurora를 구성하는 방법을 알아보려면 Amazon Aurora 이벤트에서 트리거되는 규칙 생성 섹션을 참조하세요.

RDS 프록시 이벤트

다음 표에서는 RDS 프록시가 소스 유형일 때 이벤트 범주와 이벤트 목록을 보여줍니다.

범주	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0204	RDS가 DB 프록시(*##*)를 수정했습니다.	None
구성 변경	RDS-EVENT-0207	RDS가 DB 프록시(*##*)의 엔드포인트를 수정했습니다.	None

범주	RDS 이벤트 ID	메시지	참고
구성 변경	RDS-EVENT-0213	RDS가 DB 인스턴스 추가를 감지하여 DB 프록시(##)의 대상 그룹에 자동으로 추가했습니다.	None
구성 변경	RDS-EVENT-0214	RDS가 DB 인스턴스(##)의 삭제를 감지하여 DB 프록시(##)의 대상 그룹(##)에서 자동으로 제거했습니다.	None
구성 변경	RDS-EVENT-0215	RDS가 DB 클러스터(##)의 삭제를 감지하여 DB 프록시(##)의 대상 그룹(##)에서 자동으로 제거했습니다.	None
생성	RDS-EVENT-0203	RDS가 DB 프록시(##)를 생성했습니다.	None
생성	RDS-EVENT-0206	RDS가 DB 프록시(##)에 대한 엔드포인트(##)을 생성했습니다.	None
삭제	RDS-EVENT-0205	RDS가 DB 프록시(##)를 삭제했습니다.	None
삭제	RDS-EVENT-0208	RDS가 DB 프록시(##)에 대한 엔드포인트(##)을 삭제했습니다.	None

범주	RDS 이벤트 ID	메시지	참고
실패	RDS-EVENT-0243	서브넷(##)에서 사용할 수 있는 IP 주소가 충분하지 않기 때문에 RDS가 프록시(##)의 용량을 프로비저닝하지 못했습니다. 이러한 문제를 해결하려면 RDS 프록시 설명서의 권장 사항에 따라 서브넷에 최소한의 미사용 IP 주소 개수가 있는지 확인하세요.	권장 인스턴스 클래스 수를 확인하려면 IP 주소 용량 계획 섹션을 참조하세요.
실패	RDS-EVENT-0275	RDS가 DB 프록시 ##에 대한 일부 연결을 제한했습니다. 클라이언트에서 프록시로의 동시 연결 요청 수가 제한을 초과했습니다.	None

다음은 JSON 형식의 RDS 프록시 이벤트의 예입니다. 이벤트는 RDS가 `my-rds-proxy`라는 RDS 프록시의 `my-endpoint`라는 엔드포인트를 수정했음을 보여줍니다. 이벤트 ID는 RDS-EVENT-0207입니다.


```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
```

```
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

RDS 프록시 명령줄 예제

연결 명령과 SQL 문의 조합이 어떻게 RDS Proxy와 상호 작용하는지 보려면 다음 예제를 살펴보십시오.

예시

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example 장애 조치 전반에 걸쳐 MySQL 데이터베이스에 대한 연결 유지

이 MySQL 예에서는 장애 조치 중에 어떻게 열린 연결이 계속 작동하는지 보여줍니다. 데이터베이스를 재부팅하거나 데이터베이스가 문제로 인해 사용할 수 없게 되는 경우를 예로 들 수 있습니다. 이 예제에서는 `the-proxy`라는 프록시와 DB 인스턴스 `instance-8898` 및 `instance-9814`가 있는 Aurora DB 클러스터를 사용합니다. Linux 명령줄에서 `failover-db-cluster` 명령을 실행하면 프록시와 연결된 라이터 인스턴스가 다른 DB 인스턴스로 변경됩니다. 연결이 열려 있는 상태에서 프록시와 연결된 DB 인스턴스가 변경되는 것을 확인할 수 있습니다.

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
```

```
[1]+  Stopped                  mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                  mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example Aurora DB 클러스터에 대한 max_connections 설정 조정

이 예제에서는 Aurora MySQL DB 클러스터의 `max_connections` 설정을 조정하는 방법을 보여 줍니다. 이렇게 하려면 MySQL 5.7과 호환되는 클러스터의 기본 파라미터 설정을 기반으로 자체 DB 클러스터 파라미터 그룹을 생성합니다. `max_connections` 설정에 값을 지정하여 기본값을 설정하는 공식을 재지정합니다. DB 클러스터 파라미터 그룹을 자체 DB 클러스터와 연결합니다.

```
export REGION=us-east-1
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
```

```
--output text | grep "^max_connections"
```

RDS 프록시 문제 해결

다음에서 몇 가지 일반적인 RDS 프록시 문제에 대한 문제 해결 아이디어와 RDS 프록시에 대한 CloudWatch 로그에 대한 정보를 찾을 수 있습니다.

RDS 프록시 로그에서 각 항목에는 연결된 프록시 엔드포인트의 이름이 접두사로 붙습니다. 이 이름은 사용자 정의 엔드포인트에 지정한 이름일 수 있습니다. 아니면 읽기/쓰기 요청을 수행하는 프록시의 기본 엔드포인트에 대한 특수 이름(`default`)일 수 있습니다. 프록시 엔드포인트에 대한 자세한 내용은 Amazon RDS 프록시 엔드포인트 작업 섹션을 참조하세요.

주제

프록시에 대한 연결 확인

다음 명령을 사용하여 연결 내의 모든 구성 요소(예: 프록시, 데이터베이스, 컴퓨팅 인스턴스)가 서로 통신할 수 있는지 확인할 수 있습니다.

describe-db-proxies 명령을 사용하여 프록시 자체를 검사합니다. 또한 describe-db-proxy-target-groups 명령을 사용하여 연결된 대상 그룹을 검사합니다. 대상의 세부 정보가 프록시와 연결하려는 Aurora 클러스터와 일치하는지 확인합니다. 다음과 같은 명령을 사용합니다.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

프록시가 기본 데이터베이스에 연결할 수 있는지 확인하려면 describe-db-proxy-targets 명령을 사용하여 대상 그룹에 지정된 대상을 검사합니다. 다음과 같은 명령을 사용합니다.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

describe-db-proxy-targets 명령의 출력에는 TargetHealth 필드가 포함됩니다. State 내부의 Reason, Description 및 TargetHealth 필드를 검사하여 프록시가 기본 DB 인스턴스와 통신할 수 있는지 확인할 수 있습니다.

- 프록시가 DB 인스턴스에 연결할 수 있는 State의 AVAILABLE 값입니다.
- State의 UNAVAILABLE 값은 임시 또는 영구 연결 문제를 나타냅니다. 이 경우 Reason 및 Description 필드를 검사합니다. 예를 들어 Reason의 값이 PENDING_PROXY_CAPACITY인 경우 프록시가 조정 작업을 완료한 후 다시 연결해 보십시오. Reason의 값이 UNREACHABLE, CONNECTION_FAILED 또는 AUTH_FAILURE인 경우 Description 필드의 설명을 사용하여 문제를 진단합니다.
- State 또는 REGISTERING로 변경하기 전에 AVAILABLE 필드의 값이 잠시 동안 UNAVAILABLE일 수 있습니다.

다음 Netcat 명령(nc)이 성공하면 로그인한 EC2 인스턴스 또는 다른 시스템에서 프록시 엔드포인트에 액세스할 수 있습니다. 이 명령은 프록시 및 연결된 데이터베이스와 동일한 VPC에 있지 않은 경우 실패를 보고합니다. 동일한 VPC에 있지 않아도 데이터베이스에 직접 로그인할 수 있습니다. 그러나 동일한 VPC에 있지 않으면 프록시에 로그인할 수는 없습니다.

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

다음 명령을 사용하여 EC2 인스턴스에 필수 속성이 있는지 확인할 수 있습니다. 특히 EC2 인스턴스의 VPC는 프록시가 연결하는 RDS DB 인스턴스Aurora 클러스터의 VPC와 동일해야 합니다.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

프록시에 사용되는 Secrets Manager 비밀을 검사합니다.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

get-secret-value에서 표시한 SecretString 필드가 username 및 password 필드를 포함하는 JSON 문자열로 인코딩되어 있는지 확인합니다. 다음 예제는 SecretString 필드의 형식을 보여 줍니다.

```
{
  "ARN": "some_arn",
```

```
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

일반적인 문제 및 해결 방법

이 섹션에서는 RDS 프록시를 사용할 때 발생하는 몇 가지 일반적인 문제와 잠재적 해결 방법에 대해 설명합니다.

`aws rds describe-db-proxy-targets` CLI 명령을 실행한 후 `TargetHealth` 설명에 `Proxy does not have any registered credentials`라고 표시되면 다음을 확인하세요.

- 사용자가 프록시에 액세스할 수 있도록 등록된 보안 인증 정보가 있습니다.
- 프록시에서 사용하는 Secrets Manager 보안 암호에 액세스하는 IAM 역할은 유효합니다.

DB 프록시를 생성하거나 DB 프록시에 연결하는 동안 다음과 같은 RDS 이벤트가 발생할 수 있습니다.

범주	RDS 이벤트 ID	설명
실패	RDS-EVENT-0243	서브넷에서 사용할 수 있는 IP 주소가 충분하지 않기 때문에 RDS가 프록시 용량을 프로비저닝할 수 없습니다. 이러한 문제를 해결하려면 서브넷에 최소한의 미사용 IP 주소가 있는지 확인합니다. 권장 인스턴스 클래스 수를 확인하려면 IP 주소 용량 계획 섹션을 참조하세요.
실패	RDS-EVENT-0275	RDS가 DB 프록시 *##*에 대한 일부 연결을 제한했습니다. 클라이언트에서 프록시로의 동시 연결 요청 수가 제한을 초과했습니다.

새 프록시를 생성하거나 프록시에 연결하는 동안 다음과 같은 문제가 발생할 수 있습니다.

오류	원인 또는 해결 방법
`403: The security token included in the request is invalid`	새 IAM 역할을 생성하는 대신 기존 IAM 역할을 선택합니다.

RDS for MySQL 관련 RDS 프록시 문제 해결

MySQL 프록시에 연결하는 동안 다음과 같은 문제가 발생할 수 있습니다.

오류	원인 또는 해결 방법
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	클라이언트에서 프록시로의 연결 요청 속도가 제한을 초과했습니다.
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	클라이언트에서 프록시로의 IAM 인증을 사용하는 동시 요청 수가 제한을 초과했습니다.
`ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )`	클라이언트에서 프록시로의 동시 연결 요청 수가 제한을 초과했습니다.

오류	원인 또는 해결 방법
ERROR 1045 (28000): Access denied for user '*DB_USER*'@'%' (usi password: YES)	프록시에서 사용하는 Secrets Manager 비밀이 기존 데이터베이스 사용자의 사용자 이름 및 암호와 일치하지 않습니다. Secrets Manager 비밀의 자격 증명을 업데이트하거나 데이터베이스 사용자가 존재하고 비밀과 동일한 암호를 가지고 있는지 확인합니다.
ERROR 1105 (HY000): Unknown error	알 수 없는 오류가 발생했습니다.
ERROR 1231 (42000): Variable ''charact er_set_cl ient'' can't be set to the value of *value*	`character_set_client` 파라미터에 설정된 값이 유효하지 않습니다. 예를 들어 ucs2 값은 MySQL 서버를 충돌시킬 수 있으므로 유효하지 않습니다.
ERROR 3159 (HY000): This RDS Proxy requires TLS connections.	프록시에서 전송 계층 보안 필요 설정을 활성화했지만 연결에 MySQL 클라이언트의 파라미터 `ssl-mode=DISABLED` 가 포함되었습니다. 다음 중 하나를 수행하십시오. • 프록시에 대해 전송 계층 보안 필요 설정을 비활성화합니다. • PostgreSQL 클라이언트에서 `ssl-mode=REQUIRED` 의 최소 설정을 사용하여 데이터베이스에 연결합니다.
ERROR 2026 (HY000): SSL connection error: Internal Server *Error*	프록시에 대한 TLS 핸드셰이크가 실패했습니다. 몇 가지 가능한 이유는 다음과 같습니다. • SSL은 필요하지만 서버는 이를 지원하지 않습니다. • 내부 서버 오류가 발생했습니다. • 잘못된 핸드셰이크가 발생했습니다.

오류	원인 또는 해결 방법
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	데이터베이스 연결을 얻기 위해 대기 중인 프록시 시간이 초과되었습니다. 몇 가지 가능한 이유는 다음과 같습니다. • 최대 연결에 도달했기 때문에 프록시가 데이터베이스 연결을 설정할 수 없습니다. • 데이터베이스를 사용할 수 없으므로 프록시가 데이터베이스 연결을 설정할 수 없습니다.

RDS for PostgreSQL 관련 RDS 프록시 문제 해결

PostgreSQL 프록시에 연결하는 동안 다음과 같은 문제가 발생할 수 있습니다.

오류	원인	솔루션
`ERROR 28000: IAM authentication is allowed only with SSL connections.`	사용자가 PostgreSQL 클라이언트에서 `sslmode=disable` 설정이 있는 IAM 인증을 사용하여 데이터베이스에 연결하려고 했습니다.	사용자는 PostgreSQL 클라이언트에서 `sslmode=require` 의 최소 설정을 사용하여 데이터베이스에 연결해야 합니다. 자세한 내용은 PostgreSQL SSL 지원 설명서를 참조하십시오.
`ERROR 28000: This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.`	이 역할에 대한 Secrets Manager 암호가 없습니다.	이 역할에 대한 Secrets Manager 암호를 추가하세요. 자세한 내용은 RDS 프록시에 대한 AWS Identity and Access Management(IAM) 정책 설정 섹션을 참조하세요.
`ERROR 28000: RDS supports only IAM, MD5, or SCRAM authentication.`	프록시에 연결하는 데 사용되는 데이터베이스 클라이언트가 프록시에서 현재 지원하지 않는 인증 메커니즘을 사용하고 있습니다.	IAM 인증을 사용하지 않는 경우 MD5 또는 SCRAM 암호 인증을 사용하세요.

오류	원인	솔루션
`ERROR 28000: A user name is missing from the connection startup packet. Provide a user name for this connection.`	연결을 설정하려고 할 때 프록시에 연결하는 데 사용되는 데이터베이스 클라이언트가 사용자 이름을 전송하지 않습니다.	선택한 PostgreSQL 클라이언트를 사용하여 프록시에 대한 연결을 설정할 때 사용자 이름을 정의해야 합니다.
`ERROR 28000: IAM is allowed only with SSL connections.`	클라이언트가 IAM 인증을 사용하여 연결을 시도했지만 SSL이 활성화되지 않았습니다.	PostgreSQL 클라이언트에서 SSL을 활성화하세요.
`ERROR 28000: This RDS Proxy requires TLS connections.`	사용자가 전송 계층 보안 필요 설정을 활성화했지만 PostgreSQL 클라이언트에서 `sslmode=disable` 로 연결하려고 시도했습니다.	이 오류를 해결하려면 다음 중 하나를 수행합니다. • 프록시의 전송 계층 보안 필요 옵션을 비활성화합니다. • PostgreSQL 클라이언트에서 `sslmode=allow` 의 최소 설정을 사용하여 데이터베이스에 연결합니다.
`ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.`	이 오류는 다음과 같은 이유 때문일 수 있습니다. • 클라이언트가 잘못된 IAM 사용자 이름을 제공했습니다. • 클라이언트가 사용자에게 잘못된 IAM 권한 부여 토큰을 제공했습니다. • 클라이언트가 필요한 권한이 없는 IAM 정책을 사용하고 있습니다. • 클라이언트가 사용자에게 만료된 IAM 권한 부여 토큰을 제공했습니다.	이 오류를 해결하려면 다음을 수행하십시오. 1. 제공된 IAM 사용자가 있는지 확인합니다. 2. IAM 권한 부여 토큰이 제공된 IAM 사용자에 속하는지 확인합니다. 3. IAM 정책에 RDS에 대한 적절한 권한이 있는지 확인합니다. 4. 사용된 IAM 권한 부여 토큰의 유효성을 확인합니다.

오류	원인	솔루션
`ERROR 28P01: The password that was provided for the role` *role_name* `is wrong.`	이 역할의 암호가 Secrets Manager 암호와 일치하지 않습니다.	Secrets Manager에서 이 역할의 암호를 확인하여 암호가 PostgreSQL 클라이언트에서 사용 중인 암호와 같은지 확인하세요.
`ERROR 28P01: The IAM authentication failed for the role` *role_name* `. Check the IAM token for this role and try again.`	IAM 인증에 사용되는 IAM 토큰에 문제가 있습니다.	새 인증 토큰을 생성하여 새 연결에 사용하세요.
`ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	프록시에 연결하는 데 사용되는 PostgreSQL 클라이언트는 3.0보다 오래된 프로토콜을 사용합니다.	3.0 메시징 프로토콜을 지원하는 최신 PostgreSQL 클라이언트를 사용하세요. PostgreSQL psql CLI를 사용하는 경우 7.4보다 크거나 같은 버전을 사용하세요.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	프록시에 연결하는 데 사용되는 PostgreSQL 클라이언트가 현재 RDS 프록시에서 지원되지 않는 스트리밍 복제 모드를 사용하려고 합니다.	연결하는 데 사용되는 PostgreSQL 클라이언트에서 스트리밍 복제 모드를 해제하세요.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option` *option_name* `.`	시작 메시지를 통해 프록시에 연결하는 데 사용되는 PostgreSQL 클라이언트가 현재 RDS 프록시에서 지원되지 않는 옵션을 요청하고 있습니다.	연결하는 데 사용되는 PostgreSQL 클라이언트에서 위의 메시지에서 지원되지 않는 것으로 표시되는 옵션을 해제하세요.

오류	원인	솔루션
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	클라이언트에서 프록시로의 IAM 인증을 사용하는 동시 요청 수가 제한을 초과했습니다.	PostgreSQL 클라이언트에서 IAM 인증을 사용하는 연결이 설정되는 속도를 줄이세요.
`ERROR 53300: The maximum number of client connections to the proxy exceeded` *number_value* `.`	클라이언트에서 프록시로의 동시 연결 요청 수가 제한을 초과했습니다.	PostgreSQL 클라이언트에서 이 RDS 프록시로의 활성 연결 수를 줄이세요.
`ERROR 53300: Rate of connection to proxy exceeded` *number_value* `.`	클라이언트에서 프록시로의 연결 요청 속도가 제한을 초과했습니다.	PostgreSQL 클라이언트로부터의 연결이 설정되는 속도를 줄이세요.
`ERROR XX000: Unknown error.`	알 수 없는 오류가 발생했습니다.	문제를 조사할 수 있도록 AWS Support에 문의해 주세요.

오류	원인	솔루션
`ERROR 08000: Timed-out waiting to acquire database connection.`	`ConnectionBorrowTimeout` 설정에 지정된 기간 내에 데이터베이스 연결을 획득하기 위해 대기 중인 프록시 제한 시간입니다. 몇 가지 가능한 이유는 다음과 같습니다. • 최대 연결에 도달했기 때문에 프록시가 데이터베이스 연결을 설정할 수 없습니다. • 데이터베이스를 사용할 수 없거나 데이터베이스에 대한 연결 설정이 구성된 `ConnectionBorrowTimeout` 보다 오래 걸리는 경우 프록시가 데이터베이스 연결을 설정할 수 없습니다.	가능한 해결책은 다음과 같습니다. • 프록시 연결을 고정하지 않도록 합니다. RDS 프록시 고정 방지을(를) 참조하세요. • `ConnectionBorrowTimeout` 및 `MaxConnectionsPercent` 설정을 검토합니다. RDS 프록시 연결 고려 사항을(를) 참조하세요. • 대상 가용성을 검토합니다. (Amazon CloudWatch를 사용한 RDS 프록시 지표 모니터링의 `AvailabilityPercentage` 섹션 참조)
`ERROR XX000: Request returned an error: `*`database_error`*` .`	프록시에서 설정된 데이터베이스 연결이 오류를 반환했습니다.	솔루션은 특정 데이터베이스 오류에 따라 다릅니다. 예를 들면 다음과 같습니다. `Request returned an error: database "your-database-name" does not exist` 이는 지정된 데이터베이스 이름이 데이터베이스 서버에 존재하지 않음을 의미합니다. 또는 데이터베이스 이름으로 사용되는 사용자 이름(데이터베이스 이름을 지정하지 않은 경우)이 서버에 존재하지 않는다는 의미입니다.

삭제된 **postgres** 데이터베이스 문제 해결

실수로 인스턴스에서 postgres 데이터베이스를 삭제한 경우 데이터베이스를 복원하여 인스턴스에 대한 연결을 복원해야 합니다. DB 인스턴스 내에서 다음 명령을 실행합니다.

```
CREATE DATABASE postgres;
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

RDS Proxy를 AWS CloudFormation에서 사용

RDS Proxy를 AWS CloudFormation에서 사용할 수 있습니다. 이렇게 하면 관련 리소스 그룹을 생성하는 데 도움이 됩니다. 이러한 그룹에는 새로 생성된 Aurora DB 클러스터에 연결할 수 있는 프록시가 포함될 수 있습니다. AWS CloudFormation의 RDS Proxy 지원에는 두 가지 새로운 레지스트리 유형인 DBProxy 및 DBProxyTargetGroup이 포함됩니다

다음 목록은 RDS Proxy에 대한 샘플 AWS CloudFormation 템플릿을 보여줍니다.

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

이 샘플의 리소스에 대한 자세한 내용은 DBProxy 및 DBProxyTargetGroup을 참조하세요.

AWS CloudFormation을 사용하여 생성할 수 있는 리소스에 대한 자세한 내용은 RDS 리소스 유형 참조를 확인하세요.

Aurora 글로벌 데이터베이스에 RDS 프록시 사용

Aurora 글로벌 데이터베이스는 다중 AWS 리전으로 확장할 수 있는 단일 데이터베이스로, 짧은 대기 시간의 글로벌 읽기가 가능하며 리전 규모의 가동 중단 발생 시 재해 복구를 제공합니다. DB 인스턴스는 단일 AWS 리전이 아니라 여러 리전 및 가용 영역에 의존하기 때문에, 배포한 내용에 자체 결함 용인을 제공합니다. 자세한 내용은 Amazon Aurora Global Database 사용 단원을 참조하십시오.

RDS 프록시는 Aurora 글로벌 데이터베이스의 어떤 DB 클러스터에도 사용할 수 있습니다. 이 기능들을 함께 사용하기 전에 다음 정보를 숙지해야 합니다.

> **Important**
>
> DB 클러스터가 쓰기 전달이 켜져 있는 글로벌 데이터베이스의 일부인 경우, 쓰기 전달에 할당된 할당량만큼 프록시의 `MaxConnectionsPercent` 값을 줄입니다. 쓰기 전달 할당량은 DB 클러스터 파라미터 `aurora_fwd_writer_max_connections_pct`에 설정됩니다. 쓰기 전달에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스에서 쓰기 전달 사용 단원을 참조하십시오.

글로벌 데이터베이스에서 RDS 프록시의 제한 사항

Aurora DB 클러스터에서 쓰기 전달이 켜져 있으면 RDS 프록시는 `aurora_replica_read_consistency` 변수의 `SESSION` 값을 지원하지 않습니다. 이 값을 설정하면 예상하지 못한 동작이 일어날 수 있습니다.

RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식

RDS 프록시 엔드포인트가 글로벌 데이터베이스에서 작동하는 방식을 이해하면 이 두 가지 기능을 통해 Aurora 데이터베이스를 사용하는 애플리케이션을 더 잘 관리할 수 있습니다.

글로벌 데이터베이스의 기본 클러스터를 등록된 대상으로 사용하는 프록시의 경우 프록시 엔드포인트는 모든 Aurora DB 클러스터와 동일한 방식으로 작동합니다. 프록시의 읽기/쓰기 엔드포인트는 모든 요청을 클러스터의 라이터 인스턴스로 보냅니다. 프록시의 읽기 전용 엔드포인트는 모든 요청을 리더 인스턴스로 보냅니다. 연결이 열려 있는 동안 리더를 사용할 수 없게 되면 RDS 프록시는 연결에 대한 후속 쿼리를 다른 리더 인스턴스로 리디렉션합니다. 보조 클러스터를 등록된 대상으로 사용하는 프

록시의 경우 프록시의 읽기 전용 엔드포인트로 전송된 요청도 리더 인스턴스로 전송됩니다. 클러스터에 라이터 인스턴스가 없기 때문에 읽기/쓰기 엔드포인트로 전송된 요청은 'The target group doesn't have any associated read/write instances' 오류와 함께 실패합니다.

글로벌 데이터베이스 전환 및 장애 조치 작업에는 기본 DB 클러스터와 보조 DB 클러스터 중 하나 간의 역할 전환이 포함됩니다. 선택한 보조 클러스터가 새 기본 클러스터가 되면 해당 리더 인스턴스 중 하나가 라이터로 승격됩니다. 이제 이 DB 인스턴스는 글로벌 클러스터의 새 라이터 인스턴스가 됩니다. 애플리케이션의 쓰기 작업을 새 기본 클러스터와 연결된 프록시의 적절한 읽기/쓰기 엔드포인트로 리디렉션해야 합니다. 이 프록시 엔드포인트는 기본 엔드포인트이거나 사용자 지정 읽기/쓰기 엔드포인트일 수 있습니다.

RDS 프록시는 읽기/쓰기 엔드포인트를 통해 모든 요청을 대기열에 넣고 사용 가능해지는 즉시 새 기본 클러스터의 라이터 인스턴스로 전송합니다. 이는 전환 또는 장애 조치 작업이 완료되었는지 여부에 관계없이 수행됩니다. 전환 또는 장애 조치 중에도 기존 기본 클러스터의 프록시 관련 기본 엔드포인트는 여전히 쓰기 작업을 수락합니다. 하지만 해당 클러스터가 보조 클러스터가 되는 즉시 모든 쓰기 작업이 실패합니다. 특정 글로벌 전환 또는 장애 조치 작업을 수행하는 방법과 시기를 알아보려면 다음 주제를 참조하세요.

- 글로벌 데이터베이스 전환 – [Amazon Aurora Global Database에서 전환 수행]
- 글로벌 데이터베이스 장애 조치 – [계획되지 않은 중단으로부터 Amazon Aurora Global Database 복구]

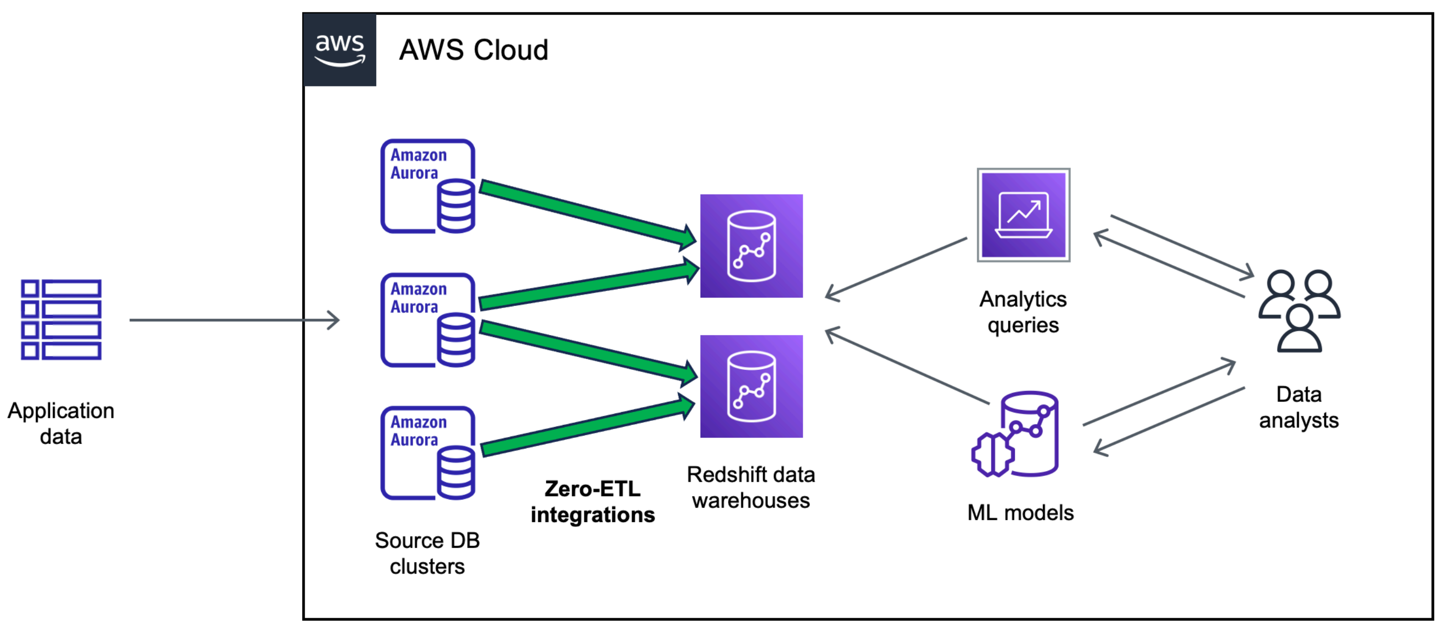
Aurora 제로 ETL 통합

Amazon Redshift 및 Amazon SageMaker와의 Aurora 제로 ETL 통합을 사용하면Aurora의 데이터를 사용해 거의 실시간에 가까운 분석 및 기계 학습(ML)을 수행할 수 있습니다. 이 솔루션은 트랜잭션 데이터가 Aurora DB 클러스터에 기록된 후 분석 대상에서 사용할 수 있도록 하기 위한 완전관리형 솔루션입니다. 추출, 전환, 적재(ETL)는 여러 소스의 데이터를 데이터 웨어하우스라는 대규모 중앙 리포지토리로 결합하는 프로세스입니다.

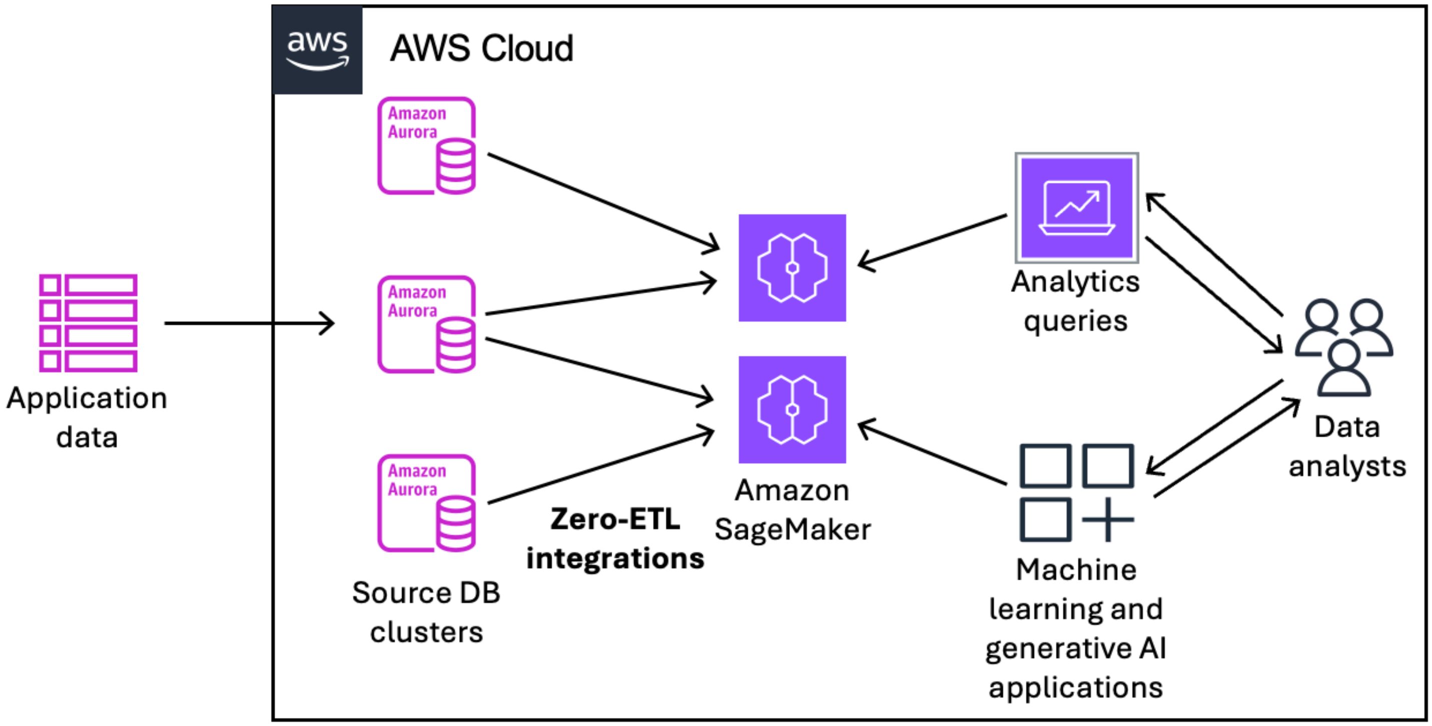
제로 ETL 통합을 통해 Aurora DB 클러스터의 데이터를 거의 실시간으로 Amazon Redshift 또는 Amazon SageMaker 레이크하우스에서 사용할 수 있습니다. 데이터가 대상 데이터 웨어하우스 또는 데이터 레이크에 저장되면 기계 학습, 구체화된 뷰, 데이터 공유, 여러 데이터 저장소 및 데이터 레이크에 대한 페더레이션 액세스, Amazon SageMaker AI, QuickSight 및 기타 AWS 서비스와의 통합과 같은 기본 제공 기능을 사용하여 분석, ML 및 AI 워크로드를 강화할 수 있습니다.

제로 ETL 통합을 만들려면 Aurora DB 클러스터를 소스로 지정하고 지원되는 데이터 웨어하우스 또는 레이크하우스를 대상으로 지정합니다. 통합은 소스 데이터베이스에서 대상 데이터 웨어하우스 또는 레이크하우스로 데이터를 복제합니다.

다음 다이어그램은 Amazon Redshift와의 제로 ETL 통합에서 이 기능을 보여줍니다.



다음 다이어그램은 Amazon SageMaker 레이크하우스와의 제로 ETL 통합에서 이 기능을 보여줍니다.



통합은 데이터 파이프라인의 상태를 모니터링하고 가능한 경우 문제로부터 복구합니다. 여러 Aurora DB 클러스터를 단일 대상 데이터 웨어하우스 또는 레이크하우스로 통합하여 여러 애플리케이션에 걸쳐 인사이트를 도출할 수 있습니다.

제로 ETL 통합 요금에 대한 자세한 내용은 Amazon Aurora 요금 및 Amazon Redshift 요금을 참조하세요.

주제

- Aurora 제로 ETL 통합 수정
- Aurora 제로 ETL 통합 삭제
- Amazon Redshift가 구성된 Aurora 제로 ETL 통합 문제 해결

이점

Aurora 제로 ETL 통합은 다음과 같은 주요 이점을 제공합니다.

- 여러 데이터 소스에서 총체적인 인사이트를 도출할 수 있도록 도와줍니다.
- 추출, 전환, 적재(ETL) 작업을 수행하는 복잡한 데이터 파이프라인을 구축하고 유지 관리할 필요가 없습니다. 제로 ETL 통합은 파이프라인을 프로비저닝하고 관리해 주므로 파이프라인을 구축하고 관리하는 데 따르는 어려움이 발생하지 않습니다.
- 운영 부담과 비용을 줄이고 애플리케이션 개선에 집중할 수 있습니다.
- 대상의 분석 및 ML 기능을 통해 트랜잭션 데이터 및 기타 데이터에서 인사이트를 도출하여 중요하고 시간에 민감한 이벤트에 효과적으로 대응할 수 있습니다.

주요 개념

제로 ETL 통합을 시작할 때는 다음 개념을 고려하세요.

통합

Aurora DB 클러스터에서 데이터 웨어하우스 또는 카탈로그로 트랜잭션 데이터 및 스키마를 자동으로 복제하는 완전관리형 데이터 파이프라인입니다.

소스 DB 클러스터

데이터가 복제되는 Aurora DB 클러스터입니다. 프로비저닝된 DB 인스턴스 또는 Aurora Serverless v2 DB 인스턴스를 소스로 사용하는 DB 클러스터를 지정할 수 있습니다.

대상

데이터가 복제되는 데이터 웨어하우스 또는 레이크하우스입니다. 데이터 웨어하우스에는 프로비저닝된 클러스터 데이터 웨어하우스와 서버리스 데이터 웨어하우스라는 2가지 유형이 있습니다. 프로비저닝된 클러스터 데이터 웨어하우스는 노드라고 하는 컴퓨팅 리소스의 모음으로, 노드는 클러스터라고 하는 그룹을 구성합니다. 서버리스 데이터 웨어하우스는 컴퓨팅 리소스를 저장하는 작업 그룹과 데이터베이스 객체 및 사용자를 수용하는 네임스페이스로 구성됩니다. 두 데이터 웨어하우스 모두 분석 엔진을 실행하며 하나 이상의 데이터베이스를 포함합니다.

대상 레이크하우스는 카탈로그, 데이터베이스, 테이블 및 뷰로 구성됩니다. 레이크하우스 아키텍처에 대한 자세한 내용은 Amazon SageMaker Unified Studio 사용 설명서의 Amazon SageMaker Lakehouse components 섹션을 참조하세요.

여러 소스 DB 클러스터가 동일한 대상에 쓸 수 있습니다.

자세한 내용은 Amazon Redshift 개발자 안내서의 데이터 웨어하우스 시스템 아키텍처를 참조하세요.

제한 사항

다음 제한 사항은 Aurora 제로 ETL 통합에 적용됩니다.

주제

일반 제한 사항

- 소스 DB 클러스터는 대상과 동일한 리전에 있어야 합니다.
- 기존 통합이 있는 경우 DB 클러스터 또는 해당 인스턴스의 이름을 변경할 수 없습니다.
- 동일한 소스 데이터베이스와 대상 데이터베이스 간에 여러 통합을 만들 수 없습니다.
- 기존 통합이 있는 DB 클러스터는 삭제할 수 없습니다. 먼저 연결된 모든 통합을 삭제해야 합니다.
- 소스 DB 클러스터를 중지하면 클러스터를 다시 시작할 때까지 마지막 몇 개의 트랜잭션이 복제되지 않을 수 있습니다.
- 클러스터가 블루/그린 배포의 소스인 경우 블루 및 그린 환경은 전환 중에 기존의 제로 ETL 통합을 가질 수 없습니다. 먼저 통합을 삭제하고 전환한 다음 다시 만들어야 합니다.
- DB 클러스터가 통합의 소스가 되려면 DB 인스턴스가 하나 이상 있어야 합니다.
- AWS Resource Access Manager(AWS RAM)를 사용하여 공유되는 것과 같이 교차 계정 복제본인 소스 DB 클러스터에 대한 통합을 만들 수 없습니다.

- 소스 클러스터가 Aurora Global Database의 기본 DB 클러스터이고 이 클러스터가 보조 클러스터 중 하나로 장애 조치되는 경우 통합이 비활성화됩니다. 통합을 삭제하고 다시 만들어야 합니다.
- 다른 통합이 활발하게 생성되고 있는 소스 데이터베이스에 대해 통합을 생성할 수 없습니다.
- 처음에 통합을 생성하거나 테이블을 재동기화할 때는 소스 데이터베이스의 크기에 따라 소스에서 대상으로 데이터를 시드하는 데 20~25분 이상 걸릴 수 있습니다. 이러한 지연으로 인해 복제 지연이 증가할 수 있습니다.
- 일부 데이터 유형은 지원되지 않습니다. 자세한 내용은 the section called “데이터 형식 차이” 섹션을 참조하세요.
- 시스템 테이블, 임시 테이블 및 뷰는 대상 웨어하우스에 복제되지 않습니다.
- `ALTER TABLE` 파티션 작업에서는 Aurora에서 분석 대상으로 데이터를 다시 로드하기 위해 테이블이 다시 동기화됩니다. 테이블을 재동기화하는 동안에는 테이블을 쿼리할 수 없습니다. 자세한 내용은 the section called “Amazon Redshift 테이블 중 하나 이상을 재동기화해야 합니다” 섹션을 참조하세요.

Aurora MySQL 제한 사항

- 소스 DB 클러스터는 지원되는 Aurora MySQL 버전을 실행해야 합니다. 지원되는 버전 목록은 the section called “제로 ETL 통합 ” 섹션을 참조하세요.
- 제로 ETL 통합은 진행 중인 데이터 변경 사항을 캡처하기 위해 MySQL 바이너리 로깅(binlog)에 의존합니다. binlog 기반 데이터 필터링을 사용하면 소스 데이터베이스와 대상 데이터베이스 간에 데이터 불일치가 발생할 수 있으므로 사용하지 않는 것이 좋습니다.
- 제로 ETL 통합은 InnoDB 스토리지 엔진을 사용하도록 구성된 데이터베이스에만 지원됩니다.
- 사전 정의된 테이블 업데이트가 포함된 외래 키 참조는 지원되지 않습니다. 특히, `CASCADE`, `SET NULL`, `SET DEFAULT` 작업에서는 `ON DELETE` 및 `ON UPDATE` 규칙이 지원되지 않습니다. 다른 테이블에 대한 이러한 참조가 포함된 테이블을 만들거나 업데이트하려고 하면 테이블이 실패 상태가 됩니다.
- 소스 DB 클러스터에서 수행되는 XA 트랜잭션으로 인해 통합이 `Syncing` 상태로 전환됩니다.

Aurora PostgreSQL 제한 사항

- 소스 DB 클러스터는 지원되는 Aurora PostgreSQL 버전을 실행해야 합니다. 지원되는 버전 목록은 the section called “제로 ETL 통합 ” 섹션을 참조하세요.
- Aurora PostgreSQL 소스 DB 클러스터를 선택하는 경우 데이터 필터 패턴을 하나 이상 지정해야 합니다. 대상 웨어하우스로 복제하려면 패턴에 최소한 하나의 데이터베이스(*database-name*`.*.*`)

가 포함되어 있어야 합니다. 자세한 내용은 the section called “제로 ETL 통합의 데이터 필터링” 섹션을 참조하세요.

- 소스 Aurora PostgreSQL DB 클러스터 내에서 생성되는 모든 데이터베이스는 UTF-8 인코딩을 사용해야 합니다.
- 소스 DB 클러스터에서 선언적 파티셔닝 트랜잭션을 수행하는 경우 영향을 받는 모든 테이블은 실패 상태가 되며 더 이상 액세스할 수 없습니다.
- 2단계 트랜잭션은 지원되지 않습니다.
- 통합의 소스인 DB 클러스터에서 모든 DB 인스턴스를 삭제한 다음 DB 인스턴스를 다시 추가하면 소스와 대상 클러스터 간에 복제가 중단됩니다.
- 소스 DB 클러스터는 Aurora Limitless Database 사용을 사용할 수 없습니다.

Amazon Redshift 제한 사항

제로 ETL 통합과 관련된 Amazon Redshift 제한 사항 목록은 Amazon Redshift 관리 가이드의 Amazon Redshift와 제로 ETL 통합을 사용할 때 고려할 사항을 참조하세요.

Amazon SageMaker 레이크하우스 제한 사항

다음은 Amazon SageMaker 레이크하우스 제로 ETL 통합에 대한 제한 사항입니다.

- 카탈로그 이름의 길이는 19자로 제한됩니다.

할당량

계정에는 Aurora 제로 ETL 통합과 관련된 다음과 같은 할당량이 있습니다. 각 할당량은 달리 지정되지 않는 한 리전별로 적용됩니다.

명칭	Default	설명
통합	100	AWS 계정 내 총 통합 수입니다.
대상당 통합	50	단일 대상 데이터 웨어하우스 또는 레이크하우스로 데이터를 보내는 통합 수입니다.
소스 클러스터별 통합	5	단일 소스 DB 클러스터에서 데이터를 보내는 통합 수입니다.

또한 대상 웨어하우스는 각 DB 인스턴스 또는 클러스터 노드에 허용되는 테이블 수에 구체적인 제한을 두고 있습니다. Amazon Redshift 할당량 및 제한에 대한 자세한 내용은 Amazon Redshift 관리 안내서의 Amazon Redshift의 할당량 및 제한을 참조하세요.

지원되는 리전

Aurora 제로 ETL 통합은 일부 AWS 리전에서 사용할 수 있습니다. 지원되는 리전 목록은 the section called "제로 ETL 통합 " 섹션을 참조하세요.

Amazon Redshift가 구성된 Aurora 제로 ETL 통합 시작하기

Amazon Redshift가 구성된 제로 ETL 통합을 생성하기 전에 필수 파라미터와 권한을 사용하여 Aurora DB 클러스터와 Amazon Redshift 데이터 웨어하우스를 구성합니다. 설정 중에 다음 단계를 완료해야 합니다.

1. 사용자 지정 DB 클러스터 파라미터 그룹을 만듭니다.
2. 소스 DB 클러스터를 생성합니다.
3. 대상 Amazon Redshift 데이터 웨어하우스를 만듭니다.

이러한 작업을 완료한 후 the section called "Amazon Redshift와 제로 ETL 통합 생성"으로 이동합니다.

AWS SDK를 사용하여 설정 프로세스를 자동화할 수 있습니다. 자세한 내용은 the section called "AWS SDK를 사용하여 통합을 설정합니다(Aurora MySQL만 해당)." 단원을 참조하십시오.

1단계: 사용자 지정 DB 클러스터 파라미터 그룹 생성

Amazon Redshift가 구성된 Aurora 제로 ETL 통합은 복제를 제어하는 DB 클러스터 파라미터에 대한 특정 값이 필요합니다. 특히 Aurora MySQL에는 향상된 binlog(`aurora_enhanced_binlog`)가 필요하고 Aurora PostgreSQL에는 향상된 논리적 복제(`aurora.enhanced_logical_replication`)가 필요합니다.

이진 로깅 또는 논리적 복제를 구성하려면 먼저 사용자 지정 DB 클러스터 파라미터 그룹을 만든 다음 이를 소스 DB 클러스터와 연결해야 합니다.

소스 DB 엔진에 따라 다음 설정을 사용하여 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다. 파라미터 그룹을 만드는 방법에 대한 지침은 the section called "DB 클러스터 파라미터 그룹" 섹션을 참조하세요.

Aurora MySQL(aurora-mysql8.0 제품군):

- `aurora_enhanced_binlog=1`
- `binlog_backup=0`
- `binlog_format=ROW`
- `binlog_replication_globaldb=0`
- `binlog_row_image=full`
- `binlog_row_metadata=full`

또한 `binlog_transaction_compression` 파라미터가 `ON`으로 설정되어 있지 않고 `binlog_row_value_options` 파라미터가 `PARTIAL_JSON`으로 설정되어 있지 않아야 합니다.

Aurora MySQL의 향상된 binlog에 대한 자세한 내용은 [the section called "향상된 binlog 설정"] 섹션을 참조하세요.

Aurora PostgreSQL(aurora-postgresql15 제품군):

> **Note**
>
> Aurora PostgreSQL DB 클러스터의 경우 미국 동부(오하이오)(us-east-2) AWS 리전의 [Amazon RDS 데이터베이스 미리 보기 환경] 내에 사용자 지정 파라미터 그룹을 생성해야 합니다.

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

향상된 논리적 복제(`aurora.enhanced_logical_replication`)를 활성화하면 `REPLICA IDENTITY` 파라미터가 자동으로 `FULL`로 설정됩니다. 즉, 모든 열 값이 Wrate Ahead Log(WAL)에 기록됩니다. 이렇게 하면 소스 DB 클러스터의 IOPS가 증가합니다.

2단계: 소스 DB 클러스터 선택 또는 생성

사용자 지정 DB 클러스터 파라미터 그룹을 생성한 후 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터를 선택 또는 생성합니다. 이 클러스터는 Amazon Redshift로의 데이터 복제 소스가 됩니다.

클러스터는 Aurora MySQL 버전 3.05(MySQL 8.0.32와 호환) 이상 또는 Aurora PostgreSQL(PostgreSQL 15.4 및 제로 ETL 지원과 호환)을 실행해야 합니다. DB 클러스터를 만드는 방법에 대한 지침은 the section called "DB 클러스터 생성" 섹션을 참조하세요.

> **Note**
>
> 미국 동부(오하이오)(us-east-2) AWS 리전의 Amazon RDS 데이터베이스 미리 보기 환경 내에 Aurora PostgreSQL DB 클러스터를 생성해야 합니다.

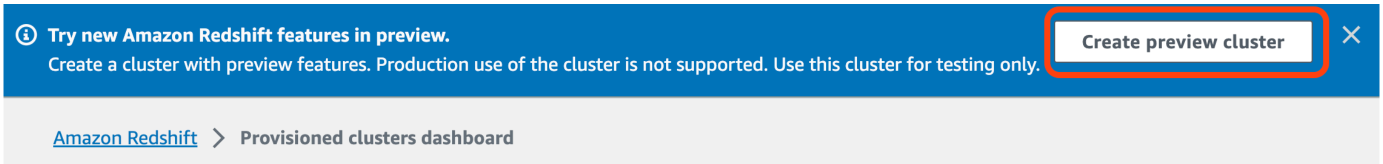
추가 구성에서 기본 DB 클러스터 파라미터 그룹을 이전 단계에서 생성한 사용자 지정 파라미터 그룹으로 변경합니다.

> **Note**
>
> Aurora MySQL의 경우, 클러스터가 이미 생성된 후에 DB 클러스터와 파라미터 그룹을 제로 ETL 통합을 만들기 전에 클러스터의 기본 DB 인스턴스를 재부팅하여 변경 사항을 적용해야 합니다. 지침은 the section called "Aurora DB 클러스터 또는 인스턴스 재부팅" 단원을 참조하십시오.
> Amazon Redshift가 구성된 Aurora PostgreSQL 제로 ETL 통합의 미리 보기 릴리스 중에는 클러스터를 생성하는 동안 클러스터를 사용자 지정 DB 클러스터 파라미터 그룹과 연결해야 합니다. 소스 DB 클러스터를 이미 생성한 후에는 이 작업을 수행할 수 없습니다. 수행하려면 클러스터를 삭제하고 다시 생성해야 합니다.

3단계: 대상 Amazon Redshift 데이터 웨어하우스 생성

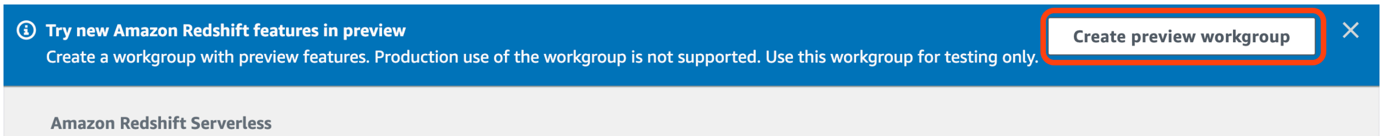
소스 DB 클러스터를 생성한 후에는 Amazon Redshift에서 대상 데이터 웨어하우스를 생성하고 구성해야 합니다. 데이터 웨어하우스는 다음 요구 사항을 충족해야 합니다.

- 미리 보기에서 생성되었습니다(Aurora PostgreSQL 소스만 해당). Aurora MySQL 소스의 경우 프로덕션 클러스터와 작업 그룹을 생성해야 합니다.
  - 미리 보기에서 프로비저닝된 클러스터를 생성하려면 프로비저닝된 클러스터 대시보드의 배너에서 미리 보기 클러스터 생성을 선택합니다. 자세한 내용은 미리 보기 클러스터 생성 섹션을 참조하세요.



클러스터를 생성할 때 미리 보기 트랙을 `preview_2023`으로 설정하세요.

- 미리 보기에서 Redshift Serverless 작업 그룹을 만들려면 서버리스 대시보드의 배너에서 미리 보기 작업 그룹 생성을 선택합니다. 자세한 내용은 미리 보기 작업 그룹 생성 섹션을 참조하세요.



- RA3 노드 유형(`ra3.xlplus`, `ra3.4xlarge`, `ra3.16xlarge`) 또는 Redshift Serverless를 사용합니다.
- 암호화되어 있습니다(프로비저닝된 클러스터를 사용하는 경우). 자세한 내용은 Amazon Redshift 데이터베이스 암호화를 참조하세요.

데이터 웨어하우스를 만드는 방법에 대한 지침은 프로비전된 클러스터의 경우 클러스터 생성을, Redshift Serverless의 경우 네임스페이스가 있는 작업 그룹 생성을 참조하세요.

데이터 웨어하우스에서 대/소문자 구분 활성화

통합이 성공하려면 데이터 웨어하우스에서 대/소문자 구분 파라미터 (`enable_case_sensitive_identifier`)를 활성화해야 합니다. 기본적으로 모든 프로비저닝된 클러스터와 Redshift Serverless 작업 그룹에서 대/소문자 구분이 비활성화되어 있습니다.

대/소문자 구분을 활성화하려면 데이터 웨어하우스 유형에 따라 다음 단계를 수행하세요.

- 프로비저닝된 클러스터 - 프로비저닝된 클러스터에서 대/소문자 구분을 활성화하려면 `enable_case_sensitive_identifier` 파라미터가 활성화된 사용자 지정 파라미터 그룹을 생성합니다. 그런 다음 이 파라미터 그룹을 클러스터와 연결합니다. 자세한 지침은 콘솔을 사용한 파라미터 그룹 관리 또는 AWS CLI를 사용한 파라미터 값 구성을 참조하세요.

> **Note**
>
> 사용자 지정 파라미터 그룹을 연결한 후 클러스터를 재부팅해야 합니다.

- Serverless 작업 그룹 - Redshift Serverless 작업 그룹에서 대/소문자 구분을 활성화하려면 AWS CLI를 사용해야 합니다. Amazon Redshift 콘솔은 현재 Redshift Serverless 파라미터 값 수정을 지원하지 않습니다. 다음 update-workgroup 업데이트 요청을 보냅니다.

```
aws redshift-serverless update-workgroup \
  --workgroup-name target-workgroup \
  --config-parameters
 parameterKey=enable_case_sensitive_identifier,parameterValue=true
```

  작업 그룹의 파라미터 값을 수정한 후 작업 그룹을 재부팅할 필요가 없습니다.

데이터 웨어하우스에 대한 권한 부여 구성

데이터 웨어하우스를 만든 후에는 소스 Aurora DB 클러스터를 승인된 통합 소스로 구성해야 합니다. 자세한 지침은 Amazon Redshift 데이터 웨어하우스에 대한 권한 부여 구성을 참조하세요.

AWS SDK를 사용하여 통합을 설정합니다(Aurora MySQL만 해당).

각 리소스를 수동으로 설정하는 대신 다음 Python 스크립트를 실행하여 필요한 리소스를 자동으로 설정할 수 있습니다. 코드 예제에서는 AWS SDK for Python (Boto3)을 사용하여 소스 Aurora MySQL DB 클러스터와 대상 Amazon Redshift 데이터 웨어하우스를 생성합니다. 각 웨어하우스에는 필수 파라미터 값이 있습니다. 그런 다음 클러스터 간에 제로 ETL 통합을 생성하기 전에 클러스터를 사용할 수 있을 때까지 기다립니다. 설정해야 하는 리소스에 따라 다양한 함수를 주석 처리할 수 있습니다.

필요한 종속성을 설치하려면 다음 명령을 실행합니다.

```
pip install boto3
pip install time
```

스크립트 내에서 필요에 따라 소스, 대상 및 파라미터 그룹의 이름을 수정할 수 있습니다. 최종 함수는 리소스 설정에 따라 `my-integration` 이름이 지정된 통합을 생성합니다.

Python 코드 예제

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.
```

```
rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
```

```
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
```

```
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
```

```
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
```

```
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

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

다음 단계

소스 Aurora DB 클러스터와 Amazon Redshift 대상 데이터 웨어하우스가 있으므로, 이제 제로 ETL 통합을 생성하고 데이터를 복제할 수 있습니다. 지침은 the section called "Amazon Redshift와 제로 ETL 통합 생성" 단원을 참조하세요.

Amazon Redshift가 구성된 Amazon Aurora 제로 ETL 통합 생성

Aurora 제로 ETL 통합을 생성할 때는 소스 Aurora DB 클러스터와 대상 Amazon Redshift 데이터 웨어하우스를 지정합니다. 암호화 설정을 사용자 지정하고 태그를 추가할 수도 있습니다. Aurora는 소스 DB 클러스터와 대상 간에 통합을 생성합니다. 통합이 활성화되면 소스 DB 클러스터에 삽입한 모든 데이터가 구성된 Amazon Redshift 대상에 복제됩니다.

사전 조건

제로 ETL 통합을 생성하기 전에 소스 DB 클러스터와 대상 Amazon Redshift 데이터 웨어하우스를 생성해야 합니다. 또한 DB 클러스터를 인증된 통합 소스로 추가하여 데이터 웨어하우스로의 복제를 허용해야 합니다.

각 단계를 완료하기 위한 지침은 the section called “제로 ETL 통합 시작하기” 섹션을 참조하세요.

필수 권한

제로 ETL 통합을 생성하려면 특정 IAM 권한이 필요합니다. 최소한 다음 작업을 수행할 수 있는 권한이 필요합니다.

- 소스 Aurora DB 클러스터를 위한 제로 ETL 통합을 생성합니다.
- 모든 제로 ETL 통합을 보고 삭제합니다.
- 대상 데이터 웨어하우스에 인바운드 통합을 생성합니다.

다음 샘플 정책은 통합을 생성하고 관리하는 데 필요한 최소 권한을 보여줍니다. 사용자 또는 역할에 더 광범위한 권한(예: `AdministratorAccess` 관리형 정책)이 있는 경우 이러한 정확한 권한이 필요하지 않을 수 있습니다.

> **Note**
>
> Redshift Amazon 리소스 이름(ARN)의 형식은 다음과 같습니다. 서버리스 네임스페이스 UUID 앞에는 콜론(`:`) 대신 슬래시((`/`)를 사용한다는 점에 유의하세요.
>
> - 프로비저닝된 클러스터 – `arn:aws:`**`redshift`**`:{region}:{account-id}:namespace`**`:`***`namespace-uuid`*
> - 서버리스 – `arn:aws:`**`redshift-serverless`**`:{region}:{account-id}:namespace`**`/`***`namespace-uuid`*

Redshift 대상에 대한 샘플 정책

JSON

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rds:CreateIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:cluster:source-db",
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DescribeIntegrations"
        ],
        "Resource": ["*"]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DeleteIntegration",
            "rds:ModifyIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "redshift:CreateInboundIntegration"
        ],
        "Resource": [
            "arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid"
        ]
    }]
}
```

다른 계정에서 대상 데이터 웨어하우스 선택

다른 AWS 계정에 있는 대상 Amazon Redshift 데이터 웨어하우스를 지정하려면 현재 계정의 사용자가 대상 계정의 리소스에 액세스할 수 있도록 허용하는 역할을 만들어야 합니다. 자세한 내용은 소유한 다른 AWS 계정의 IAM 사용자에 액세스 권한 제공을 참조하세요.

역할에는 사용자가 대상 계정에서 사용 가능한 Amazon Redshift 프로비저닝 클러스터와 Redshift Serverless 네임스페이스를 볼 수 있도록 허용하는 다음 권한이 있어야 합니다.

필수 권한 및 신뢰 정책

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

역할에는 대상 계정 ID를 지정하는 다음과 같은 신뢰 정책이 있어야 합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
```

```
            },
            "Action":"sts:AssumeRole"
        }
    ]
}
```

역할을 생성하기 위한 지침은 [사용자 지정 신뢰 정책을 사용하여 역할 생성]을 참조하세요.

제로 ETL 통합 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 제로 ETL 통합을 생성할 수 있습니다.

> **Important**
>
> 제로 ETL 통합은 새로 고침 또는 재동기화 작업을 지원하지 않습니다. 생성 후 통합에 문제가 발생하면 통합을 삭제하고 새 통합을 생성해야 합니다.

RDS 콘솔

제로 ETL 통합을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 제로 ETL 통합을 선택합니다.
3. 제로 ETL 통합 생성을 선택합니다.
4. 통합 식별자에 통합의 이름을 입력합니다. 이름은 최대 63자의 영숫자로 구성할 수 있으며, 하이픈을 포함할 수 있습니다.

   > **Important**
   >
   > 카탈로그 이름의 길이는 19자로 제한됩니다. 통합 식별자가 카탈로그 이름으로 사용될 경우 이 요구 사항을 충족하는지 확인합니다.

5. 다음을 선택합니다.
6. 소스에서 데이터의 출처가 될 Aurora DB 클러스터를 선택합니다.

> **Note**
>
> DB 클러스터 파라미터가 올바르게 구성되지 않은 경우 RDS에서 알려줍니다. 이 메시지를 받으면 수정 요청을 선택하거나 수동으로 구성할 수 있습니다. 수동으로 수정하는 방법에 대한 지침은 the section called “1단계: 사용자 지정 DB 클러스터 파라미터 그룹 생성” 섹션을 참조하세요.
> DB 클러스터 파라미터를 수정하려면 재부팅해야 합니다. 통합을 생성하려면 먼저 재부팅을 완료하고 새 파라미터 값을 클러스터에 성공적으로 적용해야 합니다.

7. (선택 사항) 데이터 필터링 옵션 사용자 지정을 선택하고 통합에 데이터 필터를 추가합니다. 데이터 필터를 사용하여 대상 데이터 웨어하우스로 복제할 범위를 정의할 수 있습니다. 자세한 내용은 the section called “제로 ETL 통합의 데이터 필터링” 섹션을 참조하세요.
8. 소스 DB 클러스터가 성공적으로 구성되면 다음을 선택합니다.
9. 대상에서 다음을 수행합니다.

   1. (선택 사항) Amazon Redshift 대상에 다른 AWS 계정를 사용하려면 다른 계정 지정을 선택합니다. 그런 다음 데이터 웨어하우스를 표시할 권한이 있는 IAM 역할의 ARN을 입력합니다. IAM 역할을 생성하는 방법에 대한 지침은 the section called “다른 계정에서 대상 데이터 웨어하우스 선택” 섹션을 참조하세요.
   2. Amazon Redshift 데이터 웨어하우스의 경우 소스 DB 클러스터에서 복제된 데이터의 대상을 선택합니다. 프로비저닝된 Amazon Redshift 클러스터 또는 Redshift Serverless 네임스페이스를 대상으로 선택할 수 있습니다.

> **Note**
>
> 지정된 데이터 웨어하우스의 리소스 정책 또는 대/소문자 구분 설정이 올바르게 구성되지 않은 경우 RDS에서 알려줍니다. 이 메시지를 받으면 수정 요청을 선택하거나 수동으로 구성할 수 있습니다. 수동으로 수정하는 방법에 대한 지침은 Amazon Redshift 관리 가이드의 데이터 웨어하우스에 대/소문자 구분 기능 사용 설정 및 데이터 웨어하우스에 대한 권한 부여 구성을 참조하세요.
> 프로비저닝된 Redshift 클러스터의 대/소문자 구분을 수정하려면 재부팅해야 합니다. 통합을 생성하려면 먼저 재부팅을 완료하고 새 파라미터 값을 클러스터에 성공적으로 적용해야 합니다.

선택한 소스와 대상이 다른 AWS 계정에 있는 경우 Amazon RDS에서 이러한 설정을 수정할 수 없습니다. Amazon Redshift에서 다른 계정으로 이동하여 수동으로 수정해야 합니다.

10. 대상 데이터 웨어하우스가 올바르게 구성되었으면 다음을 선택합니다.
11. (선택 사항) 태그에서 통합에 하나 이상의 태그를 추가합니다. 자세한 내용은 the section called "Aurora 및 RDS 리소스에 태그 지정" 섹션을 참조하세요.
12. 암호화에서 통합을 암호화할 방법을 지정합니다. 기본적으로 RDS는 AWS 소유 키를 사용하여 모든 통합을 암호화합니다. 고객 관리형 키를 대신 선택하려면 암호화 설정 사용자 지정을 활성화하고 암호화에 사용할 KMS 키를 선택합니다. 자세한 내용은 the section called "Amazon Aurora 리소스 암호화" 섹션을 참조하세요.

    선택적으로 암호화 컨텍스트를 추가합니다. 자세한 내용은 AWS Key Management Service 개발자 가이드에서 암호화 컨텍스트를 참조하세요.

    > **Note**
    >
    > Amazon RDS는 추가한 모든 항목 외에도 다음과 같은 암호화 컨텍스트 쌍을 추가합니다.
    >
    > - `aws:redshift:integration:arn` - `IntegrationArn`
    > - `aws:servicename:id` - `Redshift`
    >
    > 이렇게 하면 추가할 수 있는 전체 쌍 수가 8개에서 6개로 줄어들고 권한 부여 제약 조건의 전체 문자 제한이 늘어납니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 권한 부여 제약 사용을 참조하세요.

13. 다음을 선택합니다.
14. 통합 설정을 검토하고 제로 ETL 통합 생성을 선택합니다.

    생성에 실패할 경우 the section called "제로 ETL 통합을 생성할 수 없습니다"에서 문제 해결 단계를 확인하세요.

통합 상태는 생성 중인 동안 `Creating`이며, 대상 Amazon Redshift 데이터 웨어하우스의 상태는 `Modifying`입니다. 이 기간 동안에는 데이터 웨어하우스를 쿼리하거나 구성을 변경할 수 없습니다.

통합이 성공적으로 생성되면 통합 상태와 대상 Amazon Redshift 데이터 웨어하우스가 모두 `Active`로 변경됩니다.

AWS CLI

AWS CLI를 사용하여 제로 ETL 통합을 생성하려면 다음 옵션과 함께 create-integration 명령을 사용하세요.

> **Note**
>
> 카탈로그 이름은 19자로 제한된다는 점에 유의하세요. 카탈로그 이름으로 사용할 경우 적절하게 통합 이름을 선택합니다.

- `--integration-name` - 통합 이름을 지정합니다.
- `--source-arn` - 통합의 소스가 될 Aurora DB 클러스터의 ARN을 지정합니다.
- `--target-arn` - 통합의 대상이 될 Amazon Redshift 데이터 웨어하우스의 ARN을 지정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

Windows의 경우:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS API

Amazon RDS API를 사용하여 제로 ETL 통합을 생성하려면 다음 파라미터를 적용한 `CreateIntegration` 작업을 사용하세요.

> **Note**
>
> 카탈로그 이름은 19자로 제한됩니다. IntegrationName 파라미터가 카탈로그 이름으로 사용될 경우 이 요구 사항을 충족하는지 확인합니다.

- `IntegrationName` - 통합 이름을 지정합니다.
- `SourceArn` - 통합의 소스가 될 Aurora DB 클러스터의 ARN을 지정합니다.
- `TargetArn` - 통합의 대상이 될 Amazon Redshift 데이터 웨어하우스의 ARN을 지정합니다.

고객 관리형 키를 사용하여 통합 암호화

통합을 생성할 때 AWS 소유 키가 아닌 사용자 지정 KMS 키를 설정하는 경우 키 정책은 Amazon Redshift 서비스 보안 주체에 `CreateGrant` 작업에 대한 액세스 권한을 제공해야 합니다. 또한 현재 사용자가 `DescribeKey` 및 `CreateGrant` 작업을 수행할 수 있도록 허용해야 합니다.

다음 샘플 정책은 키 정책에 필요한 권한을 제공하는 방법을 보여줍니다. 여기에는 권한 범위를 더 축소하기 위한 컨텍스트 키가 포함됩니다.

샘플 키 정책

JSON

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
 key",
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
```

```
                "Effect": "Allow",
                "Principal": {
                    "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
                },
                "Action": "kms:DescribeKey",
                "Resource": "*"
            }
        ]
}
```

자세한 내용은 AWS Key Management Service 개발자 안내서의 키 정책 생성을 참조하세요.

다음 단계

제로 ETL 통합을 성공적으로 생성한 후에는 대상 Amazon Redshift 클러스터 또는 작업 그룹 내에 대상 데이터베이스를 생성해야 합니다. 그런 다음 소스 Aurora DB 클러스터에 데이터를 추가하고 Amazon Redshift에서 쿼리할 수 있습니다. 자세한 지침은 Amazon Redshift에서 대상 데이터베이스 생성을 참조하세요.

Amazon SageMaker 레이크하우스와 Aurora 제로 ETL 통합 생성

Amazon SageMaker 레이크하우스와 Aurora 제로 ETL 통합을 생성할 때 소스 Aurora DB 클러스터와 대상 AWS Glue 관리형 카탈로그를 지정합니다. 암호화 설정을 사용자 지정하고 태그를 추가할 수도 있습니다. Aurora는 소스 DB 클러스터와 대상 간에 통합을 생성합니다. 통합이 활성화되면 소스 DB 클러스터에 삽입한 모든 데이터가 구성된 대상에 복제됩니다.

사전 조건

Amazon SageMaker 레이크하우스와 제로 ETL 통합을 생성하기 전에 소스 DB 클러스터와 대상 AWS Glue 관리형 카탈로그를 생성해야 합니다. 또한 DB 클러스터를 인증된 통합 소스로 추가하여 카탈로그로의 복제를 허용해야 합니다.

각 단계를 완료하기 위한 지침은 the section called “제로 ETL 통합 시작하기” 섹션을 참조하세요.

필수 권한

Amazon SageMaker 레이크하우스와 제로 ETL 통합을 생성하려면 특정 IAM 권한이 필요합니다. 최소한 다음 작업을 수행할 수 있는 권한이 필요합니다.

- 소스 Aurora DB 클러스터를 위한 제로 ETL 통합을 생성합니다.
- 모든 제로 ETL 통합을 보고 삭제합니다.
- 대상 AWS Glue 관리형 카탈로그에 인바운드 통합을 생성합니다.
- AWS Glue 관리형 카탈로그에서 사용하는 Amazon S3 버킷에 액세스합니다.
- 사용자 지정 암호화가 구성된 경우 암호화에 AWS KMS 키를 사용합니다.
- Lake Formation에 리소스를 등록합니다.
- AWS Glue 관리형 카탈로그에 리소스 정책을 적용하여 인바운드 통합을 승인합니다.

다음 샘플 정책은 Amazon SageMaker 레이크하우스와 통합을 생성하고 관리하는 데 필요한 최소 권한을 보여줍니다. 사용자 또는 역할에 더 광범위한 권한(예: `AdministratorAccess` 관리형 정책)이 있는 경우 이러한 정확한 권한이 필요하지 않을 수 있습니다.

또한 인바운드 통합을 승인하려면 대상 AWS Glue 관리형 카탈로그에서 리소스 정책을 구성해야 합니다. 다음 AWS CLI명령을 실행하여 리소스 정책을 적용합니다.

대상 카탈로그에서 인바운드 통합을 승인하는 샘플 AWS CLI 명령

```
aws glue put-resource-policy \
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
```

```
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
}' \
      --region region
```

> **Note**
>
> Glue 카탈로그 Amazon 리소스 이름(ARN) 형식은 다음과 같습니다.
>
> - Glue 카탈로그 – `arn:aws:glue:{region}:{account-id}:catalog/catalog-name`

다른 계정에서 대상 AWS Glue 관리형 카탈로그 선택

다른 AWS 계정에 있는 대상 AWS Glue 관리형 카탈로그를 지정하려면 현재 계정의 사용자가 대상 계정의 리소스에 액세스할 수 있도록 허용하는 역할을 만들어야 합니다. 자세한 내용은 소유한 다른 AWS 계정의 IAM 사용자에 액세스 권한 제공을 참조하세요.

역할에는 사용자가 대상 계정에서 사용 가능한 AWS Glue 카탈로그를 볼 수 있도록 허용하는 다음 권한이 있어야 합니다.

필수 권한 및 신뢰 정책

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:GetCatalog"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

역할에는 대상 계정 ID를 지정하는 다음과 같은 신뢰 정책이 있어야 합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

역할을 생성하기 위한 지침은 사용자 지정 신뢰 정책을 사용하여 역할 생성을 참조하세요.

Amazon SageMaker 레이크하우스와 제로 ETL 통합 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Amazon SageMaker 레이크하우스와 제로 ETL 통합을 생성할 수 있습니다.

> **Important**
>
> Amazon SageMaker 레이크하우스와의 제로 ETL 통합은 새로 고침 또는 재동기화 작업을 지원하지 않습니다. 생성 후 통합에 문제가 발생하면 통합을 삭제하고 새 통합을 생성해야 합니다.

RDS 콘솔

Amazon SageMaker 레이크하우스와 제로 ETL 통합을 생성하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 제로 ETL 통합을 선택합니다.
3. 제로 ETL 통합 생성을 선택합니다.
4. 통합 식별자에 통합의 이름을 입력합니다. 이름은 최대 63자의 영숫자로 구성할 수 있으며, 하이픈을 포함할 수 있습니다.
5. 다음을 선택합니다.
6. 소스에서 데이터의 출처가 될 Aurora DB 클러스터를 선택합니다.

> **Note**
>
> DB 클러스터 파라미터가 올바르게 구성되지 않은 경우 RDS에서 알려줍니다. 이 메시지를 받으면 수정 요청을 선택하거나 수동으로 구성할 수 있습니다. 수동으로 수정하는 방법에 대한 지침은 the section called "1단계: 사용자 지정 DB 클러스터 파라미터 그룹 생성" 섹션을 참조하세요.
>
> DB 클러스터 파라미터를 수정하려면 재부팅해야 합니다. 통합을 생성하려면 먼저 재부팅을 완료하고 새 파라미터 값을 클러스터에 성공적으로 적용해야 합니다.

7. (선택 사항) 데이터 필터링 옵션 사용자 지정을 선택하고 통합에 데이터 필터를 추가합니다. 데이터 필터를 사용하여 대상 Amazon SageMaker 레이크하우스로 복제할 범위를 정의할 수 있습니다. 자세한 내용은 the section called "제로 ETL 통합의 데이터 필터링" 섹션을 참조하세요.
8. 소스 DB 클러스터가 성공적으로 구성되면 다음을 선택합니다.
9. 대상에서 다음을 수행합니다.

    1. (선택 사항) Amazon SageMaker 레이크하우스 대상에 다른 AWS 계정을 사용하려면 다른 계정 지정을 선택합니다. 그런 다음 AWS Glue 카탈로그를 표시할 권한이 있는 IAM 역할의 ARN을 입력합니다. IAM 역할을 생성하는 방법에 대한 지침은 the section called "다른 계정에서 대상 AWS Glue 관리형 카탈로그 선택" 섹션을 참조하세요.
    2. AWS Glue 카탈로그의 경우 소스 DB 클러스터에서 복제된 데이터의 대상을 선택합니다. 기존 AWS Glue 관리형 카탈로그를 대상으로 선택할 수 있습니다.
    3. 대상 IAM 역할에는 대상 카탈로그에 대한 설명 권한이 필요하며 다음 권한이 있어야 합니다.

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "VisualEditor0",
                "Effect": "Allow",
                "Action": "glue:GetCatalog",
                "Resource": [
                    "arn:aws:glue:region:account-id:catalog/*",
                    "arn:aws:glue:region:account-id:catalog"
                ]
            }
        ]
    }
    ```

대상 IAM 역할은 다음과 같은 신뢰 관계를 맺고 있어야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. ???에서 생성된 Lake Formation 관리자 역할을 사용하여 대상 AWS Glue 관리형 카탈로그에 대한 권한을 대상 IAM 역할에 부여해야 합니다.

> **Note**
>
> RDS는 지정된 AWS Glue 관리형 카탈로그의 리소스 정책 또는 구성 설정이 올바르게 구성되지 않은 경우 알려줍니다. 이 메시지를 받으면 수정 요청을 선택하거나 수동으로 구성할 수 있습니다.
> 선택한 소스와 대상이 다른 AWS 계정에 있는 경우 Amazon RDS에서 이러한 설정을 수정할 수 없습니다. SageMaker Unified Studio에서 다른 계정으로 이동하여 수동으로 수정해야 합니다.

10. 대상 AWS Glue 관리형 카탈로그가 올바르게 구성되면 다음을 선택합니다.

11. (선택 사항) 태그에서 통합에 하나 이상의 태그를 추가합니다. 자세한 내용은 the section called "Aurora 및 RDS 리소스에 태그 지정" 섹션을 참조하세요.

12. 암호화에서 통합을 암호화할 방법을 지정합니다. 기본적으로 RDS는 AWS 소유 키를 사용하여 모든 통합을 암호화합니다. 고객 관리형 키를 대신 선택하려면 암호화 설정 사용자 지정을 활성화하고 암호화에 사용할 KMS 키를 선택합니다. 자세한 내용은 the section called "Amazon Aurora 리소스 암호화" 섹션을 참조하세요.

    선택적으로 암호화 컨텍스트를 추가합니다. 자세한 내용은 AWS Key Management Service 개발자 가이드에서 암호화 컨텍스트를 참조하세요.

> **Note**
>
> Amazon RDS는 추가한 모든 항목 외에도 다음과 같은 암호화 컨텍스트 쌍을 추가합니다.
>
> - `aws:glue:integration:arn` - `IntegrationArn`
> - `aws:servicename:id` - `glue`
>
> 이렇게 하면 추가할 수 있는 전체 쌍 수가 8개에서 6개로 줄어들고 권한 부여 제약 조건의 전체 문자 제한이 늘어납니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 권한 부여 제약 사용을 참조하세요.

13. 다음을 선택합니다.
14. 통합 설정을 검토하고 제로 ETL 통합 생성을 선택합니다.

    생성에 실패할 경우 the section called “제로 ETL 통합 문제 해결”에서 문제 해결 단계를 확인하세요.

통합 상태는 생성 중인 동안 `Creating`이며, 대상 Amazon SageMaker 레이크하우스의 상태는 `Modifying`입니다. 이 기간 동안에는 카탈로그를 쿼리하거나 구성을 변경할 수 없습니다.

통합이 성공적으로 생성되면 통합과 대상 Amazon SageMaker 레이크하우스의 상태가 모두 `Active`로 변경됩니다.

AWS CLI

AWS CLI를 사용하여 제로 ETL 통합을 위한 대상 AWS Glue 관리형 카탈로그를 준비하려면 먼저 다음 옵션과 함께 create-integration-resource-property 명령을 사용해야 합니다.

- `--resource-arn` - 통합의 대상이 될 AWS Glue 관리형 카탈로그의 ARN을 지정합니다.
- `--target-processing-properties` - 대상 AWS Glue 관리형 카탈로그에 액세스할 IAM 역할의 ARN을 지정합니다.

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

AWS CLI를 사용하여 Amazon SageMaker 레이크하우스와 제로 ETL 통합을 생성하려면 다음 옵션과 함께 [create-integration](https://docs.aws.amazon.com/cli/latest/reference/rds/create-integration.html) 명령을 사용합니다.

- `--integration-name` - 통합 이름을 지정합니다.
- `--source-arn` - 통합의 소스가 될 Aurora DB 클러스터의 ARN을 지정합니다.
- `--target-arn` - 통합의 대상이 될 AWS Glue 관리형 카탈로그의 ARN을 지정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

Windows의 경우:

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

RDS API

Amazon RDS API를 사용하여 Amazon SageMaker와 제로 ETL 통합을 생성하려면 다음 파라미터를 적용한 `CreateIntegration` 작업을 사용합니다.

> **Note**
>
> 카탈로그 이름은 19자로 제한됩니다. IntegrationName 파라미터가 카탈로그 이름으로 사용될 경우 이 요구 사항을 충족하는지 확인합니다.

- `IntegrationName` - 통합 이름을 지정합니다.
- `SourceArn` - 통합의 소스가 될 Aurora DB 클러스터의 ARN을 지정합니다.
- `TargetArn` - 통합의 대상이 될 AWS Glue 관리형 카탈로그의 ARN을 지정합니다.

고객 관리형 키를 사용하여 통합 암호화

Amazon SageMaker와 통합을 생성할 때 AWS 소유 키가 아닌 사용자 지정 KMS 키를 지정하는 경우 키 정책은 SageMaker Unified Studio 서비스 위탁자에 `CreateGrant` 작업에 대한 액세스 권한을 제공해야 합니다. 또한 현재 사용자가 `DescribeKey` 및 `CreateGrant` 작업을 수행할 수 있도록 허용해야 합니다.

다음 샘플 정책은 키 정책에 필요한 권한을 제공하는 방법을 보여줍니다. 여기에는 권한 범위를 더 축소하기 위한 컨텍스트 키가 포함됩니다.

샘플 키 정책

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Glue service principal to add a grant to an AWS KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
```

```
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about a KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

자세한 내용은 AWS Key Management Service 개발자 안내서의 키 정책 생성을 참조하세요.

다음 단계

Amazon SageMaker와 제로 ETL 통합을 성공적으로 생성한 후 소스 Aurora DB 클러스터에 데이터를 추가하고 Amazon SageMaker 레이크하우스에서 쿼리를 시작할 수 있습니다. 데이터는 자동으로 복제되어 분석 및 기계 학습 워크로드에 사용할 수 있습니다.

Aurora 제로 ETL 통합의 데이터 필터링

Aurora 제로 ETL 통합은 데이터 필터링을 지원하므로 사용자는 소스 Aurora DB 클러스터에서 대상 데이터 웨어하우스로 어떤 데이터가 복제될지 제어할 수 있습니다. 전체 데이터베이스를 복제하는 대신 하나 이상의 필터를 적용하여 특정 테이블을 선택적으로 포함하거나 제외할 수 있습니다. 이렇게 하면 관련 데이터만 전송되도록 하여 스토리지 및 쿼리 성능을 최적화할 수 있습니다. 현재 필터링 기능은 데이터베이스 및 테이블 수준으로 제한됩니다. 열 수준 및 행 수준 필터링은 지원되지 않습니다.

데이터 필터링은 다음과 같은 경우에 유용할 수 있습니다.

- 서로 다른 2개 이상의 소스 클러스터의 특정 테이블을 조인하며 클러스터 중의 전체 데이터가 필요하지 않은 경우
- 전체 데이터베이스 플릿 대신 테이블의 하위 집합만 사용해 분석을 수행하여 비용을 절감하려는 경우
- 특정 테이블에서 전화번호, 주소 또는 신용카드 세부 정보와 같은 민감한 정보를 필터링하는 경우

AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 제로 ETL 통합에 데이터 필터를 추가할 수 있습니다.

통합에서 프로비저닝된 클러스터를 대상으로 하는 경우 클러스터가 패치 180 이상이어야 데이터 필터링을 사용할 수 있습니다.

주제

데이터 필터의 형식

단일 통합에 대해 여러 필터를 정의할 수 있습니다. 각 필터는 필터 표현식의 패턴 중 하나와 일치하는 기존 및 미래 데이터베이스 테이블을 포함하거나 제외합니다. Aurora 제로 ETL 통합은 데이터 필터링에 Maxwell 필터 구문을 사용합니다.

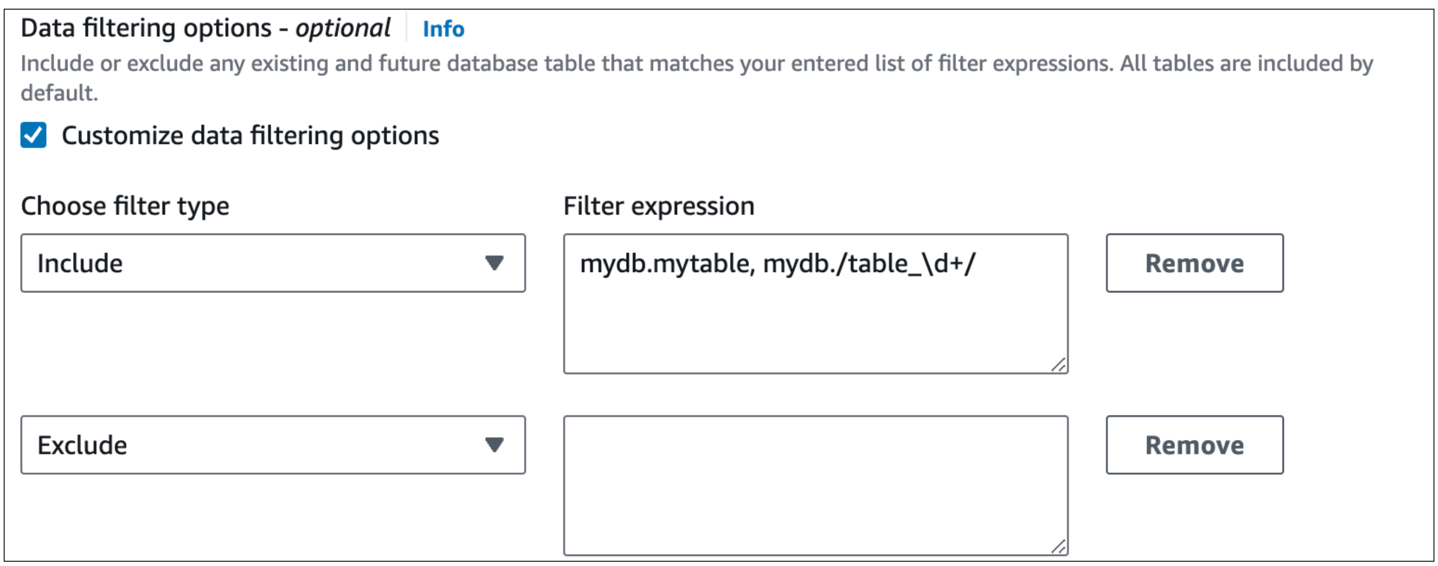
각 필터에는 다음 요소가 포함됩니다.

Element	설명
필터 유형	`Include` 필터 유형에는 필터 표현식의 패턴 중 하나와 일치하는 모든 테이블이 포함됩니다. `Exclude` 필터 유형에는 패턴 중 하나와 일치하는 모든 테이블이 제외됩니다.
필터 표현식	쉼표로 구분된 패턴 목록. 표현식은 Maxwell 필터 구문을 사용해야 합니다.
패턴	`database.table` for Aurora MySQL 또는 `database.schema.table` for Aurora PostgreSQL 형식의 필터 패턴입니다. 리터럴 이름을 지정하거나 정규 표현식을 정의할 수 있습니다. Note Aurora MySQL의 경우 데이터베이스와 테이블 이름 모두에서 정규 표현식이 지원됩니다. Aurora PostgreSQL의 경우 정규 표현식은 데이터베이스 이름이 아닌 스키마 및 테이블 이름에서만 지원됩니다. 열 수준 필터 또는 거부 목록은 포함할 수 없습니다.

Element	설명
	단일 통합은 총 99개까지 패턴을 보유할 수 있습니다. 콘솔에서는 단일 필터 표현식 내에 패턴을 입력하거나 여러 표현식에 패턴을 분산시킬 수 있습니다. 단일 패턴은 길이가 256자를 초과할 수 없습니다.

> **Important**
>
> Aurora PostgreSQL 소스 DB 클러스터를 선택하는 경우 데이터 필터 패턴을 하나 이상 지정해야 합니다. 대상 데이터 웨어하우스로 복제하려면 패턴에 최소한 하나의 데이터베이스 (*`database-name`*`.*.*`)가 포함되어 있어야 합니다.

다음 이미지는 콘솔의 Aurora MySQL 데이터 필터 구조를 보여줍니다.



> **Important**
>
> 개인 식별 정보, 기밀 정보 또는 민감한 정보를 필터 패턴에 포함하지 마세요.

AWS CLI의 데이터 필터

AWS CLI를 사용하여 데이터 필터를 추가하는 경우 콘솔의 구문과는 약간 다릅니다. 필터 유형 (Include 또는 Exclude)을 각 패턴에 개별적으로 할당해야 하나의 필터 유형으로 여러 패턴을 그룹화할 수 있습니다.

예를 들어 콘솔에서는 쉼표로 구분된 다음과 같은 패턴을 단일 Include 스테이트먼트 내에 그룹화할 수 있습니다.

Aurora MySQL

```
mydb.mytable, mydb./table_\d+/
```

Aurora PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

하지만 AWS CLI를 사용할 때는 동일한 데이터 필터가 다음 형식이어야 합니다.

Aurora MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Aurora PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

"필터 로직"

통합에서 데이터 필터를 지정하지 않는 경우 Aurora는 기본 필터(include:*.*)를 가정하고 모든 테이블을 대상 데이터 웨어하우스에 복제합니다. 그러나 최소 하나 이상의 필터를 추가하면 기본 로직이 exclude:*.*로 전환되어 기본적으로 모든 테이블이 제외됩니다. 이를 통해 복제에 포함할 데이터베이스와 테이블을 명시적으로 정의할 수 있습니다.

예를 들어 다음을 필터를 정의하는 경우:

```
'include: db.table1, include: db.table2'
```

Aurora는 다음과 같이 필터를 평가합니다.

```
'exclude:*.*, include: db.table1, include: db.table2'
```

따라서 Aurora는 db라는 데이터베이스의 `table1` 및 `table2`만 대상 데이터 웨어하우스에 복제합니다.

필터 우선순위

Aurora는 지정된 순서대로 데이터 필터를 평가합니다. AWS Management Console에서는 필터 표현식을 왼쪽에서 오른쪽으로, 위에서 아래로 처리합니다. 두 번째 필터 또는 첫 번째 뒤에 오는 개별 패턴은 이를 재정의할 수 있습니다.

예를 들어 첫 번째 필터가 `Include books.stephenking`인 경우 books 데이터베이스의 `stephenking` 테이블만 포함됩니다. 하지만 두 번째 필터(`Exclude books.*`)를 추가하면 첫 번째 필터가 재정의됩니다. 이렇게 하면 books 인덱스의 테이블이 대상 데이터 웨어하우스로 복제되지 않습니다.

최소 1개의 필터를 지정하는 경우 로직은 가정된 `exclude:*.*`로 시작되며, 이는 모든 테이블이 복제에서 자동으로 제외된다는 의미입니다. 가장 광범위한 필터부터 가장 구체적인 필터 순으로 정의하는 것이 좋습니다. 하나 이상의 `Include` 스테이트먼트로 시작하여 복제할 데이터를 지정한 다음 `Exclude` 필터를 추가하여 특정 테이블을 선택적으로 제거합니다.

AWS CLI를 사용하여 정의하는 필터에도 동일한 원칙이 적용됩니다. Aurora는 이러한 필터 패턴을 지정된 순서대로 평가하므로 앞서 지정된 패턴을 재정의할 수 있습니다.

Aurora MySQL 예제

다음 예제에서는 Aurora MySQL 제로 ETL 통합에서 데이터 필터링이 작동하는 방식을 보여줍니다.

- 모든 데이터베이스와 모든 테이블 포함:

  ```
  'include: *.*'
  ```

- books 데이터베이스 내의 모든 테이블 포함:

  ```
  'include: books.*'
  ```

- 이름이 `mystery`인 모든 테이블 제외:

  ```
  'include: *.*, exclude: *.mystery'
  ```

- books 데이터베이스 내에서 두 개의 특정 테이블 포함:

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- mystery 하위 문자열이 포함된 테이블을 제외한 모든 테이블을 books 데이터베이스에 포함:

```
'include: books.*, exclude: books./.*mystery.*/'
```

- mystery로 시작하는 테이블을 제외한 모든 테이블을 books 데이터베이스에 포함:

```
'include: books.*, exclude: books./mystery.*/'
```

- mystery로 끝나는 테이블을 제외한 모든 테이블을 books 데이터베이스에 포함:

```
'include: books.*, exclude: books./.*mystery/'
```

- books 데이터베이스 내에서 `table_stephen_king`을 제외하고 이름이 `table_`로 시작하는 모든 테이블 포함: 예를 들어 `table_movies` 또는 `table_books`는 복제되지만 `table_stephen_king`은 복제되지 않습니다.

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

Aurora PostgreSQL 예제

다음 예제에서는 Aurora PostgreSQL 제로 ETL 통합에서 데이터 필터링이 작동하는 방식을 보여줍니다.

- books 데이터베이스 내의 모든 테이블 포함:

```
'include: books.*.*'
```

- books 데이터베이스에 `mystery` 이름이 지정된 테이블 제외:

```
'include: books.*.*, exclude: books.*.mystery'
```

- `mystery` 스키마의 books 데이터베이스 내에 테이블 1개와 `finance` 스키마의 employee 데이터베이스 내에 테이블 1개를 포함합니다.

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- king 하위 문자열이 포함된 테이블을 제외한 모든 테이블을 books 데이터베이스 및 science_fiction 스키마에 포함:

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- 스키마 이름이 sci로 시작하는 테이블을 제외한 모든 테이블을 books 데이터베이스에 포함:

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- king으로 끝나는 mystery 스키마의 테이블을 제외한 모든 테이블을 books 데이터베이스에 포함:

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- books 데이터베이스 내에서 table_stephen_king을 제외하고 이름이 table_로 시작하는 모든 테이블 포함: 예를 들어 fiction 스키마의 table_movies 및 mystery 스키마의 table_books는 복제되지만 두 스키마 table_stephen_king은 복제되지 않습니다.

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

통합에 데이터 필터 추가

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 데이터 필터링을 구성할 수 있습니다.

> **Important**
>
> 통합을 만든 후 필터를 추가하면 Aurora가 필터가 항상 존재했던 것처럼 이를 처리합니다. 대상 데이터 웨어하우스에 있는 데이터 중 새로운 필터링 기준과 일치하지 않는 데이터를 제거하고 모든 영향받는 테이블을 재동기화합니다.

RDS 콘솔

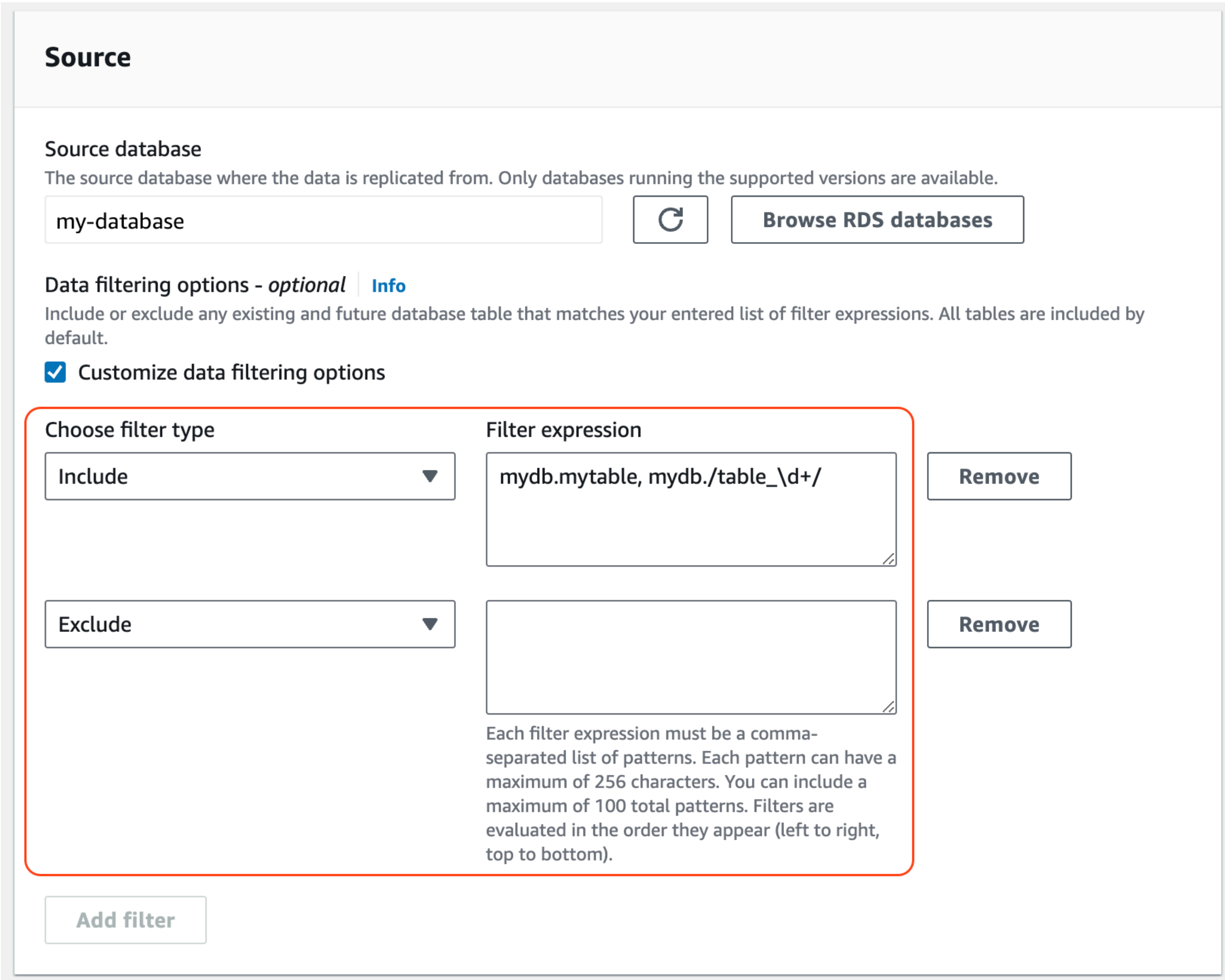
제로 ETL 통합에 데이터 필터를 추가하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

2. 탐색 창에서 제로 ETL 통합을 선택합니다. 데이터 필터를 추가할 통합을 선택한 다음 수정을 선택합니다.

3. 소스에서 하나 이상의 `Include` 및 `Exclude` 문을 추가합니다.

   다음 이미지는 MySQL 통합의 데이터 필터 예시를 보여줍니다.



4. 변경 사항에 만족하면 계속 및 변경 사항 저장을 선택합니다.

AWS CLI

AWS CLI를 사용하여 제로 ETL 통합에 데이터 필터를 추가하려면 [modify-integration](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-integration.html) 명령을 직접 호출하세요. 통합 식별자 외에도 쉼표로 구분된 `Include` 및 `Exclude` Maxwell 필터 목록을 사용하여 `--data-filter` 파라미터를 지정하세요.

Example

다음 예시에서는 `my-integration`에 필터 패턴을 추가합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

Windows의 경우:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS API

RDS API를 사용하여 제로 ETL 통합을 수정하려면 ModifyIntegration을 호출하세요. 통합 식별자를 지정하고 쉼표로 구분된 필터 패턴 목록을 제공하세요.

통합에서 데이터 필터 제거

통합에서 데이터 필터를 제거하면 Aurora는 제거된 필터가 존재하지 않았던 것처럼 나머지 필터를 재평가합니다. 그런 다음, 이전에 제외되었지만 이제 기준을 충족하는 데이터를 대상 데이터 웨어하우스에 복제합니다. 그러면 모든 영향받는 테이블의 재동기화가 트리거됩니다.

소스 Aurora DB 클러스터에 데이터 추가 및 Amazon Redshift에서 쿼리

Amazon Aurora에서 Amazon Redshift로 데이터를 복제하는 제로 ETL 통합 생성을 완료하려면 Amazon Redshift에서 대상 데이터베이스를 생성해야 합니다.

먼저 Amazon Redshift 클러스터 또는 작업 그룹에 연결하고 통합 식별자에 대한 참조가 있는 데이터베이스를 생성합니다. 그런 다음 소스 Aurora DB 클러스터에 데이터를 추가하고 Amazon Redshift에서 복제된 것을 확인할 수 있습니다.

주제

- Amazon Redshift에서 대상 데이터베이스 생성
- 소스 DB 클러스터에 데이터 추가

- Amazon Redshift에서 Aurora 데이터 쿼리
- Aurora와 Amazon Redshift 데이터베이스 간의 데이터 유형 차이

Amazon Redshift에서 대상 데이터베이스 생성

통합을 생성한 후에 Amazon Redshift로 데이터 복제를 시작하려면 먼저 대상 데이터 웨어하우스에 대상 데이터베이스를 만들어야 합니다. 이 대상 데이터베이스에는 통합 식별자에 대한 참조가 포함되어야 합니다. Amazon Redshift 콘솔 또는 쿼리 편집기 v2를 사용하여 데이터베이스를 생성할 수 있습니다.

대상 데이터베이스를 생성하는 방법에 대한 지침은 Amazon Redshift에서 대상 데이터베이스 생성을 참조하세요.

소스 DB 클러스터에 데이터 추가

통합을 구성한 후, Amazon Redshift 데이터 웨어하우스로 복제하려는 일부 데이터를 Aurora DB 클러스터에 추가합니다.

> **Note**
>
> Amazon Aurora와 Amazon Redshift의 데이터 유형 간에는 차이가 있습니다. 데이터 유형 매핑 표는 the section called “데이터 형식 차이” 섹션을 참조하세요.

먼저, 선택한 MySQL 또는 PostgreSQL 클라이언트를 사용하여 소스 DB 클러스터에 연결합니다. 지침은 the section called “DB 클러스터에 연결” 단원을 참조하십시오.

그런 다음 테이블을 만들고 샘플 데이터 행을 삽입합니다.

> **Important**
>
> 테이블에 프라이머리 키가 있는지 확인하세요. 없는 경우 대상 데이터 웨어하우스에 복제할 수 없습니다.

pg_restore 및 pg_restore PostgreSQL 유틸리티는 처음에 프라이머리 키가 없는 테이블을 만든 다음 나중에 추가합니다. 이러한 유틸리티 중 하나를 사용하는 경우 먼저 스키마를 만든 다음 별도의 명령으로 데이터를 로드하는 것이 좋습니다.

MySQL

다음 예제에서는 MySQL Workbench 유틸리티를 사용합니다.

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

다음 예제에서는 psql PostgreSQL 대화형 터미널을 사용합니다. 클러스터에 연결할 때는 통합을 생성할 때 지정한 명명된 데이터베이스를 포함하세요.

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, "The Shining", "Stephen King", 1977,
 "Supernatural fiction");
```

Amazon Redshift에서 Aurora 데이터 쿼리

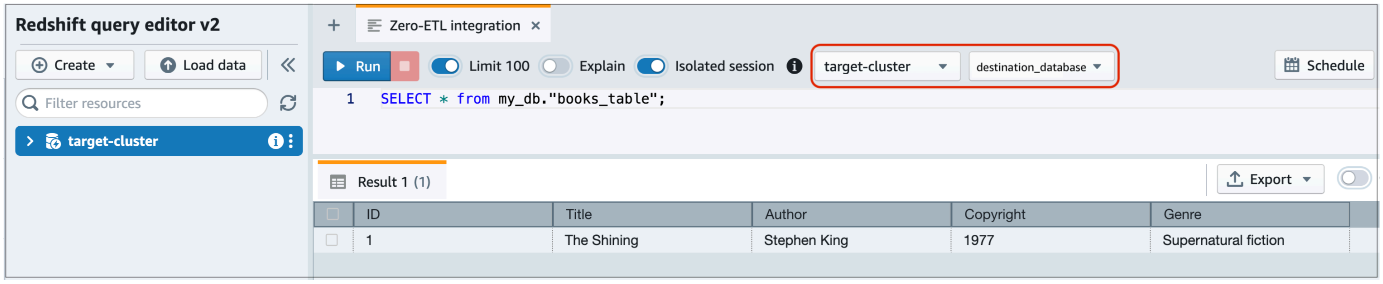
Aurora DB 클러스터에 데이터를 추가하면 데이터가 Amazon Redshift에 복제되어 쿼리할 준비가 됩니다.

복제된 데이터를 쿼리하려면

1. Amazon Redshift 콘솔로 이동한 다음 왼쪽 탐색 창에서 쿼리 편집기 v2를 선택합니다.
2. 클러스터 또는 작업 그룹에 연결하고 드롭다운 메뉴(이 예시에서는 destination_database)에서 대상 데이터베이스(통합에서 생성한 데이터베이스)를 선택합니다. 대상 데이터베이스를 생성하는 방법에 대한 지침은 Amazon Redshift에서 대상 데이터베이스 생성을 참조하세요.

3. SELECT 문을 사용하여 데이터를 쿼리합니다. 이 예제에서는 다음 명령을 실행하여 소스 Aurora DB 클러스터에서 생성한 테이블의 모든 데이터를 선택합니다.

```
SELECT * from my_db."books_table";
```



- *my_db*는 Aurora 데이터베이스 스키마 이름입니다. 이 옵션은 MySQL 데이터베이스에만 필요합니다.
- *books_table*은 Aurora 테이블 이름입니다.

명령줄 클라이언트를 사용하여 데이터를 쿼리할 수도 있습니다. 예:

```
destination_database=# select * from my_db."books_table";

 ID |       Title |          Author |  Copyright |                   Genre |  txn_seq |
 txn_id
----+-------------+----------------+-------------+-------------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> 대/소문자를 구분하려면 스키마, 테이블 및 열 이름에 큰따옴표(" ")를 사용합니다. 자세한 내용은 enable_case_sensitive_identifier 단원을 참조하세요.

Aurora와 Amazon Redshift 데이터베이스 간의 데이터 유형 차이

다음 표에는 해당하는 Amazon Redshift 데이터 유형에 대응하는 Aurora MySQL 또는 Aurora PostgreSQL 데이터 유형의 매핑이 나와 있습니다. Amazon Aurora는 현재 제로 ETL 통합에 이러한 데이터 유형만 지원합니다.

소스 DB 클러스터의 테이블에 지원되지 않는 데이터 유형이 포함된 경우 테이블이 동기화되지 않아 Amazon Redshift 대상에서 사용할 수 없습니다. 소스에서 대상으로의 스트리밍은 계속되지만 지원되지 않는 데이터 유형이 있는 테이블은 사용할 수 없습니다. 테이블을 수정하고 Amazon Redshift에서 사용할 수 있게 하려면 주요 변경 사항을 수동으로 되돌린 다음 `ALTER DATABASE...INTEGRATION REFRESH`를 실행하여 통합을 새로 고쳐야 합니다.

주제

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Aurora MySQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
INT	INTEGER	4바이트 부호화 정수	
SMALLINT	SMALLINT	2바이트 부호화 정수	
TINYINT	SMALLINT	2바이트 부호화 정수	
MEDIUMINT	INTEGER	4바이트 부호화 정수	
BIGINT	BIGINT	8바이트 부호화 정수	
INT UNSIGNED	BIGINT	8바이트 부호화 정수	
TINYINT UNSIGNED	SMALLINT	2바이트 부호화 정수	
MEDIUMINT UNSIGNED	INTEGER	4바이트 부호화 정수	

Aurora MySQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
BIGINT UNSIGNED	DECIMAL(20,0)	정밀도를 선택할 수 있는 정확한 숫자	
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL (p,s)	정밀도를 선택할 수 있는 정확한 숫자	38보다 큰 정밀도 및 37보다 큰 확장은 지원되지 않음
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL (p,s)	정밀도를 선택할 수 있는 정확한 숫자	38보다 큰 정밀도 및 37보다 큰 확장은 지원되지 않음
FLOAT4/REAL	REAL	단정밀도 부동 소수점 수	
FLOAT4/REAL UNSIGNED	REAL	단정밀도 부동 소수점 수	
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	배정밀도 부동 소수점 수	
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	배정밀도 부동 소수점 수	
BIT(n)	VARBYTE(8)	가변 길이 이진 값	
BINARY(n)	VARBYTE(n)	가변 길이 이진 값	
VARBINARY(n)	VARBYTE(n)	가변 길이 이진 값	

Aurora MySQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
CHAR(n)	VARCHAR(n)	길이가 가변적인 문자열 값	
VARCHAR(n)	VARCHAR(n)	길이가 가변적인 문자열 값	
TEXT	VARCHAR(65,535)	최대 65,535바이트의 길이가 가변적인 문자열 값	
TINYTEXT	VARCHAR(255)	최대 255바이트의 길이가 가변적인 문자열 값	
MEDIUMTEXT	VARCHAR(65,535)	최대 65,535바이트의 길이가 가변적인 문자열 값	
LONGTEXT	VARCHAR(65,535)	최대 65,535바이트의 길이가 가변적인 문자열 값	
ENUM	VARCHAR(1,020)	최대 1,020바이트의 길이가 가변적인 문자열 값	
SET	VARCHAR(1,020)	최대 1,020바이트의 길이가 가변적인 문자열 값	
날짜	날짜	날짜(년, 월, 일)	
DATETIME	TIMESTAMP	날짜/시간(시간대 제외)	

Aurora MySQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
TIMESTAMP(p)	TIMESTAMP	날짜/시간(시간대 제외)	
TIME	VARCHAR(18)	최대 18바이트의 길이가 가변적인 문자열 값	
YEAR	VARCHAR(4)	최대 4바이트의 길이가 가변적인 문자열 값	
JSON	SUPER	반정형 데이터 또는 문서 값	

Aurora PostgreSQL

Aurora PostgreSQL의 제로 ETL 통합은 사용자 지정 데이터 유형이나 확장에서 만든 데이터 유형을 지원하지 않습니다.

> **Important**
>
> Aurora PostgreSQL의 Amazon Redshift 기능과의 제로 ETL 통합은 미리 보기 릴리스에 포함되어 있습니다. 설명서 및 기능은 모두 변경될 수 있습니다. 이 기능은 프로덕션 환경이 아닌 테스트 환경에서만 사용할 수 있습니다. 미리 보기 이용 약관은 AWS 서비스 약관의 베타 및 미리 보기를 참조하세요.

Aurora PostgreSQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
bigint	BIGINT	8바이트 부호화 정수	

Aurora PostgreSQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
bigserial	BIGINT	8바이트 부호화 정수	
bit(n)	VARBYTE(n)	가변 길이 이진 값	
bit varying(n)	VARBYTE(n)	가변 길이 이진 값	
bit	VARBYTE(1,024,000)	최대 1,024,000 바이트의 길이가 가변적인 문자열 값	
boolean	BOOLEAN	논리적 부울(true/false)	
bytea	VARBYTE(1,024,000)	최대 1,024,000 바이트의 길이가 가변적인 문자열 값	
character(n)	CHAR(n)	고정 길이 문자열	
character varying(n)	VARCHAR(65,535)	길이가 가변적인 문자열 값	
date	날짜	날짜(년, 월, 일)	• 지원되지 않는 9999-12-31 값보다 큰 값 • B.C. 값은 지원되지 않음
double precision	DOUBLE PRECISION	배정밀도 부동 소수점 수	비정상 값은 지원되지 않음

Aurora PostgreSQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
정수	INTEGER	4바이트 부호화 정수	
money	DECIMAL(203)	통화 금액	
numeric(p,s)	DECIMAL (p,s)	길이가 가변적인 문자열 값	• NaN 값은 지원되지 않음 • 38보다 큰 정밀도 및 37보다 큰 확장은 지원되지 않음 • 네거티브 확장은 지원되지 않음
실제	REAL	단정밀도 부동 소수점 수	
smallint	SMALLINT	2바이트 부호화 정수	
smallserial	SMALLINT	2바이트 부호화 정수	
serial	INTEGER	4바이트 부호화 정수	
텍스트	VARCHAR(65,535)	최대 65,535바이트의 길이가 가변적인 문자열 값	
time [(p)](시간대 제외)	VARCHAR(19)	최대 19바이트의 길이가 가변적인 문자열 값	`Infinity` 및 `-Infinity` 값은 지원되지 않음

Aurora PostgreSQL 데이터 유형	Amazon Redshift 데이터 형식	설명	제한 사항
time [(p)](시간대 포함)	VARCHAR(22)	최대 22바이트의 길이가 가변적인 문자열 값	• `Infinity` 및 `-Infinity` 값은 지원되지 않음
timestamp [(p)](시간대 제외)	TIMESTAMP	날짜/시간(시간대 제외)	• `Infinity` 및 `-Infinity` 값은 지원되지 않음 • 지원되지 않는 `9999-12-31` 값보다 큰 값 • B.C. 값은 지원되지 않음
timestamp [(p)](시간대 포함)	TIMESTAMPTZ	날짜/시간(시간대 포함)	• `Infinity` 및 `-Infinity` 값은 지원되지 않음 • 지원되지 않는 `9999-12-31` 값보다 큰 값 • B.C. 값은 지원되지 않음

Aurora 제로 ETL 통합 확인 및 모니터링

Amazon Aurora 제로 ETL 통합의 세부 정보를 보고 구성 정보와 현재 상태를 확인할 수 있습니다. Amazon Redshift에서 특정 시스템 보기를 쿼리하여 통합 상태를 모니터링할 수도 있습니다. 또한 Amazon Redshift는 특정 통합 관련 지표를 Amazon CloudWatch에 게시하며, 이를 Amazon Redshift 콘솔에서 확인할 수 있습니다.

주제

- [통합 보기]
- [Amazon Redshift의 시스템 테이블을 사용한 통합 모니터링]
- [Amazon EventBridge로 Amazon Redshift에 대한 통합 모니터링]

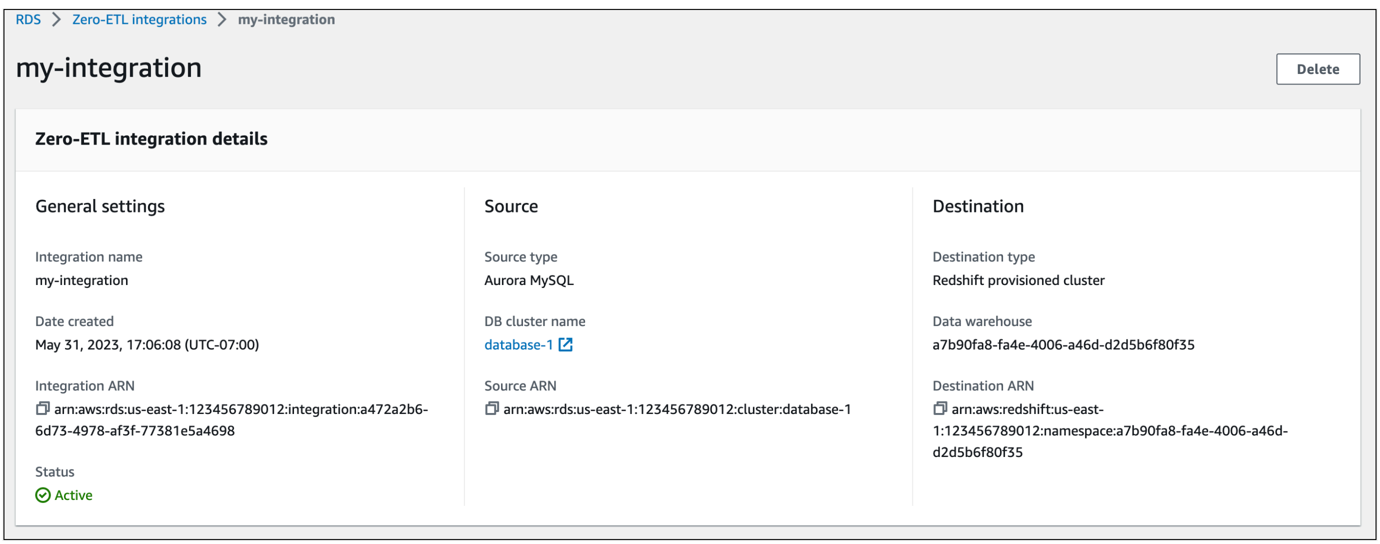
통합 보기

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora 제로 ETL 통합을 볼 수 있습니다.

콘솔

제로 ETL 통합 세부 정보를 보려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 제로 ETL 통합을 선택합니다.
3. 통합을 선택하면 소스 DB 클러스터 및 대상 데이터 웨어하우스와 같은 통합 세부 정보를 볼 수 있습니다.



통합에는 다음과 같은 상태가 있을 수 있습니다.

- `Creating` - 통합이 생성 중입니다.
- `Active` - 통합이 트랜잭션 데이터를 대상 데이터 웨어하우스로 전송하고 있습니다.
- `Syncing` - 통합에 복구 가능한 오류가 발생하여 데이터를 다시 시드하고 있습니다. 영향을 받는 테이블은 재동기화가 완료될 때까지 쿼리할 수 없습니다.

- `Needs attention` - 통합에 수동 개입이 필요한 이벤트 또는 오류가 발생하여 이를 해결해야 합니다. 문제를 해결하기 위해 통합 세부 정보 페이지에 있는 오류 메시지의 지침을 따릅니다.
- `Failed` - 통합에서 복구할 수 없는 이벤트 또는 수정할 수 없는 오류가 발생했습니다. 통합을 삭제하고 다시 만들어야 합니다.
- `Deleting` - 통합이 삭제 중입니다.

AWS CLI

AWS CLI를 사용하여 현재 계정의 모든 제로 ETL 통합을 보려면 describe-integration 명령을 사용하고 `--integration-identifier` 옵션을 지정하세요.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Windows의 경우:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

Amazon RDS API를 사용하여 제로 ETL 통합을 보려면 `IntegrationIdentifier` 파라미터와 함께 `DescribeIntegrations` 작업을 사용하세요.

Amazon Redshift의 시스템 테이블을 사용한 통합 모니터링

Amazon Redshift에는 시스템 작동 방식에 대한 정보가 저장되어 있는 시스템 테이블 및 보기가 있습니다. 이러한 시스템 테이블 및 보기 역시 다른 데이터베이스 테이블에 대한 쿼리와 동일한 방법으로 쿼리를 실행할 수 있습니다. Amazon Redshift의 시스템 테이블 및 보기에 대한 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서의 시스템 테이블 및 뷰 참조를 참조하세요.

다음 시스템 뷰와 테이블을 쿼리하여 Aurora 제로 ETL 통합에 대한 정보를 얻을 수 있습니다.

- SVV_INTEGRATION – 통합에 대한 구성 세부 정보를 제공합니다.
- SVV_INTEGRATION_TABLE_STATE – 통합 내 각 테이블의 상태를 설명합니다.

- SYS_INTEGRATION_TABLE_STATE_CHANGE - 통합에 대한 테이블 상태 변경 로그를 표시합니다.
- SYS_INTEGRATION_ACTIVITY – 완료된 통합 실행에 대한 정보를 제공합니다.

모든 통합 관련 Amazon CloudWatch 지표의 출처는 Amazon Redshift입니다. 자세한 내용은 Amazon Redshift 관리 안내서의 제로 ETL 통합의 지표를 참조하세요. 현재 Amazon Aurora는 Amazon CloudWatch에 통합 지표를 게시하지 않습니다.

Amazon EventBridge로 Amazon Redshift에 대한 통합 모니터링

Amazon Redshift는 통합 관련 이벤트를 Amazon EventBridge로 보냅니다. 이벤트 및 해당 이벤트 ID의 목록은 Amazon Redshift 관리 가이드의 Amazon EventBridge를 사용한 제로 ETL 통합 이벤트 알림을 참조하세요.

Aurora 제로 ETL 통합 수정

지원되는 데이터 웨어하우스에서 제로 ETL 통합의 이름, 설명 및 데이터 필터링 옵션만 수정할 수 있습니다. 통합을 암호화하는 데 사용된 AWS KMS 키나 소스 또는 대상 데이터베이스는 수정할 수 없습니다.

기존 통합에 데이터 필터를 추가하면 Aurora가 필터가 항상 존재했던 것처럼 필터를 재평가합니다. 현재 대상 데이터 웨어하우스에 있는 데이터 중 새로운 필터링 기준과 일치하지 않는 모든 데이터를 제거합니다. 통합에서 데이터 필터를 제거하면 이전에는 필터링 기준과 일치하지 않았던(단, 지금은 일치) 모든 데이터가 대상 데이터 웨어하우스에 복제됩니다. 자세한 내용은 the section called “제로 ETL 통합의 데이터 필터링” 섹션을 참조하세요.

AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 제로 ETL 통합을 수정할 수 있습니다.

RDS 콘솔

제로 ETL 통합을 수정하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 제로 ETL 통합을 선택한 다음 수정하려는 통합을 선택합니다.
3. 수정을 선택하고 사용 가능한 설정을 수정합니다.
4. 원하는 대로 모두 변경되었으면 수정을 선택합니다.

AWS CLI

AWS CLI를 사용하여 제로 ETL 통합을 수정하려면 modify-integration 명령을 직접 호출하세요. --integration-identifier와 함께 다음 옵션 중 하나를 지정하세요.

- `--integration-name` - 통합의 새로운 이름을 지정합니다.
- `--description` - 통합의 새로운 설명을 지정합니다.
- `--data-filter` - 통합의 데이터 필터링 옵션을 지정합니다. 자세한 내용은 the section called “제로 ETL 통합의 데이터 필터링” 섹션을 참조하세요.

Example

다음 요청은 기존 통합을 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

Windows의 경우:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS API

RDS API를 사용하여 제로 ETL 통합을 수정하려면 ModifyIntegration을 호출하세요. 통합 식별자 및 수정하려는 파라미터를 지정합니다.

Aurora 제로 ETL 통합 삭제

제로 ETL 통합을 삭제할 때 Amazon Aurora가 소스 Aurora DB 클러스터에서 해당 통합을 제거합니다. 트랜잭션 데이터는 Amazon Aurora나 분석 대상에서 삭제되지 않지만, Aurora는 Amazon Redshift 또는 Amazon SageMaker로 새 데이터를 전송하지 않습니다.

상태가 `Active`, `Failed`, `Syncing` 또는 `Needs attention`인 통합만 삭제할 수 있습니다.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 제로 ETL 통합을 삭제할 수 있습니다.

콘솔

제로 ETL 통합을 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 제로 ETL 통합을 선택합니다.
3. 삭제하려는 제로 ETL 통합을 선택합니다.
4. 작업, 삭제를 차례로 선택한 후 삭제를 확인합니다.

AWS CLI

제로 ETL 통합을 삭제하려면 delete-integration 명령을 사용하고 `--integration-identifier` 옵션을 지정합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Windows의 경우:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

Amazon RDS API를 사용하여 제로 ETL 통합을 삭제하려면 `IntegrationIdentifier` 파라미터와 함께 `DeleteIntegration` 작업을 사용합니다.

Amazon Redshift가 구성된 Aurora 제로 ETL 통합 문제 해결

Amazon Redshift에서 SVV_INTEGRATION 시스템 테이블을 쿼리하여 제로 ETL 통합의 상태를 확인할 수 있습니다. `state` 열의 값이 `ErrorState`면 문제가 있다는 뜻입니다. 자세한 내용은 the section called "시스템 테이블을 사용한 모니터링" 단원을 참조하십시오.

다음 정보를 사용하여 Amazon Redshift가 구성된 Aurora 제로 ETL 통합과 관련된 일반적인 문제를 해결하세요.

주제

제로 ETL 통합을 생성할 수 없습니다

제로 ETL 통합을 생성할 수 없는 경우 소스 DB 클러스터에 다음 사항이 올바른지 확인하세요.

- 소스 DB 클러스터는 Aurora MySQL 버전 3.05(MySQL 8.0.32와 호환) 이상 또는 Aurora PostgreSQL(PostgreSQL 15.4 및 제로 ETL 지원과 호환)을 실행해야 합니다. 엔진 버전을 확인하려면 DB 클러스터의 구성 탭을 선택하여 엔진 버전을 확인하세요.
- DB 클러스터 파라미터를 올바르게 구성했습니다. 필수 파라미터가 잘못 설정되었거나 클러스터와 연결되지 않은 경우 생성에 실패합니다. the section called "1단계: 사용자 지정 DB 클러스터 파라미터 그룹 생성" 섹션을 참조하세요.

또한 대상 데이터 웨어하우스에 대해 다음 사항이 올바른지 확인하세요.

- 대소문자 구분이 활성화되어 있습니다. 데이터 웨어하우스에 대소문자 구분 기능 사용 설정을 참조하세요.
- 올바른 권한 있는 보안 주체 및 통합 소스를 추가했습니다. Amazon Redshift 데이터 웨어하우스에 대한 권한 구성을 참조하세요.
- 데이터 웨어하우스는 암호화되어 있습니다(프로비저닝된 클러스터인 경우). Amazon Redshift 데이터베이스 암호화를 참조하세요.

내 통합이 Syncing 상태에서 멈췄습니다.

필수 DB 클러스터 파라미터 중 하나의 값을 변경하면 통합에서 Syncing 상태가 일관되게 표시될 수 있습니다.

이 문제를 해결하려면 소스 DB 클러스터와 연결된 파라미터 그룹의 파라미터 값을 살펴보고, 필수 값과 일치하는지 확인하세요. 자세한 내용은 the section called “1단계: 사용자 지정 DB 클러스터 파라미터 그룹 생성” 단원을 참조하십시오.

파라미터를 수정할 경우 DB 클러스터를 재부팅하여 변경 사항을 적용해야 합니다.

내 테이블이 Amazon Redshift에 복제되지 않는 경우

Amazon Redshift에 하나 이상의 표가 반영되지 않는 경우 다음 명령을 실행하여 표를 다시 동기화할 수 있습니다.

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

자세한 내용은 Amazon Redshift SQL 참조의 ALTER DATABASE를 참조하세요.

하나 이상의 원본 테이블에 프라이머리 키가 없어 데이터가 복제되지 않을 수 있습니다. Amazon Redshift의 모니터링 대시보드에는 이러한 테이블의 상태가 Failed로 표시되고 전체 제로 ETL 통합 상태는 Needs attention으로 변경됩니다. 이 문제를 해결하려면 테이블에서 프라이머리 키가 될 수 있는 기존 키를 식별하거나 가상 프라이머리 키를 추가할 수 있습니다. 자세한 해결 방법은 를 참조하세요. 다음 리소스를 확인하세요.

- Amazon Redshift를 사용해 Amazon Aurora MySQL 또는 Amazon RDS for MySQL 제로 ETL 통합을 생성하는 동안 프라이머리 키가 없는 테이블 처리
- Amazon Redshift를 사용해 Amazon Aurora PostgreSQL 제로 ETL 통합을 생성하는 동안 프라이머리 키가 없는 테이블 처리

Amazon Redshift 테이블 중 하나 이상을 재동기화해야 합니다

소스 DB 클러스터에서 특정 명령을 실행하려면 테이블을 재동기화해야 할 수 있습니다. 이러한 경우 SVV_INTEGRATION_TABLE_STATE 시스템 뷰에 ResyncRequired의 table_state가 표시됩니다. MySQL의 해당 테이블에서 Amazon Redshift로 데이터를 완전히 다시 로드해야 한다는 의미입니다.

테이블이 재동기화되기 시작하면 `Syncing` 상태가 됩니다. 테이블을 재동기화하기 위해 수동 작업을 수행할 필요가 없습니다. 테이블 데이터가 재동기화되는 동안에는 Amazon Redshift에서 액세스하지 못할 수도 있습니다.

다음은 테이블을 `ResyncRequired` 상태로 전환할 수 있는 몇 가지 예제 작업과 고려할 수 있는 대안입니다.

<table>
<tr><th>Operation</th><th>예</th><th>대안</th></tr>
<tr><td>특정 위치에 열 추가</td><td>

```
ALTER TABLE table_name
  ADD COLUMN column_name  INTEGER
  NOT NULL first;
```

</td><td>Amazon Redshift는 <code>first</code> 또는 <code>after</code> 키워드를 사용하여 특정 위치에 열을 추가하는 것을 지원하지 않습니다. 대상 테이블의 열 순서가 중요하지 않은 경우 다음과 같은 간단한 명령을 사용하여 테이블 끝에 열을 추가합니다.

```
ALTER
 TABLE table_name
 ADD
 COLUMN column_name  column_type ;
```

</td></tr>
<tr><td>기본값 CURRENT_T</td><td>

```
ALTER TABLE table_name
  ADD COLUMN column_name  TIMESTAMP
```

</td><td>기존 표 행의 CURRENT_T</td></tr>
</table>

Operation	예	대안
IMESTAMP 로 타임스탬프 열 추가	`NOT NULL DEFAULT CURRENT_TIMESTAMP;`	IMESTAMP 값은 Aurora MySQL에서 계산되며, 전체 표 데이터를 재동기화하지 않고는 Amazon Redshift에서 시뮬레이션할 수 없습니다. 가능하면 기본값을 리터럴 상수(예: `2023-01-01 00:00:15`)로 전환하여 테이블 가용성에 지연 시간이 생기지 않도록 하세요.

단일 명령 내에서 여러 열 작업 수행	`ALTER TABLE table_name`
	`  ADD COLUMN column_1,`
	`  RENAME COLUMN column_2 TO column_3;`
		명령을 2개의 개별 작업(ADD, RENAME)으로 분할하여 재동기화할 필요가 없도록 하는 것이 좋습니다.

Aurora Serverless v2 사용하기

Aurora Serverless v2는 Amazon Aurora에 대한 온디맨드 방식의 자동 크기 조정 구성입니다. Aurora Serverless v2는 워크로드를 모니터링하고 데이터베이스의 용량을 조정하는 프로세스를 자동화하는 데 도움이 됩니다. 용량은 애플리케이션 수요에 따라 자동으로 조정됩니다. DB 클러스터가 사용하는 리소스에 대해서만 청구됩니다. 따라서 Aurora Serverless v2는 예산 범위를 벗어나지 않도록 하고 사용하지 않는 컴퓨터 리소스에 대한 비용을 지불하지 않도록 하는 데 도움이 됩니다.

이러한 유형의 자동화는 멀티테넌트 데이터베이스, 분산 데이터베이스, 개발 및 테스트 시스템, 가변적이고 예측할 수 없는 워크로드가 있는 기타 환경에 특히 유용합니다.

주제

Aurora Serverless v2 사용 사례

Aurora Serverless v2는 여러 유형의 데이터베이스 워크로드를 지원합니다. 개발 및 테스트 환경부터 워크로드를 예측할 수 없는 웹 사이트 및 애플리케이션, 높은 확장성과 가용성이 필요한 가장 까다로운 비즈니스 크리티컬 애플리케이션에 이르기까지 다양합니다.

Aurora Serverless v2는 다음과 같은 사용 사례에 특히 유용합니다.

- 가변 워크로드 – 작업의 증가가 갑작스럽고 예측할 수 없는 워크로드를 실행하고 있는 경우입니다. 비가 내리기 시작하면 활동이 급증하는 트래픽 사이트를 예로 들 수 있습니다. 또 다른 하나는 세일 또는 특별 프로모션을 제공할 때 트래픽이 증가하는 전자 상거래 사이트입니다. Aurora Serverless v2를 사용하면 애플리케이션의 피크 로드를 처리하는 데 필요한 용량을 충족하도록 데이터베이

스가 용량을 자동으로 확장하고 활동이 급증하는 시점이 지나면 용량을 다시 줄입니다. Aurora Serverless v2를 사용하면 더 이상 피크 또는 평균 용량에 맞추어 프로비저닝하지 않아도 됩니다. 최악의 상황을 처리하기 위해 용량 상한을 지정할 수 있으며, 필요한 경우가 아니면 그 용량이 사용되지 않습니다.

Aurora Serverless v2의 세분화된 크기 조정 덕분에 데이터베이스 요구 사항과 용량을 매우 비슷하게 일치시키는 데 도움이 됩니다. 프로비저닝된 클러스터의 경우 확장하려면 완전히 새로운 DB 인스턴스를 추가해야 합니다. Aurora Serverless v1 클러스터를 확장하려면 클러스터의 Aurora 용량 단위(ACU)의 수를 배로 늘려야 합니다(예: 16에서 32로, 32에서 64로). 대조적으로, Aurora Serverless v2의 경우 용량이 조금 더 필요할 때 ACU의 절반을 추가할 수 있습니다. 워크로드의 증가를 처리하는 데 필요한 추가 용량에 따라 ACU의 0.5, 1, 1.5, 2와 같이 0.5 단위로 ACU를 추가할 수 있습니다. 또한 워크로드가 감소하고 용량이 더 이상 필요하지 않을 때 0.5, 1, 1.5, 2와 같이 0.5 단위로 ACU를 제거할 수 있습니다.

- 다중 테넌트 애플리케이션 - Aurora Serverless v2을 사용하면 플릿에서 각 애플리케이션에 대한 데이터베이스 용량을 개별적으로 관리할 필요가 없습니다. Aurora Serverless v2은 사용자를 대신하여 개별 데이터베이스 용량을 관리해 줍니다.

  테넌트별로 클러스터를 생성할 수 있습니다. 이렇게 하면 복제, 스냅샷 복구 및 Aurora 글로벌 데이터베이스와 같은 기능을 사용하여 각 테넌트에 적합하도록 고가용성 및 재해 복구를 향상시킬 수 있습니다.

  각 테넌트는 하루 중 시간, 연중 시기, 프로모션 이벤트 등에 따라 바쁜 기간과 유휴 기간이 있을 수 있습니다. 각 클러스터는 용량 범위가 넓을 수 있습니다. 다라서 활동이 적은 클러스터는 DB 인스턴스 요금이 최소화됩니다. 어느 클러스터든 높은 활동 기간을 처리하도록 빠르게 확장할 수 있습니다.
- 새 애플리케이션 – 새 애플리케이션을 배포하는 중인데, 필요한 DB 인스턴스 크기를 잘 모를 경우입니다. Aurora Serverless v2를 사용하면 DB 인스턴스를 하나 이상 사용하여 클러스터를 설정하고 애플리케이션의 필요 용량에 따라 데이터베이스의 크기를 자동으로 조정할 수 있습니다.
- 혼용 애플리케이션 - 온라인 트랜잭션 처리(OLTP) 애플리케이션이 있지만 쿼리 트래픽이 주기적으로 급증한다고 가정합시다. 클러스터에서 Aurora Serverless v2에 대한 승격 티어를 지정하면 리더 DB 인스턴스가 라이터 DB 인스턴스와 독립적으로 확장되어 추가 로드를 처리할 수 있도록 클러스터를 구성할 수 있습니다. 사용량이 줄어들면 리더 DB 인스턴스는 라이터 DB 인스턴스의 용량에 맞게 축소됩니다.
- 용량 계획 - 클러스터에 있는 모든 DB 인스턴스의 DB 인스턴스 클래스를 수정하여 일반적으로 데이터베이스 용량을 조정하거나 워크로드에 가장 적합한 데이터베이스 용량을 확인한다고 가정해 보겠습니다. Aurora Serverless v2를 사용하면 이런 관리 오버헤드를 예방할 수 있습니다. 워크로드를 실행하고 DB 인스턴스가 실제로 얼마나 조정되는지 확인하여 적절한 최소 및 최대 용량을 결정할 수 있습니다.

기존 DB 인스턴스를 프로비저닝에서 Aurora Serverless v2로 또는 Aurora Serverless v2에서 프로비저닝으로 수정할 수 있습니다. 이런 경우 새 클러스터나 새 DB 인스턴스를 생성할 필요가 없습니다.

Aurora 글로벌 데이터베이스를 사용하는 경우 보조 클러스터는 기본 클러스터만큼 용량이 많이 필요하지 않을 수 있습니다. 보조 클러스터에서는 Aurora Serverless v2 DB 인스턴스를 사용하면 됩니다. 그러면 보조 리전이 승격되어 애플리케이션의 워크로드를 인계할 경우 클러스터 용량이 확장될 수 있습니다.

- 개발 및 테스트 - 가장 까다로운 애플리케이션을 실행하는 것 외에도 개발 및 테스트 환경에 Aurora Serverless v2를 사용할 수 있습니다. Aurora Serverless v2를 사용하면 버스트 가능한 db.t* DB 인스턴스 클래스를 사용하는 대신 최소 용량이 낮은 DB 인스턴스를 생성할 수 있습니다. 최대 용량을 충분히 높게 설정하여 해당 DB 인스턴스에서 메모리가 부족하지 않게 상당한 워크로드를 실행할 수 있습니다. 데이터베이스를 사용하지 않을 때는 불필요한 요금을 피하기 위해 모든 DB 인스턴스가 스케일 다운됩니다.

> **Tip**
>
> 개발 및 테스트 환경에서 Aurora Serverless v2를 편리하게 사용할 수 있도록 AWS Management Console에서는 새 클러스터를 생성할 때 쉬운 생성(Easy create) 단축키를 제공합니다. 개발 및 테스트(Dev/Test) 옵션을 선택하면 Aurora는 Aurora Serverless v2 DB 인스턴스와 함께 개발 및 테스트 시스템에서 일반적으로 사용되는 용량 범위로 클러스터를 생성합니다.

기존 프로비저닝된 워크로드에 Aurora Serverless v2 사용

프로비저닝된 클러스터에서 이미 Aurora 애플리케이션이 실행 중이라고 가정합시다. 기존 클러스터에 리더 DB 인스턴스로 하나 이상의 Aurora Serverless v2 DB 인스턴스를 추가하여 애플리케이션이 Aurora Serverless v2와 어떻게 작동하는지 확인할 수 있습니다. 리더 DB 인스턴스가 얼마나 자주 확장 및 축소되는지 확인할 수 있습니다. Aurora 장애 조치 메커니즘을 사용하여 Aurora Serverless v2 DB 인스턴스가 라이터가 되도록 승격하고 읽기/쓰기 워크로드를 어떻게 처리하는지 확인합니다. 그러면 클라이언트 애플리케이션이 사용하는 엔드포인트를 변경하지 않고도 가동 중지 시간을 최소화하면서 전환할 수 있습니다. 기존 클러스터를 Aurora Serverless v2로 변환하는 절차에 대한 자세한 내용은 [Aurora Serverless v2로 마이그레이션](#) 섹션을 참조하세요.

Aurora Serverless v2의 장점

Aurora Serverless v2는 가변 또는 '급증' 워크로드를 위한 것입니다. 예측할 수 없는 워크로드로 인해 데이터베이스 용량을 변경하는 시기를 계획하는 데 어려움이 있을 수 있습니다. 또한 DB 인스턴스 추가 또는 DB 인스턴스 클래스 변경과 같은 익숙한 메커니즘을 사용하여 용량을 빠르게 변경하는 데 문제가 있을 수 있습니다. Aurora Serverless v2는 이러한 사용 사례에 도움이 되도록 다음과 같은 이점을 제공합니다.

- 프로비저닝된 클러스터보다 간편한 용량 관리 – Aurora Serverless v2를 사용하면 워크로드 변경에 따라 DB 인스턴스 크기를 계획하고 DB 인스턴스의 크기를 조정하기 위한 노력이 줄어듭니다. 또한 클러스터에 속한 모든 DB 인스턴스에 일관된 용량을 유지하기 위한 노력도 줄여줍니다.
- 활동이 많은 기간 동안 더 빠르고 쉽게 확장 – Aurora Serverless v2는 클라이언트 트랜잭션이나 전체 워크로드에 영향을 주지 않고 필요에 따라 컴퓨팅 및 메모리 용량을 확장합니다. 리더 DB 인스턴스 Aurora Serverless v2와 함께 사용할 수 있으므로 수직 크기 조정뿐만 아니라 수평 크기 조정도 활용할 수 있습니다. Aurora 글로벌 데이터베이스를 사용할 수 있으므로 여러 개의 AWS 리전에 Aurora Serverless v2 읽기 워크로드를 분산할 수 있습니다. 이 기능은 프로비저닝된 클러스터의 크기 조정 메커니즘보다 편리합니다. 또한 Aurora Serverless v1의 크기 조정 기능보다 빠르고 세분화되어 있습니다.
- 활동이 적은 기간 동안 비용 효율적 – Aurora Serverless v2는 DB 인스턴스의 오버프로비저닝을 예방합니다. Aurora Serverless v2는 DB 인스턴스의 크기를 확장할 때 리소스를 세분화된 단위로 추가합니다. 사용한 데이터베이스 리소스에 대해서만 요금을 지불합니다. Aurora Serverless v2 리소스 사용량은 초 단위로 측정됩니다. 따라서 DB 인스턴스가 축소되면 리소스 사용량 감소가 즉시 등록됩니다.
- 프로비저닝된 클러스터로 향상된 기능 패리티 - Aurora Serverless v1에서는 사용할 수 없었던 다양한 Aurora 기능을 Aurora Serverless v2에서는 사용할 수 있습니다. 예를 들어 Aurora Serverless v2를 사용하면 리더 DB 인스턴스, 글로벌 데이터베이스, AWS Identity and Access Management(IAM) 데이터베이스 인증, 성능 개선 도우미를 사용할 수 있습니다. 사용할 수 있는 구성 매개 변수도 Aurora Serverless v1보다 많습니다.

  특히 Aurora Serverless v2를 사용하면 프로비저닝된 클러스터로부터 다음과 같은 기능을 활용할 수 있습니다.

  - 리더 DB 인스턴스 – Aurora Serverless v2는 리더 DB 인스턴스를 활용하여 수평으로 확장할 수 있습니다. 클러스터에 하나 이상의 리더 DB 인스턴스가 포함된 경우 라이터 DB 인스턴스에 문제가 발생하면 클러스터가 즉시 장애 조치를 할 수 있습니다. 이 기능은 Aurora Serverless v1에서는 사용할 수 없는 기능입니다.

  - 다중 AZ 클러스터 - 한 클러스터의 Aurora Serverless v2 DB 인스턴스를 여러 개의 가용 영역(AZ)에 배포할 수 있습니다. 다중 AZ 클러스터를 설정하면 드물게 전체 AZ에 영향을 주는 경우가 발생해도 비즈니스 연속성을 보장할 수 있습니다. 이 기능은 Aurora Serverless v1에서는 사용할 수 없는 기능입니다.
  - 글로벌 데이터베이스 - Aurora 글로벌 데이터베이스와 Aurora Serverless v2를 결합하여 재해 복구 목적으로 다른 AWS 리전에 클러스터의 읽기 전용 복사본을 생성할 수 있습니다.
  - RDS 프록시 – Amazon RDS Proxy를 사용하면 애플리케이션이 데이터베이스 연결을 풀링하고 공유하도록 허용하여 확장 기능을 향상할 수 있습니다.
- 더 빠르고, 세밀하며, 중단이 적은 크기 조정Aurora Serverless v1 – Aurora Serverless v2는 더 빠르게 확장 및 축소할 수 있습니다. ACU 수를 두 배로 늘리거나 절반으로 줄이는 대신 0.5 단위로 ACU 용량을 변경하여 크기를 조정할 수 있습니다. 크기 조정은 일반적으로 처리를 일시 중지하지 않고 수행됩니다. 크기 조정에는 Aurora Serverless v1에서처럼 주의해야 할 이벤트가 포함되지 않습니다. 사용률이 낮은 시점을 기다릴 필요 없이 SQL 문이 실행 중이고 트랜잭션이 열려 있는 동안 크기 조정이 가능합니다.

Aurora Serverless v2 작동 방식

다음 개요는 Aurora Serverless v2 작동 방식을 설명합니다.

주제

Aurora Serverless v2 개요

Amazon Aurora Serverless v2는 가장 까다롭고 매우 가변적인 워크로드에 적합합니다. 예를 들어 데이터베이스 사용량이 짧은 시간 동안 높게 나타나고 오랜 시간 동안 작업이 적거나 전혀 작업이 없을 수 있습니다. 정기 프로모션 이벤트를 진행하는 소매점, 게임 또는 스포츠 웹 사이트와 필요한 경우 보고서를 생성하는 데이터베이스가 예로 들 수 있습니다. 이외에도 개발 및 테스트 환경과 사용량이 빠르

게 증가할 수 있는 새로운 애플리케이션도 있습니다. 이 경우와 다른 많은 경우에는 프로비저닝된 모델에서 용량을 미리 적절하게 구성하지 못할 수도 있습니다. 또한 초과 프로비저닝하여 사용하지 않는 용량이 있는 경우 비용이 더 많이 발생할 수 있습니다.

대조적으로,Aurora 프로비저닝된 클러스터는 안정적인 워크로드에 적합합니다. 프로비저닝된 클러스터에서는 미리 정의된 메모리 양, CPU 성능, I/O 대역폭 등이 있는 DB 인스턴스 클래스를 선택합니다. 워크로드가 변경되면 라이터 및 리더의 인스턴스 클래스를 수동으로 수정합니다. 프로비저닝된 모델은 예상 소비 패턴에 앞서 용량을 조정할 수 있는 경우에 적합하며 클러스터에서 라이터 및 리더의 인스턴스 클래스를 변경하는 동안 발생하는 잠시 중단을 허용할 때 잘 작동합니다.

Aurora Serverless v2는 처음부터 즉시 확장 가능한 서버리스 DB 클러스터를 지원하도록 설계되었습니다.Aurora Serverless v2는 프로비저닝된 라이터 및 리더와 동일한 수준의 보안 및 격리를 제공하도록 설계되었습니다. 이러한 측면은 멀티테넌트 서버리스 클라우드 환경에서 매우 중요합니다. 동적 크기 조정 메커니즘은 오버헤드가 매우 적기 때문에 데이터베이스 워크로드의 변화에 신속하게 대응할 수 있습니다. 또한 처리 수요의 급격한 증가에 대응할 수 있을 만큼 강력합니다.

Aurora Serverless v2를 사용하면 각 라이터 및 리더에 대한 특정 데이터베이스 용량에 얽매이지 않고 Aurora DB 클러스터를 생성할 수 있습니다. 최소 및 최대 용량 범위만 지정합니다. Aurora는 해당 용량 범위 내에서 클러스터에 있는 각 Aurora Serverless v2 라이터 또는 리더를 확장합니다. 각 라이터 또는 리더가 동적으로 확장할 수 있는 다중 AZ 클러스터를 사용하면 동적 확장 및 고가용성을 활용할 수 있습니다.

Aurora Serverless v2는 최소 및 최대 용량 사양에 따라 데이터베이스 리소스의 크기를 자동으로 조정합니다. 대부분의 크기 조정 이벤트 작업은 라이터 또는 리더를 동일한 호스트에 유지하기 때문에 확장이 빠릅니다. 드문 경우이지만 Aurora Serverless v2 라이터 또는 리더가 한 호스트에서 다른 호스트로 이동하는 경우 Aurora Serverless v2가 연결을 자동으로 관리합니다. 데이터베이스 클라이언트 애플리케이션 코드나 데이터베이스 연결 문자열을 변경할 필요가 없습니다.

Aurora Serverless v2의 경우 프로비저닝된 클러스터와 마찬가지로 스토리지 용량과 컴퓨팅 용량이 분리됩니다. Aurora Serverless v2 용량 및 확장을 언급할 때 항상 증가하거나 감소하는 컴퓨팅 용량입니다. 따라서 CPU 및 메모리 용량이 낮은 수준으로 축소되더라도 클러스터에 수 테라바이트의 데이터가 포함될 수 있습니다.

데이터베이스 서버를 프로비저닝 및 관리하는 대신 데이터베이스 용량을 지정합니다. Aurora Serverless v2 용량에 대한 자세한 내용은 Aurora Serverless v2 용량의 내용을 참조하세요. 각 Aurora Serverless v2 라이터 또는 리더의 실제 용량은 작업 부하를 바탕으로 시간이 지남에 따라 달라집니다. 이러한 메커니즘에 대한 자세한 내용은 Aurora Serverless v2 크기 조정의 내용을 참조하세요.

> **Important**
>
> Aurora Serverless v1에서는 클러스터에 최소 및 최대 용량 값 사이에서 확장할 수 있는 단일 컴퓨팅 용량 측정값이 있습니다. Aurora Serverless v2에서는 클러스터에 라이터 외에 리더도 포함될 수 있습니다. 각 Aurora Serverless v2 라이터와 리더는 최소 및 최대 용량 값 사이에서 크기를 조정할 수 있습니다. 따라서, Aurora Serverless v2 클러스터의 총 용량은 DB 클러스터에 대해 정의한 용량 범위와 클러스터의 라이터 및 리더 수에 따라 달라집니다. 특정 시간에는 Aurora DB 클러스터에서 활발하게 사용되는 Aurora Serverless v2 용량에 대해서만 요금이 부과됩니다.

Aurora DB 클러스터 구성

각 Aurora DB 클러스터에 대해 Aurora Serverless v2 용량 또는 프로비저닝된 용량을 선택하거나 두 가지 모두를 조합해서 선택할 수 있습니다.

혼합 구성 클러스터라고 하는 Aurora Serverless v2 및 프로비저닝된 용량을 모두 포함하는 클러스터를 설정할 수 있습니다. 예를 들어, Aurora Serverless v2 라이터에서 사용할 수 있는 것보다 많은 읽기/쓰기 용량이 필요하다고 가정하겠습니다. 이 경우 대규모로 프로비저닝된 라이터를 사용하여 클러스터를 설정할 수 있습니다. 이 경우에도 리더에 대해 Aurora Serverless v2를 사용할 수 있습니다. 또는 클러스터의 쓰기 워크로드는 다양하지만 읽기 워크로드는 일정하다고 가정합니다. 이 경우 Aurora Serverless v2 라이터와 하나 이상의 프로비저닝된 리더로 클러스터를 설정할 수 있습니다.

모든 용량을 Aurora Serverless v2에서 관리하는 DB 클러스터를 설정할 수도 있습니다. 이렇게 하려면 새 클러스터를 만들고 처음부터 Aurora Serverless v2를 사용할 수 있습니다. 또는 기존 클러스터에서 프로비저닝된 모든 용량을 Aurora Serverless v2로 교체할 수 있습니다. 예를 들어, 이전 엔진 버전의 일부 업그레이드 경로는 프로비저닝된 라이터로 시작하여 Aurora Serverless v2 라이터로 교체해야 합니다. Aurora Serverless v2로 새 DB 클러스터를 생성하거나 기존 DB 클러스터를 Aurora Serverless v2로 전환하는 프로시저는 [Aurora Serverless v2 DB 클러스터 생성] 및 [프로비저닝된 클러스터에서 Aurora Serverless v2로 전환]의 내용을 참조하세요.

DB 클러스터에서 Aurora Serverless v2를 전혀 사용하지 않으면 DB 클러스터의 모든 라이터와 리더가 프로비저닝됩니다. 이것은 대부분의 사용자에게 친숙하고 가장 오래되고 가장 일반적인 DB 클러스터입니다. 사실 Aurora Serverless 이전에는 이런 종류의 Aurora DB 클러스터에는 특별한 이름이 없었습니다. 프로비저닝된 용량은 일정합니다. 요금은 비교적 쉽게 예측할 수 있습니다. 그러나 얼마나 많은 용량이 필요한지는 미리 예측해야 합니다. 경우에 따라 예측이 부정확하거나 용량 요구 사항이 변경될 수 있습니다. 이러한 경우 DB 클러스터가 언더프로비저닝(기대치보다 더 느림) 또는 오버프로비저닝(기대치보다 더 비쌈)될 수 있습니다.

Aurora Serverless v2 용량

Aurora Serverless v2의 측정 단위는 Aurora 용량 단위(ACU)입니다. Aurora Serverless v2 용량은 프로비저닝된 클러스터에 사용하는 DB 인스턴스 클래스에 연결되지 않습니다.

각 ACU는 약 2기가바이트(GB)의 메모리로 해당 CPU 및 네트워킹의 합한 용량입니다. 이 측정 단위를 사용하여 데이터베이스 용량 범위를 지정합니다. `ServerlessDatabaseCapacity` 및 `ACUUtilization` 지표는 데이터베이스가 실제로 사용하고 있는 용량과 해당 용량이 지정된 범위에 속하는 위치를 결정하는 데 도움이 됩니다.

언제든지 각 Aurora Serverless v2 DB 라이터 또는 리더에는 용량이 있습니다. 용량은 ACU를 나타내는 부동 소수점 숫자로 표시됩니다. 라이터 또는 리더가 확장될 때마다 용량이 증가하거나 감소합니다. 이 값은 매초마다 측정됩니다. Aurora Serverless v2를 사용하려는 각 DB 클러스터에 대해 용량 범위, 즉 각 Aurora Serverless v2 라이터 또는 리더가 확장할 수 있는 최소 및 최대 용량 값을 정의합니다. 용량 범위는 DB 클러스터의 각 Aurora Serverless v2 라이터 또는 리더에 대해 동일합니다. 각 Aurora Serverless v2 라이터 또는 리더는 해당 범위의 어딘가에 해당하는 고유한 용량을 가지고 있습니다.

다음 표에는 Aurora MySQL 및 Aurora PostgreSQL에 지원되는 Aurora Serverless v2 용량 범위가 나와 있습니다.

용량 범위(ACU)	Aurora MySQL 지원 버전	Aurora PostgreSQL 지원되는 버전
0.5~128	3.02.0 이상	13.6 이상, 14.3 이상, 15.2 이상, 16.1 이상
0.5~256	3.06.0 이상	13.13 이상, 14.10 이상, 15.5 이상, 16.1 이상
0~256	3.08.0 이상	13.15 이상, 14.12 이상, 15.7 이상, 16.3 이상

자동 일시 중지 기능을 지원하는 Aurora Serverless v2 버전의 경우 정의할 수 있는 최소 Aurora Serverless v2 용량은 0ACU입니다. 최대 용량 값보다 작거나 같은 경우 더 높은 숫자를 지정할 수 있습니다. 최소 용량을 작은 수로 설정하면 로드가 적은 DB 클러스터가 최소한의 컴퓨팅 리소스를 소비할 수 있습니다. 동시에 연결을 즉시 수락하고 사용량이 많을 때 확장할 준비가 되어 있습니다.

각 DB 라이터 또는 리더가 버퍼 풀에서 애플리케이션의 작업 집합을 보유할 수 있도록 최소값을 설정하는 것이 좋습니다. 이렇게 하면 유휴 기간 동안 버퍼 풀의 내용이 삭제되지 않습니다. 최소 용량 값을

선택할 때 모든 고려 사항은 클러스터에 대한 최소 Aurora Serverless v2 용량 설정 선택 페이지를 참조하세요. 최대 용량 값을 선택할 때 모든 고려 사항은 클러스터의 최대 Aurora Serverless v2 용량 설정 선택 페이지를 참조하세요.

다중 AZ 배포에서 리더를 구성하는 방법에 따라 리더의 용량은 라이터의 용량에 연결되거나 별도로 구성될 수 있습니다. 이를 수행하는 방법에 대한 자세한 내용은 Aurora Serverless v2 크기 조정 페이지를 참조하세요.

Aurora Serverless v2 모니터링에는 시간 경과에 따른 DB 클러스터의 라이터 및 리더 용량 값 측정이 포함됩니다. 데이터베이스가 최소 용량으로 축소되지 않는 경우 최소값을 조정하고 데이터베이스 애플리케이션을 최적화하는 등의 작업을 수행할 수 있습니다. 데이터베이스가 지속적으로 최대 용량에 도달하면 최대값을 늘리는 등의 작업을 수행할 수 있습니다. 또한 데이터베이스 애플리케이션을 최적화하고 더 많은 리더에 쿼리 로드를 분산시킬 수 있습니다.

Aurora Serverless v2 용량에 대한 요금은 ACU 시간으로 측정됩니다. Aurora Serverless v2 요금 계산 방법에 대한 정보는 Aurora 요금 페이지를 참조하세요.

클러스터의 총 라이터 및 리더 수가 `n`개라고 가정해 보겠습니다. 이 경우 클러스터는 데이터베이스 작업을 실행하지 않을 때 대략적으로 *n* x `minimum ACUs`를 사용합니다. Aurora 자체는 약간의 로드를 유발하는 모니터링 또는 유지 관리 작업을 실행할 수 있습니다. 해당 클러스터는 데이터베이스가 전체 용량으로 실행 중일 때 *n* x `maximum ACUs`만 소비합니다.

적절한 최소 및 최대 ACU 값 선택에 대한 자세한 내용은 Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택 페이지를 참조하세요. 지정하는 최소 및 최대 ACU 값은 일부 Aurora 구성 파라미터가 Aurora Serverless v2에 대해 작동하는 방식에도 영향을 미칩니다. 용량 범위와 구성 파라미터터 간의 상호 작용에 대한 자세한 내용은 Aurora Serverless v2에 대한 파라미터 그룹 작업 페이지를 참조하세요.

Aurora Serverless v2 크기 조정

각 Aurora Serverless v2 라이터 또는 리더에 대해 Aurora는 CPU, 메모리 및 네트워크와 같은 리소스 사용률을 지속적으로 추적합니다. 이러한 측정값을 통칭하여 로드라고 합니다. 로드에는 애플리케이션에서 수행한 데이터베이스 작업이 포함됩니다. 또한 데이터베이스 서버 및 Aurora 관리 작업에 대한 백그라운드 처리가 포함됩니다. 용량이 이들 중 하나로 제한되면 Aurora Serverless v2가 확장합니다. Aurora Serverless v2는 또한 확장함으로써 해결할 수 있는 성능 문제를 감지할 때 확장합니다. Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표 및 성능 개선 도우미로 Aurora Serverless v2 성능 모니터링의 프로시저를 사용하여 리소스 사용률과 이것이 Aurora Serverless v2의 크기기 조정에 미치는 영향을 모니터링할 수 있습니다.

로드는 DB 클러스터의 라이터와 리더에 따라 다를 수 있습니다. 라이터는 `CREATE TABLE`, `ALTER TABLE` 및 `DROP TABLE`과 같은 모든 데이터 정의 언어(DDL) 문을 처리합니다. 라이터는 또한 `INSERT` 및 `UPDATE`와 모든 데이터 조작 언어(DML) 문을 처리합니다. 리더는 `SELECT` 쿼리와 같은 읽기 전용 문을 처리할 수 있습니다.

크기 조정은 데이터베이스의 Aurora Serverless v2 용량을 늘리거나 줄이는 작업입니다. Aurora Serverless v2의 경우 각 라이터와 리더에는 ACU로 측정되는 고유한 현재 용량 값이 있습니다. Aurora Serverless v2는 현재 용량이 로드를 처리하기에 너무 낮을 때 라이터 또는 리더를 더 높은 용량으로 확장합니다. 현재 용량이 필요 이상으로 높으면 라이터 또는 리더를 더 낮은 용량으로 축소합니다.

DB 클러스터가 임계값에 도달할 때마다 용량을 두 배로 늘려 확장하는 Aurora Serverless v1과 달리 Aurora Serverless v2는 용량을 점진적으로 늘릴 수 있습니다. 워크로드 요구량이 라이터 또는 리더의 현재 데이터베이스 용량에 도달하기 시작하면 Aurora Serverless v2는 해당 라이터 또는 리더에 대한 ACU 수를 늘립니다. Aurora Serverless v2는 소비된 리소스에 대해 최상의 성능을 제공하는 데 필요한 증분으로 용량을 확장합니다. 크기 조정은 0.5 ACU의 작은 단위로 발생합니다. 현재 용량이 클수록 크기 확장분이 커지므로 크기 조정이 빨라질 수 있습니다.

Aurora Serverless v2 확장은 매우 빈번하고 세부적이며 중단되지 않기 때문에 AWS Management Console에서와 같이 Aurora Serverless v1에서 개별 이벤트를 일으키지 않습니다. 대신 `ServerlessDatabaseCapacity` 및 `ACUUtilization`과 같은 Amazon CloudWatch 지표를 측정하고 시간 경과에 따른 최소값, 최대값 및 평균값을 추적할 수 있습니다. Aurora 지표에 대해 자세히 알아보려면 [Amazon Aurora 클러스터에서 지표 모니터링](#) 페이지를 참조하세요. Aurora Serverless v2 모니터링에 대한 팁은 [Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표](#) 페이지를 참조하세요.

연결된 라이터와 동시에 또는 라이터와 독립적으로 리더를 확장하도록 선택할 수 있습니다. 해당 리더에 대한 프로모션 티어를 지정하면 됩니다.

- 프로모션 티어 0 및 1의 리더는 라이터와 동시에 규모가 조정됩니다. 이러한 확장 동작은 우선 순위 티어 0 및 1의 리더가 가용성에 이상적입니다. 장애 조치(failover)의 경우 라이터의 워크로드를 인계할 수 있도록 적절한 용량으로 항상 크기가 조정되기 때문입니다.
- 프로모션 티어 2~15의 리더는 라이터와 독립적으로 확장됩니다. 각 판독기는 클러스터에 대해 지정한 최소 및 최대 ACU 값 내에 유지됩니다. 리더가 연결된 라이터 DB와 독립적으로 확장되면 라이터가 계속해서 많은 양의 트랜잭션을 처리하는 동안 유휴 상태가 되어 축소될 수 있습니다. 더 낮은 프로모션 티어에서 사용할 수 있는 다른 판독기가 없는 경우 장애 조치 대상으로 계속 사용할 수 있습니다. 그러나 라이터로 프로모션되는 경우 라이터의 전체 워크로드를 처리하기 위해 확장해야 할 수 있습니다.

프로모션 티어에 대한 자세한 내용은 Aurora Serverless v2 리더에 대한 승격 티어 선택 페이지를 참조하세요.

Aurora Serverless v1의 크기 조정 포인트 및 관련 타임아웃 기간의 개념은 Aurora Serverless v2에 적용되지 않습니다. Aurora Serverless v2 크기 조정은 데이터베이스 연결이 열려 있는 동안, SQL 트랜잭션이 진행 중인 동안, 테이블이 잠겨 있는 동안, 임시 테이블을 사용하는 동안 발생할 수 있습니다. Aurora Serverless v2는 조용한 지점이 크기 조정을 시작할 때까지 기다리지 않습니다. 확장은 진행 중인 데이터베이스 작업을 방해하지 않습니다.

워크로드에 단일 라이터 및 단일 리더에서 사용할 수 있는 것보다 더 많은 읽기 용량이 필요한 경우 클러스터에 여러 Aurora Serverless v2 리더를 추가할 수 있습니다. 각 B1 리더는 DB 클러스터에 대해 지정한 최소 및 최대 용량 값 범위 내에서 확장할 수 있습니다. 클러스터의 리더 엔드포인트를 사용하여 읽기 전용 세션을 리더로 보내고 라이터의 로드를 줄일 수 있습니다.

Aurora Serverless v2가 확장을 수행하는지 여부와 확장이 시작된 후 발생하는 속도는 클러스터의 최소 및 최대 ACU 설정에 따라 달라집니다. 또한 리더가 라이터와 함께 확장되도록 구성되었는지 또는 독자적으로 구성되었는지에 따라 다릅니다. Aurora Serverless v2 크기 조정에 영향을 미치는 요인에 대한 자세한 내용은 Aurora Serverless v2의 성능 및 크기 조정 페이지를 참조하세요.

0으로 크기 조정

최근 Aurora MySQL 및 Aurora PostgreSQL 버전에서 Aurora Serverless v2 라이터와 리더는 0ACU까지 스케일 다운할 수 있습니다. 이 기능을 자동 일시 중지 및 재개 또는 자동 일시 중지라고 합니다. 최소 용량에 0 또는 0이 아닌 값을 지정하여 이 동작을 허용할지 선택할 수 있습니다. Aurora Serverless v2 인스턴스가 일시 중지되기까지 대기할 시간을 선택할 수도 있습니다. 이 기능이 있는 버전에 대한 자세한 내용은 Aurora Serverless v2 용량 섹션을 참조하세요. 이 기능을 효과적으로 사용하는 방법에 대한 자세한 내용은 Aurora Serverless v2에 대해 자동 일시 중지 및 재개를 사용하여 0ACU로 크기 조정 섹션을 참조하세요.

이전 Aurora MySQL 및 Aurora PostgreSQL 버전에서는 유휴 Aurora Serverless v2 라이터와 리더가 클러스터에 지정된 최소 ACU 값으로 스케일 다운할 수 있지만 0ACU까지는 스케일 다운할 수 없습니다. 이 경우 용량 범위를 설정할 때 0ACU를 선택할 수 없습니다. 해당 동작은 유휴 기간 후에 일시 중지될 수 있지만 새 연결을 열 때 다시 시작하는 데 시간이 걸리는 Aurora Serverless v1과 다릅니다.

Aurora Serverless v2 용량을 사용하는 DB 클러스터가 한동안 필요하지 않은 경우 프로비저닝된 DB 클러스터와 마찬가지로 클러스터를 중지했다가 시작할 수 있습니다. 이 기법은 한 번에 여러 시간 동안 필요하지 않을 수 있고 클러스터를 재개하는 속도가 중요하지 않은 개발 및 테스트 시스템에 가장 적합합니다. 클러스터 중지/시작 기능은 모든 Aurora Serverless v2 버전에서 사용할 수 있습니다. 해당 기능에 대한 자세한 내용은 Amazon Aurora DB 클러스터 중지 및 시작 섹션을 참조하세요.

Aurora Serverless v2 및 고가용성

Aurora DB 클러스터에 대한 고가용성을 설정하는 방법은 이를 다중 AZ DB 클러스터로 만드는 것입니다. 다중 AZ Aurora DB 클러스터에는 둘 이상의 가용 영역(AZ)에서 항상 사용 가능한 컴퓨팅 용량이 있습니다. 이 구성은 심각한 중단이 발생한 경우에도 데이터베이스를 계속 가동하고 실행합니다. Aurora는 라이터 또는 전체 AZ에 영향을 미치는 문제의 경우 자동 장애 조치를 수행합니다. Aurora Serverless v2를 사용하면 라이터의 용량에 따라 확장 및 축소할 대기 컴퓨팅 용량을 선택할 수 있습니다. 이렇게 하면 두 번째 AZ의 컴퓨팅 용량이 언제든지 현재 워크로드를 인수할 준비가 됩니다. 동시에 모든 AZ의 컴퓨팅 용량은 데이터베이스가 유휴 상태일 때 축소될 수 있습니다. Aurora가 AWS 리전 및 가용 영역에서 작동하는 방식에 대한 자세한 내용은 Aurora DB 인스턴스의 고가용성 페이지를 참조하세요.

Aurora Serverless v2 다중 AZ 기능은 라이터와 함께 리더를 사용합니다. 리더 지원은 Aurora Serverless v1에는 없는 Aurora Serverless v2의 새로운 기능입니다. 3개의 AZ에 분산된 최대 15개의 Aurora Serverless v2 리더를 Aurora DB 클러스터에 추가할 수 있습니다.

전체 클러스터 또는 전체 AWS 리전에 영향을 미치는 문제의 경우에도 계속 사용 가능해야 하는 비즈니스 크리티컬 애플리케이션의 경우 Aurora 글로벌 데이터베이스를 설정할 수 있습니다. 보조 클러스터에서 Aurora Serverless v2 용량을 사용하여 재해 복구 중에 인계받을 준비가 되도록 할 수 있습니다. 데이터베이스가 사용 중이 아닐 때 축소할 수도 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora Global Database 사용 페이지를 참조하세요.

Aurora Serverless v2는 장애 조치 및 기타 고가용성 기능을 위해 프로비저닝된 것처럼 작동합니다. 자세한 정보는 Amazon Aurora의 고가용성의 내용을 참조하세요.

Aurora Serverless v2 클러스터의 최대 가용성을 보장한다고 가정합니다. 라이터 외에 리더를 만들 수 있습니다. 리더를 프로모션 티어 0 또는 1에 할당하면 라이터에 대해 발생하는 모든 크기 조정은 리더에게도 발생합니다. 이렇게 하면 장애 조치(failover)가 발생할 때 동일한 용량의 리더가 항상 라이터를 대신할 수 있는 상태가 됩니다.

클러스터가 트랜잭션을 계속 처리하는 동시에 비즈니스에 대한 분기별 보고서를 실행하려고 한다고 가정합니다. 클러스터에 Aurora Serverless v2 리더를 추가하고 2에서 15까지의 프로모션 티어에 할당하면 해당 리더에 직접 연결하여 보고서를 실행할 수 있습니다. 보고 쿼리의 메모리 집약적 및 CPU 집약적 정도에 따라 해당 리더는 워크로드를 수용하도록 확장할 수 있습니다. 그런 다음 보고서가 완료되면 다시 축소할 수 있습니다.

Aurora Serverless v2 및 스토리지

각 Aurora DB 클러스터의 스토리지는 모든 데이터의 사본 6개로 구성되며 3개의 AZ에 분산되어 있습니다. 이 기본 제공 데이터 복제는 DB 클러스터에 라이터 외에 리더가 포함되어 있는지와 관계없이 적

용됩니다. 그렇게 하면 클러스터의 컴퓨팅 용량에 영향을 미치는 문제로부터 데이터를 안전하게 보호할 수 있습니다.

Aurora Serverless v2 스토리지는 Amazon Aurora 스토리지에서 설명한 것과 동일한 신뢰성 및 내구성 특성을 가지고 있습니다. 컴퓨팅 용량이 Aurora Serverless v2를 사용하든 프로비저닝하든 상관없이 Aurora DB 클러스터용 스토리지가 동일하게 작동하기 때문입니다.

Aurora 클러스터에 대한 구성 파라미터

프로비저닝된 DB 클러스터와 마찬가지로 Aurora Serverless v2 용량이 있는 클러스터에 대해 동일한 클러스터 및 데이터베이스 구성 파라미터를 모두 조정할 수 있습니다. 그러나 일부 용량 관련 파라미터는 Aurora Serverless v2에 대해 다르게 처리됩니다. 혼합 구성 클러스터에서 해당 용량 관련 파라미터에 대해 지정하는 파라미터 값은 프로비저닝된 라이터 및 리더에 계속 적용됩니다.

거의 모든 파라미터는 프로비저닝된 파라미터와 마찬가지로 Aurora Serverless v2 라이터 및 리더에 대해 동일한 방식으로 작동합니다 크기 조정 중에 Aurora가 자동으로 조정하는 일부 파라미터와 Aurora가 최대 용량 설정에 따라 고정된 값으로 유지하는 일부 파라미터는 예외입니다.

예를 들어 버퍼 캐시용으로 예약된 메모리 양은 라이터나 리더가 확장되면 증가하고 축소되면 감소합니다. 이렇게 하면 데이터베이스가 사용 중이지 않을 때 메모리를 해제할 수 있습니다. 반대로 Aurora는 최대 연결 수를 최대 용량 설정에 따라 적절한 값으로 자동 설정합니다. 이렇게 하면 로드가 떨어지고 Aurora Serverless v2의 크기가 축소되더라도 활성 연결이 끊어지지 않습니다. Aurora Serverless v2가 특정 파라미터를 처리하는 방법에 대한 정보는 Aurora Serverless v2에 대한 파라미터 그룹 작업 페이지를 참조하세요.

Aurora Serverless v2 요구 사항 및 제한 사항

Aurora Serverless v2 DB 인스턴스를 사용하려는 클러스터를 생성할 때 다음 요구 사항 및 제한 사항에 주의하세요.

주제

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. Aurora와 Aurora Serverless v2 버전 및 리전 가용성에 대한 자세한 정보는 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

다음 예에서는 특정 AWS 리전에 대해 Aurora Serverless v2와 함께 사용할 수 있는 정확한 DB 엔진 값을 확인하는 AWS CLI 명령을 보여줍니다. Aurora Serverless v2에 대한 `--db-instance-class` 파라미터는 항상 `db.serverless`입니다. `--engine` 파라미터는 `aurora-mysql` 또는 `aurora-postgresql`일 수 있습니다. 적절한 `--region` 및 `--engine` 값을 대체하여 사용할 수 있는 `--engine-version` 값을 확인하세요. 명령이 출력을 생성하지 않으면 Aurora Serverless v2 및 DB 엔진 조합에 대해 AWS 리전를 사용할 수 없습니다.

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output text
```

Aurora Serverless v2를 사용하는 클러스터에는 용량 범위가 지정되어 있어야 합니다.

`db.serverless` DB 인스턴스 클래스를 사용하는 DB 인스턴스를 추가하려면 먼저 Aurora 클러스터에 `ServerlessV2ScalingConfiguration` 속성이 있어야 합니다. 이 속성은 용량 범위를 지정합니다. Aurora Serverless v2 용량 범위는 최소 0ACU(Aurora 용량 단위)에서 최대 256ACU까지 0.5ACU 단위로 증가합니다. 허용되는 최소값은 Aurora 버전에 따라 다릅니다. 각 ACU는 약 2GiB RAM 및 관련 CPU 및 네트워킹에 해당하는 용량을 제공합니다. Aurora Serverless v2 용량 범위 설정을 사용하는 방법에 대한 자세한 내용은 Aurora Serverless v2 작동 방식 페이지를 참조하세요.

다양한 DB 엔진 버전에 허용되는 용량 범위는 Aurora Serverless v2 용량 섹션을 참조하세요.

클러스터 및 연결된 Aurora Serverless v2 DB 인스턴스를 생성할 때 AWS Management Console에서 최소 및 최대 ACU 값을 지정할 수 있습니다. AWS CLI에서 `--serverless-v2-scaling-configuration` 옵션을 지정할 수도 있습니다. 또는 Amazon RDS API로 `ServerlessV2ScalingConfiguration` 파라미터를 지정할 수 있습니다. 클러스터를 생성하거나

기존 클러스터를 수정할 때 이 속성을 지정할 수 있습니다. 용량 범위를 설정하는 프로시저는 클러스터의 Aurora Serverless v2 용량 설정 페이지를 참조하세요. 최소 및 최대 용량 값을 선택하는 방법과 이러한 설정이 일부 데이터베이스 파라미터에 미치는 영향에 대한 자세한 내용은 Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택 페이지를 참조하세요.

일부 프로비저닝된 기능은 Aurora Serverless v2에서 지원되지 않습니다.

Aurora 프로비저닝된 DB 인스턴스의 다음 기능은 현재 Amazon Aurora Serverless v2에서 사용할 수 없습니다

- 데이터베이스 활동 스트림(DAS)
- Aurora PostgreSQL용 클러스터 캐시 관리하다 `apg_ccm_enabled` 구성 파라미터는 Aurora Serverless v2 DB 인스턴스에 적용되지 않습니다.

일부 Aurora 기능은 Aurora Serverless v2에서 작동하지만 특정 워크로드가 있는 해당 기능의 메모리 요구 사항에 필요한 것보다 용량 범위가 낮은 경우 문제가 발생할 수 있습니다. 이 경우 데이터베이스가 평소와 같이 제대로 작동되지 않거나 메모리 부족 오류가 발생할 수 있습니다. 적절한 용량 범위 설정에 대한 권장 사항은 Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택 페이지를 참조하세요. 잘못 구성된 용량 범위로 인해 데이터베이스에 메모리 부족 오류가 발생하는 경우 문제 해결 정보는 메모리 부족 오류 방지의 내용을 참조하세요.

Aurora Auto Scaling은 지원되지 않습니다. 이러한 유형의 크기 조정은 CPU 사용량에 따라 새 리더를 추가하여 읽기 집약적인 추가 워크로드를 처리합니다. 그러나 CPU 사용량에 기반한 크기 조정은 Aurora Serverless v2에 의미가 있지 않습니다. 아니면 Aurora Serverless v2 리더 DB 인스턴스를 미리 생성하고 낮은 용량으로 축소된 상태로 둘 수 있습니다. 이는 새로운 DB 인스턴스를 동적으로 추가하는 것보다 클러스터의 읽기 용량을 확장하는 더 빠르고 덜 방해가 되는 방법입니다.

일부 Aurora Serverless v2 측면은 Aurora Serverless v1과 다름

Aurora Serverless v1 사용자이고 Aurora Serverless v2를 처음 사용하는 경우 Aurora Serverless v2와 Aurora Serverless v1 요구 사항의 차이점을 참조하여 Aurora Serverless v1과 Aurora Serverless v2 간의 요구 사항이 어떻게 다른지 이해하세요.

Aurora Serverless v2를 사용하는 DB 클러스터 생성

Aurora Serverless v2 DB 인스턴스를 추가할 수 있는 Aurora 클러스터를 생성하려면 Amazon Aurora DB 클러스터 생성의 절차를 따르세요. Aurora Serverless v2를 사용하면 클러스터를 프로비저닝된 클

러스터와 교환할 수 있습니다. 일부 DB 인스턴스는 Aurora Serverless v2를 사용하고 일부 DB 인스턴스는 프로비저닝되도록 클러스터를 구성할 수 있습니다.

주제

- Aurora Serverless v2 DB 클러스터 설정
- Aurora Serverless v2 DB 클러스터 생성
- Aurora Serverless v2 라이터 DB 인스턴스 생성

Aurora Serverless v2 DB 클러스터 설정

클러스터의 초기 설정이 Aurora Serverless v2 요구 사항 및 제한 사항에 나열된 요구 사항을 충족하는지 확인합니다. 클러스터에 Aurora Serverless v2 DB 클러스터를 추가할 수 있도록 다음 설정을 지정하세요.

AWS 리전

Aurora Serverless v2 DB 인스턴스를 사용할 수 있는 AWS 리전 내에 클러스터를 생성합니다. 사용 가능한 리전에 대한 자세한 내용은 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

DB 엔진 버전

Aurora Serverless v2와 호환되는 엔진 버전을 선택합니다. Aurora Serverless v2 버전 요구 사항에 대한 자세한 내용은 Aurora Serverless v2 요구 사항 및 제한 사항 섹션을 참조하세요.

DB 인스턴스 클래스

AWS Management Console을 사용하여 클러스터를 생성하는 경우 라이터 DB 인스턴스에 DB 인스턴스 클래스를 동시에 선택합니다. 서버리스(Serverless) DB 인스턴스 클래스를 선택합니다. 해당 DB 인스턴스 클래스를 선택할 때 라이터 DB 인스턴스의 용량 범위도 지정합니다. 동일한 용량 범위가 해당 클러스터에 추가하는 다른 모든 Aurora Serverless v2 DB 인스턴스에 적용됩니다.

DB 인스턴스 클래스에 대한 선택 항목에 서버리스가 없는 경우 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진에 대해 지원되는 DB 엔진 버전을 선택했는지 확인합니다.

AWS CLI 또는 Amazon RDS API를 사용할 때 DB 인스턴스 클래스에 지정하는 파라미터는 `db.serverless`입니다.

용량 범위

클러스터의 모든 DB 인스턴스에 적용되는 최소 및 최대 Aurora 용량 단위(ACU) 값을 입력합니다. 이 옵션은 DB 인스턴스 클래스로 서버리스(Serverless)를 선택하는 경우 클러스터 생성(Create cluster)과 리더 추가(Add reader) 콘솔 페이지에서 모두 사용할 수 있습니다.

다양한 DB 엔진 버전에 허용되는 용량 범위는 Aurora Serverless v2 용량 섹션을 참조하세요.

최소 및 최대 ACU 필드가 보이지 않으면 라이터 DB 인스턴스에 대해 서버리스 DB 인스턴스 클래스를 선택했는지 확인합니다.

처음에 프로비저닝된 DB 인스턴스로 클러스터를 생성하는 경우 최소 및 최대 ACU를 지정하지 않습니다. 이 경우 나중에 클러스터를 수정하여 해당 설정을 추가할 수 있습니다. 클러스터에 Aurora Serverless v2 리더 DB 인스턴스를 추가할 수도 있습니다. 해당 프로세스의 일부로 용량 범위를 지정합니다.

클러스터의 용량 범위를 지정하기 전까지는 AWS CLI 또는 RDS API를 사용하여 클러스터에 Aurora Serverless v2 DB 인스턴스를 추가할 수 없습니다. Aurora Serverless v2 DB 인스턴스를 추가하려고 하면 오류가 발생합니다. AWS CLI 또는 RDS API 절차에서 용량 범위는 `ServerlessV2ScalingConfiguration` 속성으로 표시됩니다.

두 개 이상의 리더 DB 인스턴스를 포함하는 클러스터의 경우 각 Aurora Serverless v2 리더 DB 인스턴스의 장애 조치 우선순위가 해당 DB 인스턴스의 확장 및 축소 방식에 중요한 역할을 합니다. 클러스터를 처음 생성할 때 우선순위를 지정할 수 없습니다. 클러스터에 두 번째 이후의 리더 DB 인스턴스를 추가할 때 이 속성을 염두에 두세요. 자세한 내용은 Aurora Serverless v2 리더에 대한 승격 티어 선택 단원을 참조하십시오.

Aurora Serverless v2 DB 클러스터 생성

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 Aurora Serverless v2 DB 클러스터를 생성할 수 있습니다.

콘솔

Aurora Serverless v2 라이터를 사용하여 클러스터를 생성하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

3. 데이터베이스 생성을 선택합니다. 나타나는 페이지에서 다음 옵션을 선택합니다.

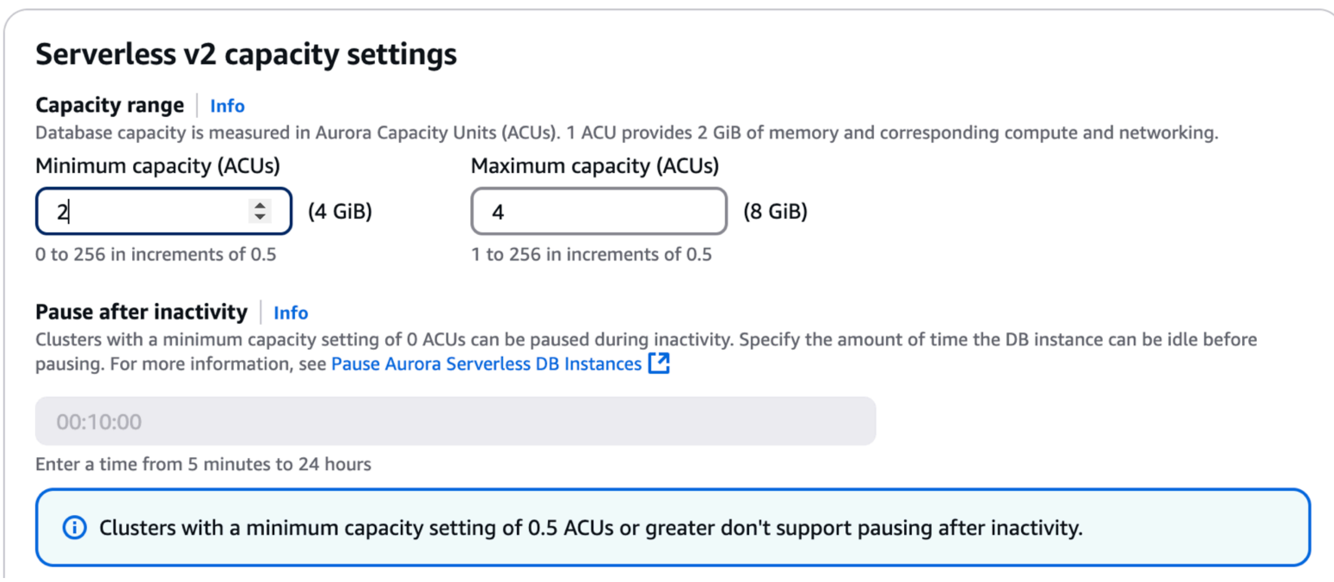
   - 엔진 유형에서 Aurora(MySQL 호환) 또는 Aurora(PostgreSQL 호환)를 선택합니다.
   - 버전의 경우 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진에 대해 지원되는 버전 중 하나를 선택합니다.

4. DB 인스턴스 클래스에서 Serverless v2를 선택합니다.

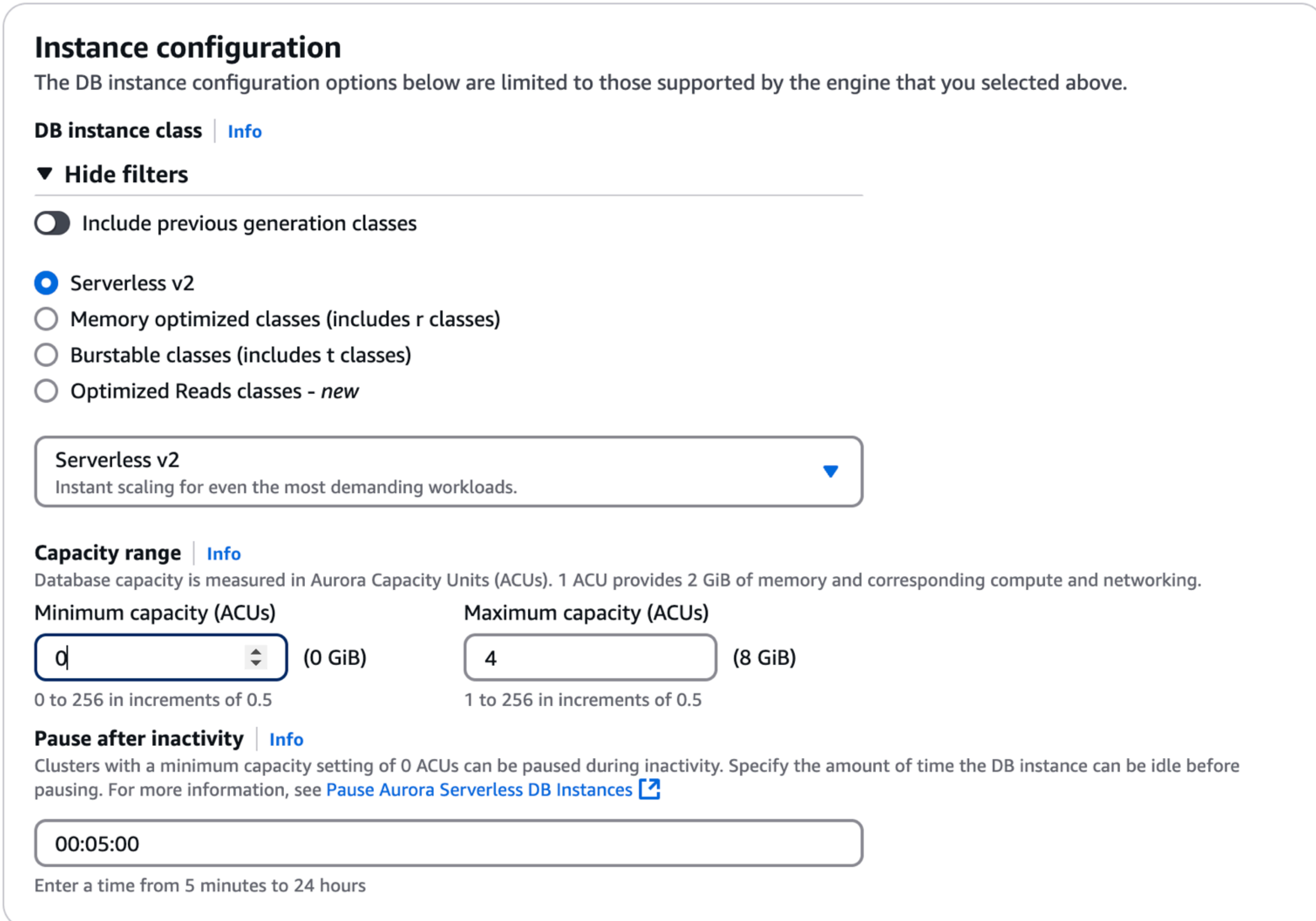
5. 용량 범위의 경우 기본 범위를 사용할 수 있습니다. 또는 최소 및 최대 용량 단위에 대해 다른 값을 선택할 수 있습니다. 최소 0ACU에서 최대 256ACU까지 0.5ACU 단위로 선택할 수 있습니다.

   Aurora Serverless v2 용량 단위에 대한 자세한 내용은 Aurora Serverless v2 용량 및 Aurora Serverless v2의 성능 및 크기 조정 섹션을 참조하세요.

   선택한 엔진 및 버전에 따라 상한이 128ACU이거나 하한이 0.5ACU이거나 둘 다일 수 있습니다. Aurora 엔진과 버전의 각 조합에 대한 제한 사항은 Aurora Serverless v2 용량 섹션에서 자세히 알아보세요.



   최소 용량을 0ACU로 선택하면 Aurora Serverless v2 자동 일시 중지 및 재개 기능이 활성화됩니다. 이 경우 Aurora Serverless v2 DB 인스턴스가 데이터베이스 연결 없이 대기한 후 자동으로 일시 중지되기까지의 시간을 추가로 선택할 수 있습니다. 자동 일시 중지 및 재개 기능에 대한 자세한 내용은 Aurora Serverless v2에 대해 자동 일시 중지 및 재개를 사용하여 0ACU로 크기 조정 섹션을 참조하세요.



6. [Aurora DB 클러스터 설정]에 설명된 대로 다른 DB 클러스터 설정을 선택합니다.
7. 데이터베이스 생성을 선택하여 Aurora Serverless v2 DB 인스턴스를 라이터 인스턴스, 즉 기본 DB 인스턴스로 Aurora DB 클러스터를 생성합니다.

CLI

AWS CLI를 사용하여 Aurora Serverless v2 DB 인스턴스와 호환되는 DB 클러스터를 생성하려면 [Amazon Aurora DB 클러스터 생성]의 CLI 절차를 따릅니다. `create-db-cluster` 명령에는 다음과 같은 파라미터가 포함됩니다.

- --region *`AWS_Region_where_Aurora Serverless v2_instances_are_available`*
- --engine-version *`serverless_v2_compatible_engine_version`*
- --serverless-v2-scaling-configuration MinCapacity=*`minimum_capacity`*,MaxCapacity=*`maximum_capacity`*

다음 예에서는 Aurora Serverless v2 DB 클러스터의 생성을 보여줍니다.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> AWS CLI를 사용하여 Aurora Serverless v2 DB 클러스터를 생성하면 엔진 모드가 `serverless`가 아닌 `provisioned`로 출력에 나타납니다. `serverless` 엔진 모드는 Aurora Serverless v1을 나타냅니다.

이 예제에서는 관리자 암호를 생성하고 이를 Secrets Manager에서 관리하는 `--manage-master-user-password` 옵션을 지정합니다. 자세한 내용은 [Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리](#) 단원을 참조하십시오. 또는 `--master-password` 옵션을 사용하여 암호를 직접 지정하고 관리할 수 있습니다.

Aurora Serverless v2 버전 요구 사항에 대한 자세한 내용은 [Aurora Serverless v2 요구 사항 및 제한 사항](#) 섹션을 참조하세요. 용량 범위에 허용되는 숫자 및 해당 숫자에 대한 자세한 내용은 [Aurora Serverless v2 용량](#) 및 [Aurora Serverless v2의 성능 및 크기 조정](#) 섹션을 참조하세요.

기존 클러스터에 지정된 용량 설정이 있는지 확인하려면 `ServerlessV2ScalingConfiguration` 속성에 대한 `describe-db-clusters` 명령의 출력을 확인하세요. 이 속성은 다음과 비슷합니다.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> 클러스터를 만들 때 최소 및 최대 ACU를 지정하지 않으면 나중에 `modify-db-cluster` 명령을 사용하여 해당 설정을 추가할 수 있습니다. 이 설정을 추가하기 전에는 클러스터에 Aurora

Serverless v2 DB 인스턴스를 추가할 수 없습니다. `db.serverless` DB 인스턴스를 추가하려고 하면 오류가 발생합니다.

API

RDS API를 사용하여 Aurora Serverless v2 DB 인스턴스와 호환되는 DB 클러스터를 생성하려면 Amazon Aurora DB 클러스터 생성의 API 절차를 따릅니다. 다음 설정을 선택합니다. `CreateDBCluster` 작업에 다음 파라미터가 포함되어 있어야 합니다.

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

Aurora Serverless v2 버전 요구 사항에 대한 자세한 내용은 Aurora Serverless v2 요구 사항 및 제한 사항 섹션을 참조하세요. 용량 범위에 허용되는 숫자 및 해당 숫자에 대한 자세한 내용은 Aurora Serverless v2 용량 및 Aurora Serverless v2의 성능 및 크기 조정 섹션을 참조하세요.

기존 클러스터에 지정된 용량 설정이 있는지 확인하려면 `ServerlessV2ScalingConfiguration` 속성에 대한 `DescribeDBClusters` 작업의 출력을 확인하세요. 이 속성은 다음과 비슷합니다.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

Tip

클러스터를 만들 때 최소 및 최대 ACU를 지정하지 않으면 나중에 `ModifyDBCluster` 작업을 사용하여 해당 설정을 추가할 수 있습니다. 이 설정을 추가하기 전에는 클러스터에 Aurora Serverless v2 DB 인스턴스를 추가할 수 없습니다. `db.serverless` DB 인스턴스를 추가하려고 하면 오류가 발생합니다.

Aurora Serverless v2 라이터 DB 인스턴스 생성

Aurora 클러스터를 만들 때 Aurora Serverless v2 용량 범위를 지정하더라도 클러스터의 각 DB 인스턴스에 대해 Aurora Serverless v2를 사용할지 또는 프로비저닝된 것을 사용할지 선택할 수 있습니다. DB 클러스터에서 즉시 Aurora Serverless v2 사용을 시작하려면 `db.serverless` 인스턴스 클래스를

사용하는 라이터 DB 인스턴스를 추가합니다. 콘솔에서는 일반적으로 DB 클러스터를 만드는 워크플로의 일부로 이 옵션을 선택합니다. 따라서 이 프로시저는 CLI를 통해 설정을 수행할 때 주로 적용됩니다.

콘솔

AWS Management Console을 사용하여 DB 클러스터를 생성할 때 동시에 라이터 DB 인스턴스의 속성을 지정합니다. 라이터 DB 인스턴스가 Aurora Serverless v2를 사용하도록 설정하려면 서버리스 (Serverless) DB 인스턴스 클래스를 선택합니다.

그런 다음 최소 및 최대 Aurora 용량 단위(ACU) 값을 지정하여 클러스터의 용량 범위를 설정합니다. 동일한 최소 및 최대 값이 클러스터의 각 Aurora Serverless v2 DB 인스턴스에 적용됩니다. 해당 프로시저와 최소 및 최대 용량 설정의 중요성은 [Aurora Serverless v2 DB 클러스터 생성] 섹션을 참조하세요.

처음 클러스터를 생성할 때 Aurora Serverless v2 DB 인스턴스를 생성하지 않은 경우 나중에 하나 이상의 Aurora Serverless v2 DB 인스턴스를 추가할 수 있습니다. 그러려면 [Aurora Serverless v2 리더 추가] 및 [프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환]의 절차를 따르세요. 첫 번째 Aurora Serverless v2 DB 인스턴스 용량을 클러스터에 추가할 때 용량 범위를 지정합니다. 나중에 [클러스터의 Aurora Serverless v2 용량 설정]의 절차를 수행하여 용량 범위를 변경할 수 있습니다.

CLI

AWS CLI를 사용하여 Aurora Serverless v2 DB 클러스터를 생성할 때는 [create-db-instance] 명령을 사용하여 라이터 DB 인스턴스를 명시적으로 추가합니다. 다음 파라미터를 포함합니다.

- `--db-instance-class db.serverless`

다음 예에서는 Aurora Serverless v2 라이터 DB 인스턴스의 생성을 보여줍니다.

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

Aurora Serverless v2 DB 클러스터 관리

Aurora Serverless v2를 사용하면 클러스터를 프로비저닝된 클러스터와 교환할 수 있습니다. Aurora Serverless v2 속성이 클러스터 내의 하나 이상의 DB 인스턴스에 적용됩니다. 따라서 클러스터 생성, 클러스터 수정, 스냅샷 생성 및 복원 등의 절차는 기본적으로 다른 종류의 Aurora 클러스터와 동일합니

다. Aurora 클러스터 및 DB 인스턴스를 관리하는 일반적인 절차는 Amazon Aurora DB 클러스터 관리 섹션을 참조하세요.

다음 주제에서 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터에 대한 관리 고려 사항에 대해 자세히 알아볼 수 있습니다.

주제

클러스터의 Aurora Serverless v2 용량 설정

Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터 또는 DB 인스턴스 자체에 대한 구성 파라미터 또는 기타 설정을 수정하려면 프로비저닝된 클러스터의 일반적인 절차를 따릅니다. 세부 정보는 Amazon Aurora DB 클러스터 수정을 참조하세요.

Aurora Serverless v2에만 해당하는 가장 중요한 설정은 용량 범위입니다. Aurora 클러스터에 대한 최소 및 최대 Aurora 용량 단위(ACU) 값을 설정한 후에는 클러스터에서 Aurora Serverless v2 DB 인스턴스의 용량을 능동적으로 조정할 필요가 없습니다. Aurora 가 이 작업을 수행합니다. 이 설정은 클러스터 수준에서 관리됩니다. 동일한 최소 및 최대 ACU 값이 클러스터의 각 Aurora Serverless v2 DB 인스턴스에 적용됩니다.

다음 특정 값을 설정할 수 있습니다.

- 최소 ACU - Aurora Serverless v2 DB 인스턴스가 이 ACU 수만큼 용량을 축소할 수 있습니다.
- 최대 ACU - Aurora Serverless v2 DB 인스턴스가 이 ACU 수만큼 용량을 확장할 수 있습니다.

> **Note**
>
> Aurora Serverless v2 DB 클러스터의 용량 범위를 수정하면 변경 사항을 즉시 적용하는 옵션과 예정된 다음 유지 관리 기간에 적용하는 옵션 중 무엇을 선택했든지 변경 사항이 즉시 적용됩니다.

용량 범위의 영향과 이를 모니터링하고 미세 조정하는 방법에 대한 자세한 내용은 Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표 및 Aurora Serverless v2의 성능 및 크기 조정 섹션을 참조하세요. 목표는 클러스터의 최대 용량이 워크로드의 급증을 처리할 수 있을 만큼 충분히 높고, 최소 용량이 클러스터의 사용량이 많지 않을 때 비용을 최소화할 수 있을 만큼 낮도록 하는 것입니다.

모니터링을 기반으로 클러스터 ACU 범위의 높이와 너비를 결정한다고 가정해 보세요. AWS Management Console, AWS CLI 또는 Amazon RDS API를 사용하여 Aurora 클러스터의 용량을 특정 범위의 ACU로 설정할 수 있습니다. 이 용량 범위는 클러스터 내의 모든 Aurora Serverless v2 DB 인스턴스에 적용됩니다.

예를 들어 클러스터의 용량 범위가 1~16ACU이고 Aurora Serverless v2 DB 인스턴스 2개가 포함되어 있다고 가정합시다. 그러면 클러스터 전체로 보면 2ACU(유휴 상태일 경우)와 32ACU(완전히 사용되는 경우) 사이의 용량을 소비합니다. 용량 범위를 8~20.5ACU로 변경하면 이제 클러스터는 유휴 상태일 때 16ACU를 소비하고 완전히 사용되면 최대 41ACU를 소비합니다.

Aurora는 Aurora Serverless v2 DB 인스턴스에 대해 용량 범위 내에서 최대 ACU 값에 따라 자동으로 특정 파라미터 값을 설정합니다. 이런 파라미터 전체 목록은 Aurora Serverless v2의 최대 연결 수 섹션을 참조하세요. 이러한 유형의 계산에 의존하는 정적 파라미터의 경우 DB 인스턴스를 재부팅하면 값이 다시 평가됩니다. 따라서 용량 범위를 변경한 후 DB 인스턴스를 재부팅하여 이러한 파라미터의 값을 업데이트할 수 있습니다. DB 인스턴스를 재부팅하여 파라미터 변경을 적용해야 하는지 확인하려면 DB 인스턴스의 `ParameterApplyStatus` 속성을 확인하세요. `pending-reboot` 값은 재부팅하면 일부 파라미터 값에 변경 사항이 적용됨을 나타냅니다.

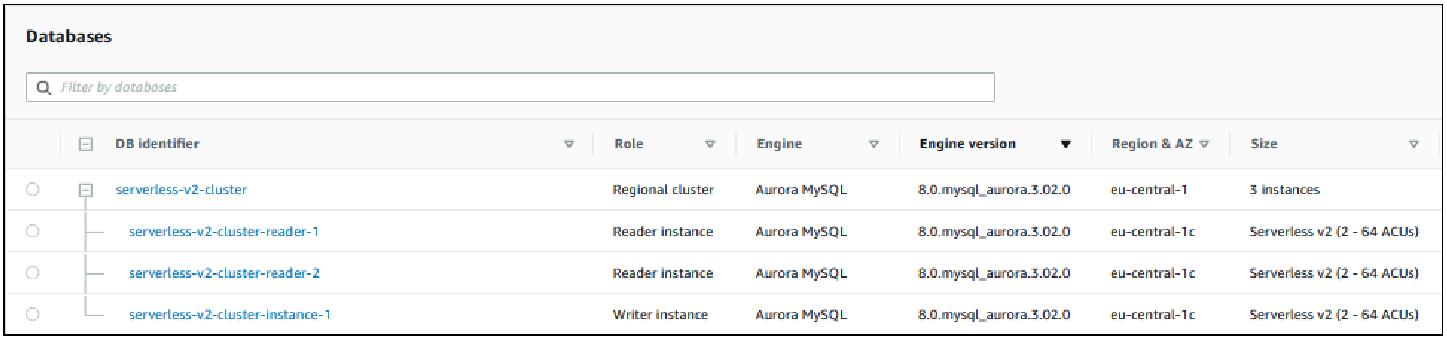
콘솔

AWS Management Console을 사용하여 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터의 용량 범위를 설정할 수 있습니다.

콘솔을 사용하는 경우 클러스터에 첫 번째 Aurora Serverless v2 DB 인스턴스를 추가할 때 클러스터의 용량 범위를 설정합니다. 클러스터를 생성할 때 라이터 DB 인스턴스에 서버리스 v2(Severless v2)를 선택하면서 설정하면 됩니다. 클러스터에 Aurora Serverless v2 리더 DB 인스턴스를 추가할 때 서버리스(Severless) DB 인스턴스를 선택하면서 설정하는 방법도 있습니다. 또는 클러스터에서 기존의 프로

비저닝된 DB 인스턴스를 서버리스(Serverless) DB 인스턴스 클래스로 변환하면서 설정해도 됩니다. 이 절차의 전체 버전은 Aurora Serverless v2 라이터 DB 인스턴스 생성, Aurora Serverless v2 리더 추가, 프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환 섹션을 참조하세요.

클러스터 수준에서 어떤 용량 범위를 설정하든 클러스터 내의 모든 Aurora Serverless v2 DB 인스턴스에 적용됩니다. 다음 이미지는 여러 개의 Aurora Serverless v2 리더 DB 인스턴스가 있는 클러스터를 보여 줍니다. 각각의 용량 범위는 2~64ACU로 동일합니다.



Databases

Filter by databases

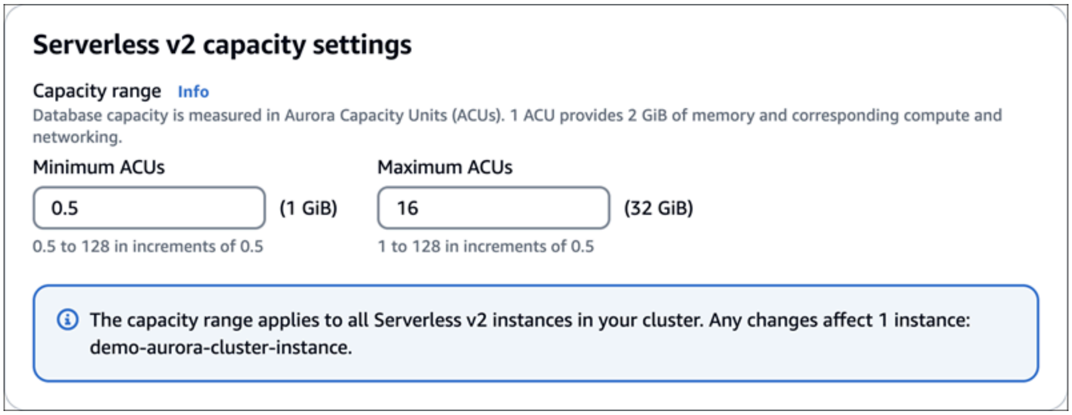
DB identifier	Role	Engine	Engine version	Region & AZ	Size
serverless-v2-cluster	Regional cluster	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1	3 instances
serverless-v2-cluster-reader-1	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)
serverless-v2-cluster-reader-2	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)
serverless-v2-cluster-instance-1	Writer instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)

Aurora Serverless v2 클러스터의 용량 범위를 수정하는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 목록에서 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터를 선택합니다. 클러스터에 하나 이상의 Aurora Serverless v2 DB 인스턴스가 있어야 합니다. 그렇지 않으면 Aurora 에 용량 범위(Capacity range) 섹션이 표시되지 않습니다.
4. 작업에서 수정을 선택합니다.
5. 용량 범위(Capacity range) 섹션에서 다음을 선택합니다.

    a. 최소 ACU(Minimum ACUs)에 값을 입력합니다. 콘솔에 허용되는 값 범위가 표시됩니다. 최소 용량을 0.5~128ACU까지 선택할 수 있습니다. 최대 용량을 1~128ACU까지 선택할 수 있습니다. 용량 값을 0.5ACU단위로 조정할 수 있습니다.

    b. 최대 ACU(Maximum ACUs) 값을 입력합니다. 이 값은 최소 ACU보다 크거나 같아야 합니다. 콘솔에 허용되는 값 범위가 표시됩니다. 다음 그림에 해당 선택이 나와 있습니다.



6. [Continue]를 선택합니다. 수정 요약 페이지가 나타납니다.
7. 즉시 적용을 선택합니다.

   용량 수정은 즉시 적용하는 옵션과 예정된 다음 유지 관리 기간에 적용하는 옵션 중 무엇을 선택했든지 즉시 적용됩니다.
8. [클러스터 수정(Modify cluster)]을 선택하여 수정 사항 요약을 적용합니다. [뒤로(Back)]를 선택하여 변경 사항을 수정하거나 [취소(Cancel)]를 선택하여 변경 사항을 취소할 수도 있습니다.

AWS CLI

AWS CLI를 사용하여 Aurora Serverless v2 DB 인스턴스를 사용할 클러스터의 용량을 설정하려면 [modify-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html) AWS CLI 명령을 실행합니다. `--serverless-v2-scaling-configuration` 옵션을 지정합니다. 클러스터에 이미 하나 이상의 Aurora Serverless v2 DB 인스턴스가 있을 수도 있고 나중에 DB 인스턴스를 추가할 수도 있습니다. `MinCapacity` 및 `MaxCapacity` 필드에 유효한 값은 다음과 같습니다.

- `0.5`, `1`, `1.5`, `2`와 같이 0.5 단위로 최대 128까지 가능합니다.

이 예제에서는 이름이 `sample-cluster`인 Aurora DB 클러스터의 ACU 범위를 최소 `48.5`에서 최대 64로 설정합니다.

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

용량 수정은 즉시 적용하는 옵션과 예정된 다음 유지 관리 기간에 적용하는 옵션 중 무엇을 선택했든지 즉시 적용됩니다.

그런 다음 클러스터에 Aurora Serverless v2 DB 인스턴스를 추가하고 새로운 DB 인스턴스는 각각 48.5~64ACU 범위에서 크기를 조정할 수 있습니다. 새로운 용량 범위는 이미 클러스터에 있던 Aurora Serverless v2 DB 인스턴스에도 적용됩니다. 필요한 경우 DB 인스턴스는 새 용량 범위 내에 속하도록 확장 또는 축소됩니다.

CLI를 사용하여 용량 범위를 설정하는 추가 예제는 Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택 섹션을 참조하세요.

AWS CLI를 사용하여 Aurora Serverless DB 클러스터의 조정 구성을 수정하려면 modify-db-cluster AWS CLI 명령을 실행합니다. 최소 용량, 최대 용량을 구성하도록 `--serverless-v2-scaling-configuration` 옵션을 지정합니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: `0.5`, `1`, `1.5`, `2`와 같이 0.5ACU 단위로 최대 `128`까지 확장 가능합니다.
- Aurora PostgreSQL: `0.5`, `1`, `1.5`, `2`와 같이 0.5ACU 단위로 최대 `128`까지 확장 가능합니다.

다음 예제에서는 이름이 `sample-cluster`인 클러스터에 속한 `sample-instance`라는 Aurora Serverless v2 DB 클러스터의 크기 조정 구성을 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

Windows의 경우:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

RDS API

ModifyDBCluster API 작업을 사용하여 Aurora DB 클러스터의 용량을 설정할 수 있습니다. `ServerlessV2ScalingConfiguration` 파라미터를 지정합니다. `MinCapacity` 및 `MaxCapacity` 필드에 유효한 값은 다음과 같습니다.

- `0.5`, `1`, `1.5`, `2`와 같이 0.5 단위로 최대 `128`까지 가능합니다.

ModifyDBCluster API 작업으로 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터의 크기 조정 구성을 수정할 수 있습니다. 최소 용량, 최대 용량을 구성하도록

ServerlessV2ScalingConfiguration 파라미터를 지정합니다. 유효한 용량 값은 다음과 같습니다.

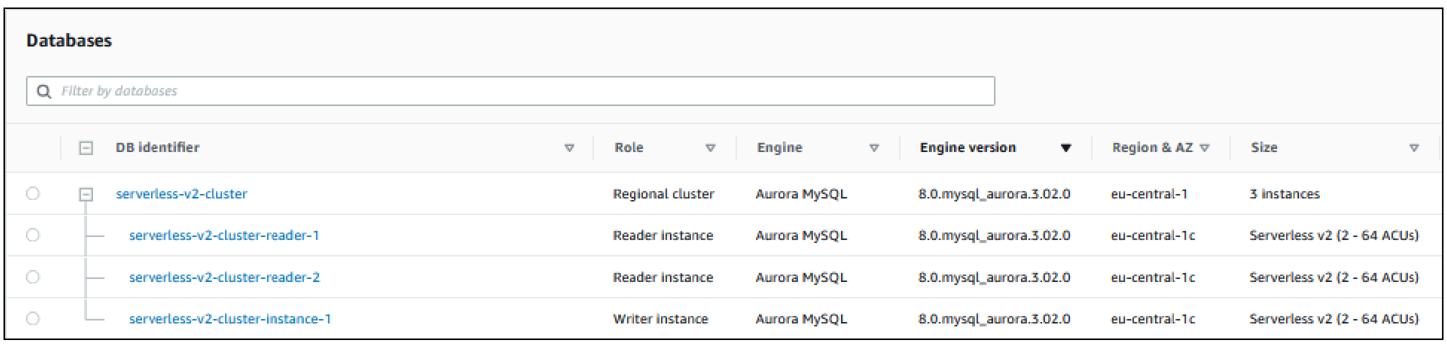
- Aurora MySQL: 0.5, 1, 1.5, 2와 같이 0.5ACU 단위로 최대 128까지 확장 가능합니다.
- Aurora PostgreSQL: 0.5, 1, 1.5, 2와 같이 0.5ACU 단위로 최대 128까지 확장 가능합니다.

용량 수정은 즉시 적용하는 옵션과 예정된 다음 유지 관리 기간에 적용하는 옵션 중 무엇을 선택했든지 즉시 적용됩니다.

Aurora Serverless v2의 용량 범위 확인

Aurora Serverless v2 클러스터의 용량 범위를 확인하는 절차를 따르려면 먼저 용량 범위를 설정해야 합니다. 용량 범위를 아직 설정하지 않았다면 클러스터의 Aurora Serverless v2 용량 설정의 절차를 따르세요.

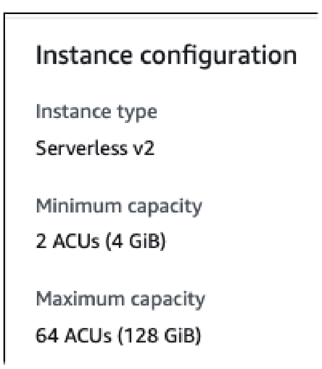
클러스터 수준에서 어떤 용량 범위를 설정하든 클러스터 내의 모든 Aurora Serverless v2 DB 인스턴스에 적용됩니다. 다음 이미지는 여러 개의 Aurora Serverless v2 DB 인스턴스가 있는 클러스터를 보여 줍니다. 각각의 용량 범위는 동일합니다.


Databases

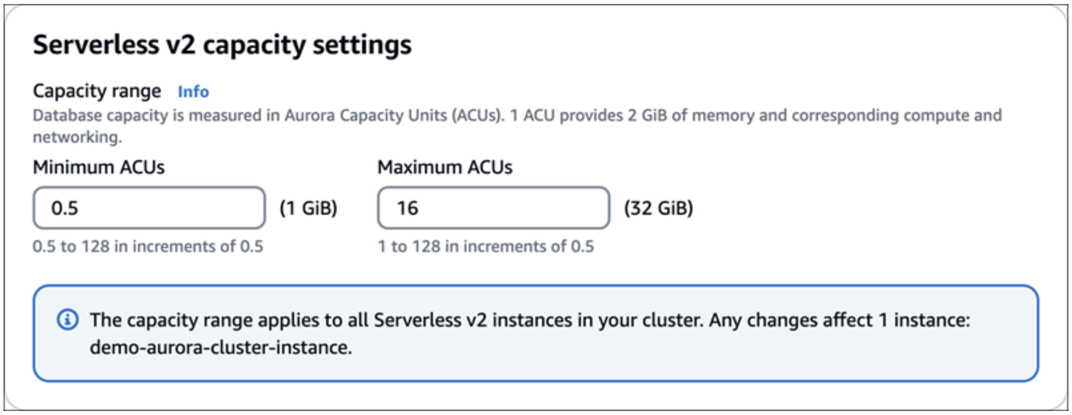
Filter by databases

DB identifier	Role	Engine	Engine version	Region & AZ	Size
serverless-v2-cluster	Regional cluster	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1	3 instances
serverless-v2-cluster-reader-1	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)
serverless-v2-cluster-reader-2	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)
serverless-v2-cluster-instance-1	Writer instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	Serverless v2 (2 - 64 ACUs)

클러스터 내의 Aurora Serverless v2 DB 인스턴스에 대한 세부 정보 페이지도 볼 수 있습니다. DB 인스턴스의 용량 범위가 구성(Configuration) 탭에 나타납니다.



수정(Modify) 페이지에서 클러스터의 현재 용량 범위도 볼 수 있습니다. 다음 이미지는 방법을 보여줍니다. 이 시점에서 용량 범위를 변경할 수 있습니다. 용량 범위를 설정하거나 변경할 수 있는 모든 방법은 [클러스터의 Aurora Serverless v2 용량 설정](#) 섹션을 참조하세요.



Aurora 클러스터의 현재 용량 범위 확인

클러스터의 `ServerlessV2ScalingConfiguration` 속성을 검사하여 클러스터 내의 Aurora Serverless v2 DB 인스턴스에 대해 구성된 용량 범위를 확인할 수 있습니다. 다음 AWS CLI예제는 최소 용량이 0.5ACU(Aurora 용량 단위)이고 최대 용량이 16ACU인 클러스터를 보여줍니다.

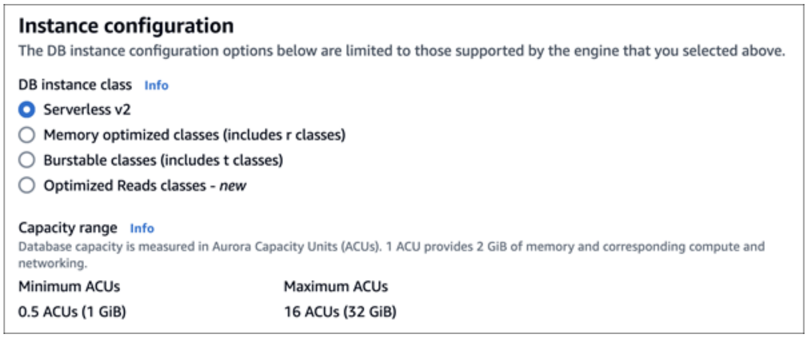
```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

Aurora Serverless v2 리더 추가

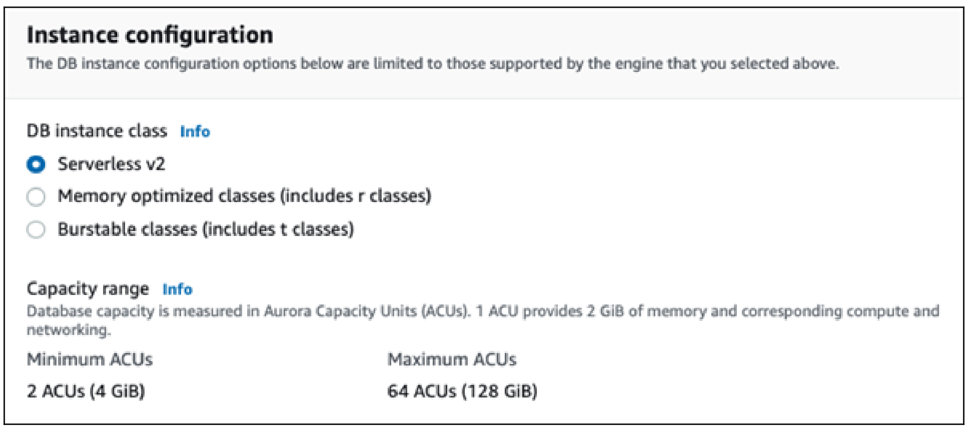
클러스터에 Aurora Serverless v2 리더 DB 인스턴스를 추가하려면 [DB 클러스터에 Aurora 복제본 추가](#)의 일반적인 절차를 따릅니다. 새로운 DB 인스턴스에 서버리스 v2(Severless v2) 인스턴스 클래스를 선택합니다.

리더 DB 인스턴스가 클러스터의 첫 번째 Aurora Serverless v2 DB 인스턴스인 경우 용량 범위도 선택합니다. 다음 이미지는 최소 및 최대 Aurora 용량 단위(ACU)를 지정하는 데 사용하는 컨트롤을 보여줍니다. 이 설정은 이 리더 DB 인스턴스 및 클러스터에 추가하는 다른 Aurora Serverless v2 DB 인스턴스

에 적용됩니다. 각 Aurora Serverless v2 DB 인스턴스는 최소 및 최대 ACU 값 사이에서 크기가 조정될 수 있습니다.



클러스터에 이미 Aurora Serverless v2 DB 인스턴스를 하나라도 추가한 경우 다른 Aurora Serverless v2 리더 DB 인스턴스를 추가하면 현재 용량 범위가 표시됩니다. 다음 이미지는 이러한 읽기 전용 컨트롤을 보여줍니다.



클러스터의 용량 범위를 변경하려면 클러스터의 Aurora Serverless v2 용량 설정의 절차를 따르세요.

두 개 이상의 리더 DB 인스턴스를 포함하는 클러스터의 경우 각 Aurora Serverless v2 리더 DB 인스턴스의 장애 조치 우선순위가 해당 DB 인스턴스의 확장 및 축소 방식에 중요한 역할을 합니다. 클러스터를 처음 생성할 때 우선순위를 지정할 수 없습니다. 클러스터에 두 번째 리더 또는 그 이후의 DB 인스턴스를 추가할 때 이 속성을 염두에 두세요. 자세한 내용은 Aurora Serverless v2 리더에 대한 승격 티어 선택 단원을 참조하십시오.

클러스터의 현재 용량 범위를 볼 수 있는 다른 방법은 Aurora Serverless v2의 용량 범위 확인 섹션을 참조하세요.

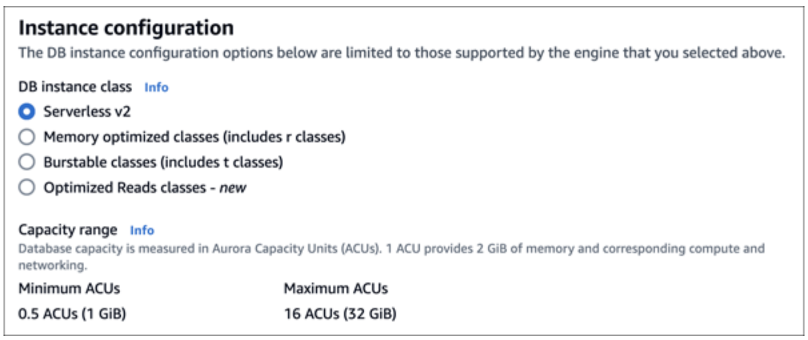
프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환

프로비저닝된 DB 인스턴스가 Aurora Serverless v2를 사용하도록 변환할 수 있습니다. 이 작업을 수행하려면 DB 클러스터에서 DB 인스턴스 수정의 절차를 따르세요. 클러스터는 Aurora Serverless v2 요

구 사항 및 제한 사항의 요구 사항을 충족해야 합니다. 예를 들어 Aurora Serverless v2 DB 인스턴스는 클러스터에서 특정 최소 엔진 버전을 실행하도록 요구합니다.

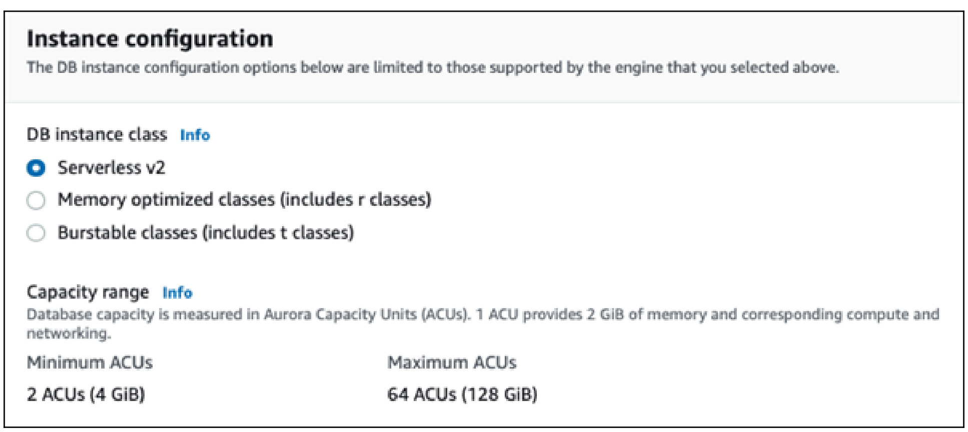
실행 중인 프로비저닝된 클러스터를 변환하여 Aurora Serverless v2의 이점을 활용하려고 한다고 가정해 봅시다. 이 경우 전환 프로세스의 첫 번째 단계로 DB 인스턴스를 Aurora Serverless v2로 변환하여 다운타임을 최소화할 수 있습니다. 전체 절차는 프로비저닝된 클러스터에서 Aurora Serverless v2로 전환 섹션을 참조하세요.

변환하는 DB 인스턴스가 클러스터의 첫 번째 Aurora Serverless v2 DB 인스턴스인 경우 수정 작업의 일환으로 클러스터의 용량 범위를 선택합니다. 이 용량 범위는 클러스터에 추가하는 각 Aurora Serverless v2 DB 인스턴스에 적용됩니다. 다음 이미지는 최소 및 최대 Aurora 용량 단위(ACU)를 지정하는 페이지를 보여줍니다.



용량 범위의 중요성에 대한 자세한 내용은 Aurora Serverless v2 용량 섹션을 참조하세요.

클러스터에 하나 이상의 Aurora Serverless v2 DB 인스턴스가 이미 포함되어 있는 경우 수정 작업 중에 기존 용량 범위가 표시됩니다. 다음 이미지는 해당 정보 패널의 예를 보여줍니다.



더 많은 Aurora Serverless v2 DB 인스턴스를 추가한 후 클러스터의 용량 범위를 변경하려면 클러스터의 Aurora Serverless v2 용량 설정의 절차를 따르세요.

Aurora Serverless v2 라이터 또는 리더를 프로비저닝으로 변환

Aurora Serverless v2 DB 인스턴스를 프로비저닝된 DB 인스턴스로 변환할 수 있습니다. 이 작업을 수행하려면 DB 클러스터에서 DB 인스턴스 수정의 절차를 따르세요. 서버리스(Serverless)가 아닌 DB 인스턴스 클래스를 선택합니다.

Aurora Serverless v2 DB 인스턴스의 최대 Aurora 용량 단위(ACU)에서 사용할 수 있는 것보다 더 큰 용량이 필요한 경우 Aurora Serverless v2 DB 인스턴스를 프로비저닝으로 변환할 수 있습니다. 예를 들어, 가장 큰 db.r5 및 db.r6g DB 인스턴스 클래스의 메모리 용량은 Aurora Serverless v2 DB 인스턴스가 확장할 수 있는 용량보다 더 큽니다.

> **Tip**
>
> db.r3 및 db.t2와 같은 이전 DB 인스턴스 클래스 중 일부는 Aurora Serverless v2에서 사용하는 Aurora 버전에서는 사용할 수 없습니다. Aurora Serverless v2 DB 인스턴스를 프로비저닝으로 변환할 때 사용할 수 있는 DB 인스턴스 클래스를 확인하려면 DB 인스턴스 클래스에 지원되는 DB 엔진 섹션을 참조하세요.

클러스터의 라이터 DB 인스턴스를 Aurora Serverless v2에서 프로비저닝으로 변환하려면 프로비저닝된 클러스터에서 Aurora Serverless v2로 전환의 절차를 역방향으로 따르세요. 클러스터의 리더 DB 인스턴스 중 하나를 Aurora Serverless v2에서 프로비저닝으로 전환합니다. 그런 다음 장애 조치를 수행하여 프로비저닝된 DB 인스턴스를 라이터로 만듭니다.

클러스터에 대해 이전에 지정한 모든 용량 범위는 Aurora Serverless v2 DB 인스턴스가 클러스터에서 제거되더라도 모두 유지됩니다. 용량 범위를 변경하려는 경우 클러스터의 Aurora Serverless v2 용량 설정에 설명된 대로 클러스터를 수정할 수 있습니다.

Aurora Serverless v2 리더에 대한 승격 티어 선택

여러 개의 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터 또는 프로비저닝된 DB 인스턴스와 Aurora Serverless v2 DB 인스턴스가 결합된 클러스터의 경우 각 Aurora Serverless v2 DB 인스턴스의 승격 티어 설정을 유의해야 합니다. 이 설정은 프로비저닝된 DB 인스턴스보다 Aurora Serverless v2 DB 인스턴스에 대해 더 많은 동작을 제어합니다.

AWS Management Console에서 데이터베이스 생성(Create database), 인스턴스 수정(Modify instance) 및 리더 추가(Add reader) 페이지의 추가 구성(Additional configuration)에서 장애 조치 우선순위(Failover priority) 선택을 사용하여 이 설정을 지정합니다. 데이터베이스(Database) 페이지에서 선

택 사항인 우선순위 티어(Priority tier) 열에서 기존 DB 인스턴스에 대해 이 속성이 표시됩니다. DB 클러스터 또는 DB 인스턴스의 세부 정보 페이지에서도 이 속성을 확인할 수 있습니다.

프로비저닝된 DB 인스턴스의 경우 티어 0~15를 선택하면 Aurora가 장애 조치 작업 중에 라이터로 승격할 리더 DB 인스턴스를 선택하는 순서만 결정됩니다. Aurora Serverless v2 리더 DB 인스턴스의 경우, 티어 번호에 따라 DB 인스턴스가 라이터 DB 인스턴스의 용량에 맞게 크기를 조정할지, 아니면 자체 워크로드에 따라 독립적으로 크기를 조정할지도 결정합니다. 티어 0 또는 1의 Aurora Serverless v2 리더 DB 인스턴스는 라이터 DB 인스턴스 용량 이상의 최소 용량으로 유지됩니다. 따라서 장애 조치가 발생할 경우 라이터 DB 인스턴스에서 인계할 준비가 됩니다. 라이터 DB 인스턴스가 프로비저닝된 DB 인스턴스인 경우 Aurora는 이와 동등한 Aurora Serverless v2 용량을 추정하고 이 추정치를 Aurora Serverless v2 리더 DB 인스턴스의 최소 용량으로 사용합니다.

티어 2~15의 Aurora Serverless v2 리더 DB 인스턴스는 최소 용량에 대해 이러한 제약 조건을 가지고 있지 않습니다. 유휴 상태일 때 클러스터의 용량 범위에 지정된 최소 Aurora 용량 단위(ACU) 값으로 축소할 수 있습니다.

다음 Linux AWS CLI 예제에서는 클러스터에 있는 모든 DB 인스턴스의 승격 티어를 검사하는 방법을 보여줍니다. 마지막 필드에는 라이터 DB 인스턴스에 대해 `True` 값, 모든 리더 DB 인스턴스에 대해 `False` 값이 포함됩니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

다음 Linux AWS CLI 예제에서는 클러스터에 있는 특정 DB 인스턴스의 승격 티어를 변경하는 방법을 보여줍니다. 이 명령은 먼저 새 승격 티어로 DB 인스턴스를 수정합니다. 그런 다음 DB 인스턴스를 다시 사용할 수 있을 때까지 기다렸다가 DB 인스턴스의 새 승격 티어를 확인합니다.

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

다양한 사용 사례에 대한 승격 티어 지정에 대한 자세한 지침은 Aurora Serverless v2 크기 조정 섹션을 참조하세요.

Aurora Serverless v2에서 TLS/SSL 사용

Aurora Serverless v2는 TLS/SSL(Transport Layer Security/Secure Sockets Layer) 프로토콜을 사용하여 클라이언트와 Aurora Serverless v2 DB 인스턴스 간의 통신을 암호화합니다. TLS/SSL 버전 1.0, 1.1 및 1.2를 지원합니다. Aurora와 TLS/SSL 사용에 대한 일반적인 정보는 Aurora MySQL DB 클러스터에 대한 TLS 연결 섹션을 참조하세요.

MySQL 클라이언트를 사용하여 Aurora MySQL 데이터베이스에 연결하는 방법에 대한 자세한 내용은 MySQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기를 참조하세요.

Aurora Serverless v2는 다음 표에 나열된 것을 포함하여 MySQL 클라이언트(`mysql`) 및 PostgreSQL 클라이언트(`psql`)에서 사용 가능한 모든 TLS/SSL 모드를 지원합니다.

TLS/SSL 모드에 대한 설명	mysql	psql
TLS/SSL을 사용하지 않고 연결합니다.	비활성화됨	disable
먼저 TLS/SSL을 사용하여 연결을 시도하지만 필요한 경우 비 SSL로 폴백합니다.	PREFERRED	prefer(기본값)
TLS/SSL을 사용하도록 합니다.	필수	require
TLS/SSL을 설정하고 인증 기관(CA)을 확인합니다.	VERIFY_CA	[verify-ca]
TLS/SSL을 적용하고 CA를 확인한 다음 CA 호스트 이름을 확인합니다.	VERIFY_IDENTITY	[verify-full]

Aurora Serverless v2는 와일드카드 인증서를 사용합니다. TLS/SSL을 사용할 때 “CA 확인” 또는 “CA 및 CA 호스트 이름 확인” 옵션을 지정하는 경우 먼저 Amazon Trust Services에서 Amazon 루트 CA 1 신뢰할 수 있는 스토어를 다운로드합니다. 이렇게 하면 클라이언트 명령에서 이 PEM 형식의 파일을 식별할 수 있습니다. PostgreSQL 클라이언트를 사용하여 이렇게 하려면 다음을 수행합니다.

대상 LinuxmacOS, 또는Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Postgres 클라이언트를 사용하여 Aurora PostgreSQL 데이터베이스 작업을 수행하는 방법에 대한 자세한 내용은 PostgreSQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기를 참조하세요.

일반적인 Aurora DB 클러스터에 연결하는 방법에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

Aurora Serverless v2 DB 클러스터에 연결을 위해 지원되는 암호 그룹

구성 가능한 암호 그룹을 사용하면 데이터베이스 연결의 보안을 더 잘 제어할 수 있습니다. 데이터베이스에 대한 클라이언트 TLS/SSL 연결을 보호하도록 허용할 암호 그룹 목록을 지정할 수 있습니다. 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. 이렇게 하면 안전하지 않거나 더 이상 사용되지 않는 암호의 사용을 방지할 수 있습니다.

Aurora MySQL 기반 Aurora Serverless v2 DB 클러스터는 Aurora MySQL 프로비저닝된 DB 클러스터와 동일한 암호 그룹을 지원합니다. 이러한 암호 그룹에 대한 자세한 내용은 Aurora MySQL DB 클러스터 연결을 위한 암호 그룹 구성 섹션을 참조하세요.

Aurora PostgreSQL 기반 Aurora Serverless v2 DB 클러스터는 Aurora PostgreSQL 프로비저닝된 DB 클러스터와 동일한 암호 그룹을 지원합니다. 이러한 암호 그룹에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터 연결을 위한 암호 그룹 구성 섹션을 참조하세요.

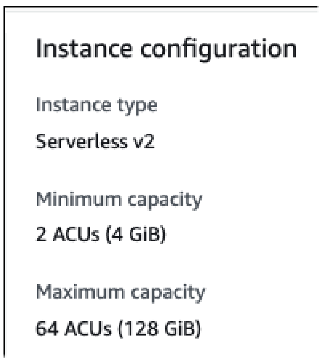
Aurora Serverless v2 라이터 및 리더 보기

프로비저닝된 DB 인스턴스의 세부 정보를 볼 때와 같은 방법으로 Aurora Serverless v2 DB 인스턴스의 세부 정보를 볼 수 있습니다. 이를 수행하려면 Amazon Aurora DB 클러스터 보기의 일반적인 절차를 따르세요. 하나의 클러스터에는 모든 Aurora Serverless v2 DB 인스턴스 또는 모든 프로비저닝된 인스턴스가 포함되거나 둘 다 약간씩 포함될 수 있습니다.

Aurora Serverless v2 DB 인스턴스를 하나 이상 생성하면 유형이 서버리스인 DB 인스턴스와 인스턴스인 DB 인스턴스를 볼 수 있습니다. Aurora Serverless v2 DB 인스턴스가 사용할 수 있는 최소 및 최대 Aurora 용량 단위(ACU)도 볼 수 있습니다. 각 ACU는 처리(CPU) 용량과 메모리(RAM) 용량의 조합입니다. 이 용량 범위는 클러스터 내의 각 Aurora Serverless v2 DB 인스턴스에 적용됩니다. 클러스터 또는 클러스터에 있는 Aurora Serverless v2 DB 인스턴스의 용량 범위를 확인하는 절차는 Aurora Serverless v2의 용량 범위 확인 섹션을 참조하세요.

AWS Management Console에서 Aurora Serverless v2 DB 인스턴스는 데이터베이스(Databases) 페이지의 크기(Size) 열에 표시되어 있습니다. 프로비저닝된 DB 인스턴스는 r6g.xlarge와 같은 DB 인스턴스 클래스의 이름을 보여줍니다. Aurora Serverless DB 인스턴스는 DB 인스턴스 클래스에 대해 서버리스(Serverless)와 함께 DB 인스턴스의 최소 및 최대 용량을 보여줍니다. 예를 들면 서버리스 v2(4~64ACU)(Serverless v2 (4–64 ACUs) 또는 서버리스 v2(1~40ACU)(Serverless v2 (1–40 ACUs))와 같이 표시됩니다.

콘솔에서 각 Aurora Serverless v2 DB 인스턴스의 구성(Configuration) 탭에서 같은 정보를 확인할 수 있습니다. 예를 들어, 다음과 같은 인스턴스 유형(Instance type) 섹션을 볼 수 있습니다. 여기서 인스턴스 유형(Instance type) 값은 서버리스 v2(Severless v2)이고, 최소 용량(Minimum capacity) 값은 2ACU(4GiB)(2 ACUs (4 GiB)), 최대 용량(Maximum capacity) 값은 64ACU(128GiB)(64 ACUs (128 GiB))입니다.



시간의 흐름에 따른 각 Aurora Serverless v2 DB 인스턴스의 용량을 모니터링할 수 있습니다. 이를 통해 각 DB 인스턴스에 사용된 최소, 최대 및 평균 ACU를 확인할 수 있습니다. DB 인스턴스가 최소 또는 최대 용량에 얼마나 근접했는지도 확인할 수 있습니다. AWS Management Console에서 이러한 세부 정보를 보려면 DB 인스턴스의 모니터링(Monitoring) 탭에서 Amazon CloudWatch 지표의 그래프를 살펴봅니다. 살펴볼 지표와 해석 방법에 대한 정보는 Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표 섹션을 참조하세요.

Aurora Serverless v2의 로깅

데이터베이스 로깅을 켜려면 사용자 지정 파라미터 그룹에서 구성 파라미터를 사용할 수 있도록 로그를 지정합니다.

Aurora MySQL의 경우 다음 로그를 활성화할 수 있습니다.

Aurora MySQL	설명
general_log	일반 로그를 생성합니다. 활성화하려면 1로 설정합니다. 기본값은 비활성화(0)입니다.
log_queries_not_using_indexes	인덱스를 사용하지 않는 느린 쿼리 로그에 모든 쿼리를 로깅합니다. 기본값은 비활성화(0)입니다. 이 로그를 활성화하려면 1로 설정합니다.
long_query_time	빠른 실행 쿼리가 느린 쿼리 로그에 로깅되지 않도록 합니다. 0에서 31536000 사이의 부동 소수점 값으로 설정할 수 있습니다. 기본값은 0(비활성화)입니다.
server_audit_events	로그에 캡처할 이벤트 목록입니다. 지원되는 값은 CONNECT, QUERY, QUERY_DCL , QUERY_DDL , QUERY_DML 및 TABLE입니다.
server_audit_logging	서버 감사 로깅을 활성화하려면 1로 설정합니다. 이 옵션을 설정하면 server_audit_events 매개 변수에 감사 이벤트를 CloudWatch 나열하여 보낼 감사 이벤트를 지정할 수 있습니다.
slow_query_log	느린 쿼리 로그를 생성합니다. 느린 쿼리 로그를 활성화하려면 1로 설정합니다. 기본값은 비활성화(0)입니다.

자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.

Aurora PostgreSQL의 경우 Aurora Serverless v2 DB 인스턴스에서 다음 로그를 활성화할 수 있습니다.

Aurora PostgreSQL	설명
log_connections	성공한 연결을 모두 기록합니다.
log_disconnections	시간을 포함해 세션 종료를 로그합니다.

Aurora PostgreSQL	설명
`log_lock_waits`	기본값은 0(비활성화)입니다. 잠금 대기를 로깅하려면 1로 설정합니다.
`log_min_duration_statement`	로깅하기 전에 문을 실행할 최소 시간(밀리초)입니다.
`log_min_messages`	기록되는 메시지 수준을 설정합니다. 지원되는 값은 `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`입니다. 성능 데이터를 `postgres` 로그에 로깅하려면 이 값을 `debug1`로 설정합니다.
`log_temp_files`	지정된 KB(킬로바이트)를 초과하는 임시 파일 사용을 로깅합니다.
`log_statement`	로깅되는 특정 SQL 문을 제어합니다. 지원되는 값은 `none`, `ddl`, `mod` 및 `all`입니다. 기본값은 `none`.

주제

- Amazon CloudWatch를 사용한 로깅
- Amazon CloudWatch에서 Aurora Serverless v2 로그 보기
- 모니터링 용량 Amazon CloudWatch

Amazon CloudWatch를 사용한 로깅

활성화할 데이터베이스 로그를 선택하기 위해 Aurora Serverless v2의 로깅의 절차를 수행한 후 Amazon CloudWatch에 업로드('게시')할 로그를 선택할 수 있습니다.

Amazon CloudWatch를 통해 로그 데이터에 대한 분석을 수행할 수 있으며 경보를 만들고 지표를 볼 수 있습니다. Aurora Serverless v2의 오류 로그는 기본적으로 활성화되고 CloudWatch에 자동으로 업로드됩니다. Aurora Serverless v2 DB 인스턴스에서 다른 로그를 CloudWatch에 업로드하는 것도 가능합니다.

그런 다음 AWS Management Console에서 Log exports(로그 내보내기)를 사용하거나 AWS CLI에서 `--enable-cloudwatch-logs-exports` 옵션을 사용하여 CloudWatch에 업로드할 로그를 선택합니다.

CloudWatch에 업로드할 Aurora Serverless v2 로그를 선택할 수 있습니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 단원을 참조하십시오.

모든 유형의 Aurora DB 클러스터와 마찬가지로 기본 DB 클러스터 파라미터 그룹은 수정할 수 없습니다. 대신 DB 클러스터 및 엔진 유형에 대한 기본 파라미터를 기반으로 DB 클러스터 파라미터 그룹을 생성합니다. 콘솔에서 데이터베이스를 생성할 때 선택할 수 있도록 Aurora Serverless v2 DB 클러스터를 생성하기 전에 사용자 지정 DB 클러스터 파라미터 그룹을 생성하는 것이 좋습니다.

> **Note**
>
> Aurora Serverless v2에서는 DB 클러스터와 DB 파라미터 그룹을 모두 생성할 수 있습니다. 이는 DB 클러스터 파라미터 그룹만 생성할 수 있는 Aurora Serverless v1과 대조적입니다.

Amazon CloudWatch에서 Aurora Serverless v2 로그 보기

활성화할 데이터베이스 로그를 선택하기 위해 Amazon CloudWatch를 사용한 로깅의 절차를 수행한 후 로그의 내용을 볼 수 있습니다.

Aurora MySQL 및 Aurora PostgreSQL 로그에 CloudWatch을 사용하는 방법에 대한 자세한 내용은 Amazon CloudWatch에서 로그 이벤트 모니터링 및 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 섹션을 참조하세요.

Aurora Serverless v2 DB 클러스터에 대한 로그를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. AWS 리전를 선택합니다.
3. 로그 그룹을 선택합니다.
4. 목록에서 Aurora Serverless v2 DB 클러스터 로그를 선택합니다. 로그 이름 지정 패턴은 다음과 같습니다.

   ```
   /aws/rds/cluster/cluster-name/log_type
   ```

> **Note**
>
> Aurora MySQL 호환 Aurora Serverless v2 DB 클러스터의 경우, 오류가 없을 때도 오류 로그에 버퍼 풀 조정 이벤트가 포함됩니다.

모니터링 용량 Amazon CloudWatch

Aurora Serverless v2에서는 CloudWatch를 사용하여 클러스터에 있는 모든 Aurora Serverless v2 DB 인스턴스의 용량과 사용률을 모니터링할 수 있습니다. 인스턴스 수준 지표를 확인하여 각 Aurora Serverless v2 DB 인스턴스가 확장 및 축소할 때 용량을 확인할 수 있습니다. 또한 용량 관련 지표를 다른 지표와 비교하여 워크로드의 변경 사항이 리소스 소비에 미치는 영향을 확인할 수 있습니다. 예를 들면, `ServerlessDatabaseCapacity`와 `DatabaseUsedMemory`, `DatabaseConnections`와 `DMLThroughput`을 비교하여 작업 중 DB 클러스터가 어떻게 응답하는지 평가할 수 있습니다. Aurora Serverless v2에 적용되는 용량 관련 지표에 대한 자세한 내용은 [Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표](#) 섹션을 참조하세요.

Aurora Serverless v2 DB 클러스터의 용량을 모니터링하려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. [지표(Metrics)]를 선택합니다. 사용 가능한 모든 지표는 서비스 이름별로 그룹화된 카드로 콘솔에 표시됩니다.
3. RDS를선택합니다.
4. (선택 사항) 검색(Search) 상자를 사용하여 Aurora Serverless v2에 특히 중요한 지표 검색: `ServerlessDatabaseCapacity`, `ACUUtilization`, `CPUUtilization` 및 `FreeableMemory`.

용량 관련 지표를 사용하여 Aurora Serverless v2 DB 클러스터 용량을 모니터링하도록 CloudWatch 대시보드를 설정하는 것이 좋습니다. 방법에 대한 자세한 내용은 [CloudWatch를 사용하여 대시보드 구축을](#)참조하십시오.

Amazon Aurora와 함께 Amazon CloudWatch를 사용하는 방법에 대한 자세한 내용은 [Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시](#) 섹션을 참조하세요.

Aurora Serverless v2의 성능 및 크기 조정

다음 프로시저와 예는 Aurora Serverless v2 클러스터 및 연결된 DB 인스턴스의 용량 범위를 설정하는 방법을 보여줍니다. 다음 프로시저를 사용하여 DB 인스턴스의 사용량을 모니터링할 수도 있습니다. 그런 다음 결과를 사용하여 용량 범위를 상향 조정해야 하는지 하향 조정해야 하는지 결정할 수 있습니다.

이 프로시저를 사용하기 전에 Aurora Serverless v2 크기 조정이 작동하는 방식을 잘 알고 있어야 합니다. 크기 조정 메커니즘은 Aurora Serverless v1과 다릅니다. 자세한 내용은 Aurora Serverless v2 크기 조정 페이지을 참조하세요.

목차

Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택

Aurora Serverless v2 DB 인스턴스를 사용하면 DB 클러스터에 첫 번째 Aurora Serverless v2 DB 인스턴스를 추가하는 동시에 DB 클러스터의 모든 DB 인스턴스에 적용되는 용량 범위를 설정합니다. 이러한 절차는 클러스터의 Aurora Serverless v2 용량 설정 페이지를 참조하세요.

기존 클러스터의 용량 범위도 변경할 수 있습니다. 다음 섹션에서는 적절한 최소값과 최대값을 선택하는 방법과 용량 범위를 변경하면 어떻게 되는지 자세히 설명합니다. 예를 들어 용량 범위를 변경하면 일부 구성 파라미터의 기본값을 수정할 수 있습니다. 모든 파라미터 변경 사항을 적용하려면 각 Aurora Serverless v2 DB 인스턴스를 재부팅해야 할 수 있습니다.

주제

클러스터에 대한 최소 Aurora Serverless v2 용량 설정 선택

최소한으로 Aurora Serverless v2 용량 설정으로 항상 0.5를 선택하는 것이 좋습니다. 이 값을 사용하면 DB 인스턴스가 완전히 유휴 상태일 때 활성 상태로 유지하면서 최소 용량으로 스케일 다운할 수 있습니다. Aurora Serverless v2에 대해 자동 일시 중지 및 재개를 사용하여 0ACU로 크기 조정에 설명된 대로 최소 용량을 0ACU로 지정하여 자동 일시 중지 동작을 활성화할 수도 있습니다. 그러나 해당 클러스터를 사용하는 방법과 구성하는 기타 설정에 따라 가장 효과적인 최소 용량은 다를 수도 있습니다. 최소 용량 설정을 선택할 때는 다음 요소를 고려하세요.

- Aurora Serverless v2 DB 인스턴스의 크기 조정 비율은 현재 용량에 따라 다릅니다. 현재 용량이 높을수록 더 빠르게 확장할 수 있습니다. DB 인스턴스를 매우 높은 용량으로 빠르게 확장해야 하는 경우 크기 조정 속도가 요구 사항을 충족하는 값으로 최소 용량을 설정하는 것이 좋습니다.
- 일반적으로 특히 높거나 낮은 워크로드를 예상하여 DB 인스턴스의 DB 인스턴스 클래스를 수정하는 경우 해당 경험을 사용하여 동등한 Aurora Serverless v2 용량 범위를 대략적으로 추정할 수 있습니다. 트래픽이 적은 시간에 사용할 메모리 크기를 결정할 때는 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양 페이지를 참조하세요.

  예를 들어 클러스터의 워크로드가 낮은 경우 db.r6g.xlarge DB 인스턴스 클래스를 사용한다고 가정합니다. 해당 DB 인스턴스 클래스에는 32GiB의 메모리가 있습니다. 따라서 최소 Aurora 용량 단위(ACU) 설정을 16으로 지정하여 거의 동일한 용량으로 축소할 수 있는 Aurora Serverless v2 DB 인스턴스를 설정할 수 있습니다. 이는 각 ACU가 약 2GiB의 메모리에 해당하기 때문입니다. db.r6g.xlarge DB 인스턴스가 때때로 충분히 활용되지 않는 경우에 대비하여 DB 인스턴스가 더 축소되도록 약간 더 낮은 값을 지정할 수 있습니다.
- DB 인스턴스의 버퍼 캐시에 일정량의 데이터가 있을 때 애플리케이션이 가장 효율적으로 작동하는 경우 메모리가 자주 액세스하는 데이터를 저장할 수 있을 만큼 충분히 큰 최소 ACU 설정을 지정하는

것이 좋습니다. 그렇지 않으면 Aurora Serverless v2 DB 인스턴스가 더 낮은 메모리 크기로 축소될 때 버퍼 캐시에서 일부 데이터가 제거됩니다. 그런 다음 DB 인스턴스가 다시 확장되면 시간이 지남에 따라 정보가 버퍼 캐시로 다시 읽혀집니다. 데이터를 버퍼 캐시로 다시 가져오기 위한 I/O 양이 많은 경우 최소 ACU 값을 높이는 것이 더 효과적일 수 있습니다.

- Aurora Serverless v2 DB 인스턴스가 특정 용량에서 대부분 실행되는 경우 해당 기준보다 낮지만 너무 낮지 않은 최소 용량 설정을 지정하는 것이 좋습니다. Aurora Serverless v2 DB 인스턴스는 현재 용량이 필요한 용량보다 크게 낮지 않은 경우 확장할 양과 속도를 가장 효과적으로 추정할 수 있습니다.
- 프로비저닝된 워크로드에 T3 또는 T4g와 같은 소규모 DB 인스턴스 클래스에 비해 너무 높은 메모리 요구 사항이 있는 경우 R5 또는 R6g DB 인스턴스에 필적하는 메모리를 제공하는 최소 ACU 설정을 선택합니다.

  특히 지정된 기능과 함께 사용하려면 다음과 같은 최소 용량을 권장합니다(이 권장 사항은 변경될 수 있음).

  - 성능 개선 도우미— 2 ACU
  - Aurora Global Database - 8 ACU(기본 AWS 리전에만 적용)
- Aurora에서 복제는 스토리지 계층에서 발생하므로 리더 용량은 복제에 직접적인 영향을 미치지 않습니다. 그러나 독립적으로 규모가 조정되는 Aurora Serverless v2 리더 DB 인스턴스의 경우, 쿼리 지연 시간을 방지하기 위해 쓰기 집약적인 기간 동안 워크로드를 처리하기에 최소 용량이 충분한지 확인합니다. 승격 티어 2~15의 리더 DB 인스턴스에 성능 문제가 발생하는 경우 클러스터의 최소 용량을 늘리는 것이 좋습니다. 리더 DB 인스턴스가 라이터와 함께 또는 독자적으로 확장되는지를 선택하는 방법에 대한 자세한 내용은 Aurora Serverless v2 리더에 대한 승격 티어 선택 페이지를 참조하세요.
- Aurora Serverless v2 리더 DB 인스턴스가 있는 DB 클러스터가 있는 경우, 리더의 승격 티어가 0 또는 1이 아니면 리더는 라이터 DB 인스턴스와 함께 조정되지 않습니다. 이 경우 최소 용량을 낮게 설정하면 과도한 복제 지연이 발생할 수 있습니다. 데이터베이스가 사용 중일 때 리더의 용량이 라이터의 변경 사항을 적용하기에 충분하지 않을 수 있기 때문입니다. 최소 용량을 라이터 DB 인스턴스와 비슷한 메모리 및 CPU 용량을 나타내는 값으로 설정하는 것이 좋습니다.
- Aurora Serverless v2 DB 인스턴스의 `max_connections` 파라미터 값은 최대 ACU에서 파생된 메모리 크기를 기반으로 합니다. 하지만 PostgreSQL 호환 DB 인스턴스의 최소 용량을 0 또는 0.5ACU로 지정하면 `max_connections`의 최대 용량을 2,000으로 제한됩니다.

  연결이 높은 워크로드에 Aurora PostgreSQL 클러스터를 사용하려는 경우 최소 ACU 설정을 1 이상으로 사용하는 것이 좋습니다. Aurora Serverless v2가 `max_connections` 구성 파라미터를 처리하는 방법에 대한 자세한 내용은 Aurora Serverless v2의 최대 연결 수 페이지를 참조하세요.

- Aurora Serverless v2 DB 인스턴스가 최소 용량에서 최대 용량으로 확장되는 데 걸리는 시간은 최소 ACU 값과 최대 ACU 값의 차이에 따라 다릅니다. DB 인스턴스의 현재 용량이 클 때 Aurora Serverless v2는 DB 인스턴스가 작은 용량에서 시작할 때보다 더 큰 증분으로 크기를 확장합니다. 따라서 상대적으로 큰 최대 용량을 지정하고 DB 인스턴스가 해당 용량 근처에서 대부분의 시간을 보내는 경우 최소 ACU 설정을 높이는 것이 좋습니다. 이렇게 하면 유휴 DB 인스턴스가 더 빠르게 최대 용량까지 다시 확장할 수 있습니다.

클러스터의 최대 Aurora Serverless v2 용량 설정 선택

최대 Aurora Serverless v2 용량 설정에 대해 항상 높은 값을 선택하는 것이 좋습니다. 최대 용량이 크면 DB 인스턴스가 집약적인 워크로드를 실행할 때 가장 많이 확장할 수 있습니다. 낮은 값은 예기치 않은 요금의 가능성을 방지합니다. 해당 클러스터를 사용하는 방법과 구성하는 기타 설정에 따라 가장 효과적인 값은 원래 생각했던 것보다 높거나 낮을 수 있습니다. 최대 용량 설정을 선택할 때는 다음 요소를 고려하세요.

- 최대 용량은 최소 용량보다는 커야 합니다. 최대 및 최소 용량은 동일하게 설정할 수 있습니다. 그러나 이 경우 용량은 절대 확장하거나 축소되지 않습니다. 따라서 최소 및 최대 용량에 대해 동일한 값을 사용하는 것은 테스트 상황 외에는 적절하지 않습니다.
- 최대 용량은 0.5ACU보다 커야 합니다. 대부분의 경우 최소 및 최대 용량을 동일하게 설정할 수 있습니다. 단, 최소값과 최대값 모두에 0.5를 지정할 수는 없습니다. 최대 용량은 값 1 이상이어야 합니다.
- 일반적으로 특히 높거나 낮은 워크로드를 예상하여 DB 인스턴스의 DB 인스턴스 클래스를 수정하는 경우 해당 경험을 사용하여 동등한 Aurora Serverless v2 용량 범위를 추정할 수 있습니다. 트래픽이 많은 시간에 사용할 메모리 크기를 결정하려면 [Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양](#) 페이지를 참조하세요.

  예를 들어 클러스터의 워크로드가 높은 경우 db.r6g.4xlarge DB 인스턴스 클래스를 사용한다고 가정합니다. 해당 DB 인스턴스 클래스에는 128GiB의 메모리가 있습니다. 따라서 최대 ACU 설정을 64로 지정하여 거의 동일한 용량으로 확장할 수 있는 Aurora Serverless v2 DB 인스턴스를 설정할 수 있습니다. 이는 각 ACU가 약 2GiB의 메모리에 해당하기 때문입니다. db.r6g.4xlarge DB 인스턴스에 워크로드를 효과적으로 처리할 수 있는 용량이 충분하지 않은 경우에 대비하여 DB 인스턴스를 더 확장할 수 있도록 다소 높은 값을 지정할 수 있습니다.
- 데이터베이스 사용량에 대한 예산 상한이 있는 경우 모든 Aurora Serverless v2 DB 인스턴스가 항상 최대 용량으로 실행되더라도 해당 상한 내에서 유지되는 값을 선택하세요. 클러스터에 n Aurora Serverless v2 DB 인스턴스가 있는 경우 클러스터가 언제든지 사용할 수 있는 이론상 최대 B1 용량은 Aurora Serverless v2n에 클러스터의 최대 ACU 설정을 곱한 것입니다. (예를 들어 일부 리더가 라이터와 독자적으로 확장하는 경우 실제 소비량은 더 적을 수 있습니다.)

- Aurora Serverless v2 리더 DB 인스턴스를 사용하여 라이터 DB 인스턴스에서 읽기 전용 워크로드의 일부를 오프로드하는 경우 더 낮은 최대 용량 설정을 선택할 수 있습니다. 클러스터에 단일 DB 인스턴스만 포함되어 있는 것처럼 각 리더 DB 인스턴스를 높게 확장할 필요가 없음을 반영하기 위해 이 작업을 수행합니다.
- 애플리케이션에서 잘못 구성된 데이터베이스 파라미터 또는 비효율적인 쿼리로 인한 과도한 사용을 방지하는 경우를 가정해 보겠습니다. 이 경우 설정할 수 있는 절대 최대값보다 낮은 최대 용량 설정을 선택하여 우발적인 남용을 방지할 수 있습니다.
- 실제 사용자 활동으로 인한 스파이크가 드물지만 발생하는 경우 최대 용량 설정을 선택할 때 이러한 경우를 고려할 수 있습니다. 애플리케이션이 전체 성능과 확장성을 유지하여 계속 실행하는 것이 우선 순위인 경우 정상 사용에서 관찰되는 것보다 더 높은 최대 용량 설정을 지정할 수 있습니다. 활동이 극도로 급증하는 동안 애플리케이션이 감소된 처리량으로 실행되는 것이 괜찮다면 약간 더 낮은 최대 용량 설정을 선택할 수 있습니다. 애플리케이션을 계속 실행하기에 충분한 메모리와 CPU 리소스가 있는 설정을 선택했는지 확인하세요.
- 클러스터에서 각 DB 인스턴스의 메모리 사용량을 늘리는 설정을 켜면 최대 ACU 값을 결정할 때 해당 메모리를 고려합니다. 이러한 설정에는 성능 개선 도우미, Aurora MySQL 병렬 쿼리, Aurora MySQL 성능 스키마 및 Aurora MySQL 바이너리 로그 복제에 대한 설정이 포함됩니다. 최대 ACU 값이 Aurora Serverless v2 DB 인스턴스가 해당 기능이 사용 중일 때 워크로드를 처리할 수 있을 만큼 충분히 확장할 수 있도록 허용하는지 확인하세요. 낮은 최대 ACU 설정과 메모리 오버헤드를 부과하는 Aurora 기능의 조합으로 인해 발생하는 문제 해결에 대한 정보는 메모리 부족 오류 방지 페이지를 참조하세요.

예: Aurora MySQL 클러스터의 Aurora Serverless v2 용량 범위 변경

다음 AWS CLI 예제는 기존 Aurora MySQL 클러스터에서 Aurora Serverless v2 DB 인스턴스의 ACU 범위를 업데이트하는 방법을 보여줍니다. 처음에 클러스터의 용량 범위는 8~32 ACU였습니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

DB 인스턴스는 유휴 상태이며 8 ACU로 축소됩니다. 이 시점에서 DB 인스턴스에 다음 용량 관련 설정이 적용됩니다. 버퍼 풀의 크기를 쉽게 읽을 수 있는 단위로 나타내기 위해 2의 30제곱으로 나누어 기비바이트(GiB) 단위로 측정합니다. Aurora의 메모리 관련 측정은 10의 거듭제곱이 아닌 2의 거듭제곱을 기반으로 하는 단위를 사용하기 때문입니다.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

다음으로 클러스터의 용량 범위를 변경합니다. `modify-db-cluster` 명령이 완료된 후 클러스터의 ACU 범위는 12.5~80입니다.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

용량 범위를 변경하면 일부 구성 파라미터의 기본값이 변경됩니다. Aurora는 이러한 새로운 기본값 중 일부를 즉시 적용할 수 있습니다. 그러나 일부 파라미터 변경 사항은 재부팅 후에만 적용됩니다. `pending-reboot` 상태는 일부 파라미터터 변경 사항을 적용하려면 재부팅이 필요함을 나타냅니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

이 시점에서 클러스터는 유휴 상태이고 DB 인스턴스 serverless-v2-instance-1은 12.5 ACU를 사용하고 있습니다. innodb_buffer_pool_size 파라미터는 DB 인스턴스의 현재 용량을 기반으로 이미 조정되었습니다. max_connections 파라미터는 여전히 이전 최대 용량의 값을 반영합니다. 해당 값을 재설정하려면 DB 인스턴스를 재부팅해야 합니다.

> **Note**
>
> 사용자 지정 DB 파라미터 그룹에서 max_connections 파라미터를 직접 설정하는 경우 재부팅할 필요가 없습니다.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
```

```
+----------------+
| 14.5029296875 |
+----------------+
1 row in set (0.00 sec)
```

이제 DB 인스턴스를 재부팅하고 다시 사용할 수 있을 때까지 기다립니다.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

`pending-reboot` 상태가 지워집니다. 값 `in-sync`는 Aurora가 보류 중인 모든 파라미터 변경 사항을 적용했음을 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

유휴 DB 인스턴스에 대해 `innodb_buffer_pool_size` 파라미터가 최종 크기로 증가했습니다. `max_connections` 파라미터는 최대 ACU 값에서 파생된 값을 반영하도록 증가했습니다. Aurora가 `max_connections`에 사용하는 공식은 메모리 크기가 2배가 되면 1,000이 증가합니다.

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
```

```
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

용량 범위를 0.5~128ACU로 설정하고, DB 인스턴스를 재부팅했습니다. 이제 유휴 DB 인스턴스의 버퍼 캐시 크기가 1GiB 미만이므로 이를 메비바이트(MiB) 단위로 측정합니다. `max_connections` 값 5000은 최대 용량 설정의 메모리 크기에서 파생됩니다.

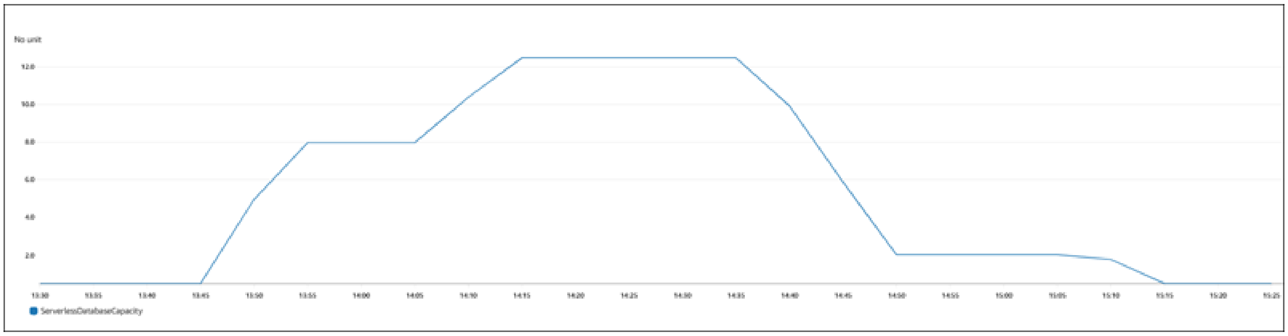
```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

예: Aurora MySQL 클러스터의 Aurora Serverless v2 용량 범위 변경

다음 CLI 예제는 기존 Aurora PostgreSQL 클러스터에서 Aurora Serverless v2 DB 인스턴스의 ACU 범위를 업데이트하는 방법을 보여줍니다.

1. 클러스터의 용량 범위는 0.5~1 ACU에서 시작합니다.
2. 용량 범위를 8~32 ACU로 변경합니다.
3. 용량 범위를 12.5~80 ACU로 변경합니다.
4. 용량 범위를 0.5~128 ACU로 변경합니다.
5. 용량을 초기 범위인 0.5~1 ACU로 되돌립니다.

다음 그림은 Amazon CloudWatch에서의 용량 변화를 보여 줍니다.



DB 인스턴스는 유휴 상태이며 0.5 ACU로 축소됩니다. 이 시점에서 DB 인스턴스에 다음 용량 관련 설정이 적용됩니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

다음으로 클러스터의 용량 범위를 변경합니다. `modify-db-cluster` 명령이 완료된 후 클러스터의 ACU 범위는 8.0~32입니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

용량 범위를 변경하면 일부 구성 파라미터의 기본값이 변경됩니다. Aurora는 이러한 새로운 기본값 중 일부를 즉시 적용할 수 있습니다. 그러나 일부 파라미터 변경 사항은 재부팅 후에만 적용됩니다. `pending-reboot` 상태는 일부 파라미터 변경 사항을 적용하려면 재부팅이 필요함을 나타냅니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

이 시점에서 클러스터는 유휴 상태이고 DB 인스턴스 serverless-v2-instance-1은 8.0 ACU를 사용하고 있습니다. shared_buffers 파라미터는 DB 인스턴스의 현재 용량을 기반으로 이미 조정되었습니다. max_connections 파라미터는 여전히 이전 최대 용량의 값을 반영합니다. 해당 값을 재설정하려면 DB 인스턴스를 재부팅해야 합니다.

> **Note**
>
> 사용자 지정 DB 파라미터 그룹에서 max_connections 파라미터를 직접 설정하는 경우 재부팅할 필요가 없습니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

이제 DB 인스턴스를 재부팅하고 다시 사용할 수 있을 때까지 기다립니다.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
```

```
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

DB 인스턴스가 재부팅되었으므로 이제 `pending-reboot` 상태가 지워집니다. 값 `in-sync`는 Aurora가 보류 중인 모든 파라미터 변경 사항을 적용했음을 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

재부팅이 끝나면 `max_connections`에 새 최대 용량의 값이 표시됩니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

다음으로 클러스터의 용량 범위를 12.5~80 ACU로 변경합니다.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
```

```
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

이 시점에서 클러스터는 유휴 상태이고 DB 인스턴스 `serverless-v2-instance-1`은 12.5 ACU를 사용하고 있습니다. `shared_buffers` 파라미터는 DB 인스턴스의 현재 용량을 기반으로 이미 조정되었습니다. `max_connections` 값은 여전히 5000입니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

다시 재부팅하지만 파라미터 값은 동일하게 유지됩니다. `max_connections`의 경우 Aurora PostgreSQL을 실행하는 Aurora Serverless v2 DB 클러스터의 최대값이 5000이기 때문입니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

이제 용량 범위를 0.5~128 ACU로 설정합니다. DB 클러스터는 10 ACU로 축소된 다음 2 ACU로 축소됩니다. DB 인스턴스를 재부팅합니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 2000
```

```
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2 DB 인스턴스의 max_connections 값은 최대 ACU에서 파생된 메모리 크기를 기반으로 합니다. 하지만 PostgreSQL 호환 DB 인스턴스의 최소 용량을 0 또는 0.5ACU로 지정하면 max_connections의 최대 용량을 2,000으로 제한됩니다.

이제 용량을 최초 범위인 0.5~1 ACU로 되돌리고 DB 인스턴스를 재부팅합니다. max_connections 파라미터가 원래 값으로 돌아갔습니다.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2에 대한 파라미터 그룹 작업

Aurora Serverless v2 DB 클러스터를 생성할 때 특정 Aurora DB 엔진 및 연결된 DB 클러스터 파라미터 그룹을 선택합니다. Aurora가 파라미터 그룹을 사용하여 클러스터 간에 구성 설정을 일관되게 적용하는 방법에 익숙하지 않은 경우 [Amazon Aurora의 파라미터 그룹](https://docs.aws.amazon.com/) 페이지를 참조하세요. 파라미터 그룹에 대한 생성, 수정, 적용 및 기타 작업에 대한 모든 프로시저는 Aurora Serverless v2에 적용됩니다.

파라미터 그룹 기능은 일반적으로 프로비저닝된 클러스터와 Aurora Serverless v2 DB 인스턴스를 포함하는 클러스터 간에 동일하게 작동합니다.

- 클러스터의 모든 DB 인스턴스에 대한 기본 파라미터 값은 클러스터 파라미터 그룹에 의해 정의됩니다.
- 특정 DB 인스턴스에 대한 사용자 지정 DB 파라미터 그룹을 지정하여 특정 DB 인스턴스에 대한 일부 파라미터를 재정의할 수 있습니다. 특정 DB 인스턴스에 대한 디버깅 또는 성능 조정 중에 그렇게

할 수 있습니다. 예를 들어, 일부 Aurora Serverless v2 DB 인스턴스와 일부 프로비저닝된 DB 인스턴스가 포함된 클러스터가 있다고 가정합니다. 이 경우 사용자 지정 DB 파라미터 그룹을 사용하여 프로비저닝된 DB 인스턴스에 대해 몇 가지 다른 파라미터를 지정할 수 있습니다.

- Aurora Serverless v2의 경우 매개변수 그룹의 `SupportedEngineModes` 속성에 값이 `provisioned`인 모든 파라미터를 사용할 수 있습니다. Aurora Serverless v1에서는 `SupportedEngineModes` 속성에 `serverless`가 있는 파라미터의 하위 집합만 사용할 수 있습니다.

주제

기본 파라미터 값

프로비저닝된 DB 인스턴스와 Aurora Serverless v2 DB 인스턴스의 중요한 차이점은 Aurora가 DB 인스턴스 용량과 관련된 특정 파라미터에 대한 모든 사용자 지정 파라미터 값을 재정의한다는 것입니다. 사용자 지정 파라미터 값은 클러스터의 프로비저닝된 DB 인스턴스에 계속 적용됩니다. Aurora Serverless v2 DB 인스턴스가 Aurora 파라미터 그룹의 파라미터를 해석하는 방법에 대한 자세한 내용은 Aurora 클러스터에 대한 구성 파라미터 섹션을 참조하세요. Aurora Serverless v2에서 재정의하는 특정 파라미터는 Aurora Serverless v2의 확장 및 축소에 따라 Aurora가 조정하는 파라미터 및 Aurora가 Aurora Serverless v2 최대 용량을 기반으로 계산하는 파라미터 페이지를 참조하세요.

describe-db-cluster-parameters CLI 명령을 사용하고 AWS 리전을 쿼리하여 다양한 Aurora DB 엔진의 기본 파라미터 그룹의 기본값 목록을 가져올 수 있습니다. 다음은 Aurora Serverless v2와 호환되는 엔진 버전의 `--db-parameter-group-family` 및 `-db-parameter-group-name` 옵션에 사용할 수 있는 값입니다.

데이터베이스 엔진 및 버전	파라미터 그룹 패밀리	기본 파라미터 그룹 이름
Aurora MySQL 버전 3	`aurora-mysql8.0`	`default.aurora-mysql8.0`
Aurora PostgreSQL 버전 13.x	`aurora-postgresql13`	`default.aurora-postgresql13`

데이터베이스 엔진 및 버전	파라미터 그룹 패밀리	기본 파라미터 그룹 이름
Aurora PostgreSQL 버전 14.x	aurora-postgresql14	default.aurora-postgresql14
Aurora PostgreSQL 버전 15.x	aurora-postgresql15	default.aurora-postgresql15
Aurora PostgreSQL 버전 16.x	aurora-postgresql16	default.aurora-postgresql16
Aurora PostgreSQL 버전 17.x	aurora-postgresql17	default.aurora-postgresql17

다음 예제에서는 Aurora MySQL 버전 3 및 Aurora PostgreSQL 13의 기본 DB 클러스터 그룹에서 파라미터 목록을 가져옵니다. 이 버전은 Aurora Serverless v2와 함께 사용하는 Aurora MySQL 및 Aurora PostgreSQL 버전입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

Windows의 경우:

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
```

```
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Aurora Serverless v2의 최대 연결 수

Aurora MySQL 및 Aurora PostgreSQL 모두에서 Aurora Serverless v2 DB 인스턴스는 `max_connections` 파라미터를 일정하게 유지하므로 DB 인스턴스가 축소될 때 연결이 끊어지지 않습니다. 이 파라미터의 기본값은 DB 인스턴스의 메모리 크기를 기반으로 하는 공식에서 파생됩니다. 프로비저닝된 DB 인스턴스 클래스의 공식 및 기본값에 대한 자세한 내용은 Aurora MySQL DB 인스턴스에 대한 최대 연결 및 Aurora PostgreSQL DB 인스턴스에 대한 최대 연결 페이지를 참조하세요.

Aurora Serverless v2는 공식을 평가할 때 현재 ACU 값이 아니라 DB 인스턴스의 최대 Aurora 용량 단위(ACU)를 기반으로 하는 메모리 크기를 사용합니다. 기본값을 변경하는 경우 상수 값을 지정하는 대신 수식의 변형을 사용하는 것이 좋습니다. 이렇게 하면 Aurora Serverless v2는 최대 용량을 기준으로 적절한 크기의 설정을 사용할 수 있습니다.

Aurora Serverless v2 DB 클러스터의 최대 용량을 변경할 때는 Aurora Serverless v2 DB 인스턴스를 재부팅해야 `max_connections` 값이 업데이트됩니다. `max_connections`가 Aurora Serverless v2의 정적 파라미터이기 때문입니다.

다음 표에서는 최대 ACU 값을 기반으로 하는 Aurora Serverless v2의 `max_connections` 기본값을 확인할 수 있습니다.

최대 ACU	Aurora MySQL의 기본 최대 연결 수	Aurora PostgreSQL의 기본 최대 연결 수
1	90	189
4	135	823
8	1,000	1,669
16	2,000	3,360

최대 ACU	Aurora MySQL의 기본 최대 연결 수	Aurora PostgreSQL의 기본 최대 연결 수
32	3,000	5,000
64	4,000	5,000
128	5,000	5,000
192	6,000	5,000
256	6,000	5,000

> **Note**
>
> Aurora Serverless v2 DB 인스턴스의 `max_connections` 값은 최대 ACU에서 파생된 메모리 크기를 기반으로 합니다. 하지만 PostgreSQL 호환 DB 인스턴스의 최소 용량을 0 또는 0.5ACU로 지정하면 `max_connections`의 최대 용량을 2,000으로 제한됩니다.

최대 ACU 값에 따른 `max_connections` 변화를 보여주는 구체적인 예는 [예: Aurora MySQL 클러스터의 Aurora Serverless v2 용량 범위 변경](#) 및 [예: Aurora MySQL 클러스터의 Aurora Serverless v2 용량 범위 변경](#)에서 확인할 수 있습니다.

Aurora Serverless v2의 확장 및 축소에 따라 Aurora가 조정하는 파라미터

Auto Scaling 중에 Aurora Serverless v2는 증가 또는 감소된 용량에 가장 적합하도록 각 DB 인스턴스의 파라미터를 변경할 수 있어야 합니다. 따라서 용량과 관련된 일부 파라미터를 재정의할 수 없습니다. 재정의할 수 있는 일부 파라미터의 경우 고정 값을 하드코딩하지 마세요. 용량과 관련된 이러한 설정에는 다음 고려 사항이 적용됩니다.

Aurora MySQL의 경우 Aurora Serverless v2는 크기 조정 중에 일부 파라미터의 크기를 동적으로 조정합니다. 다음 파라미터의 경우 Aurora Serverless v2는 사용자가 지정하는 사용자 지정 파라미터 값을 사용하지 않습니다.

- `innodb_buffer_pool_size`
- `innodb_purge_threads`
- `table_definition_cache`

- table_open_cache

Aurora PostgreSQL의 경우 Aurora Serverless v2는 크기 조정 중에 다음 파라미터의 크기를 동적으로 조정합니다. 다음 파라미터의 경우 Aurora Serverless v2는 사용자가 지정하는 사용자 지정 파라미터 값을 사용하지 않습니다.

- shared_buffers

여기에 나열된 파라미터 이외의 모든 파라미터에 대해, Aurora Serverless v2 DB 인스턴스는 프로비저닝된 DB 인스턴스와 동일하게 작동합니다. 기본 파라미터 값은 클러스터 파라미터 그룹에서 상속됩니다. 사용자 지정 클러스터 파라미터 그룹을 사용하여 전체 클러스터의 기본값을 수정할 수 있습니다. 또는 사용자 지정 DB 파라미터 그룹을 사용하여 특정 DB 인스턴스의 기본값을 수정할 수 있습니다. 동적 파라미터는 즉시 업데이트됩니다. 정적 파라미터에 대한 변경 사항은 DB 인스턴스를 재부팅한 후에만 적용됩니다.

Aurora가 Aurora Serverless v2 최대 용량을 기반으로 계산하는 파라미터

다음 파라미터에 대해 Aurora PostgreSQL은 max_connections와 마찬가지로 최대 ACU 설정을 기반으로 메모리 크기에서 파생된 기본값을 사용합니다.

- autovacuum_max_workers
- autovacuum_vacuum_cost_limit
- autovacuum_work_mem
- effective_cache_size
- maintenance_work_mem

메모리 부족 오류 방지

Aurora Serverless v2 DB 인스턴스 중 하나가 지속적으로 최대 용량 한도에 도달하는 경우 Aurora는 DB 인스턴스를 incompatible-parameters 상태로 설정하여 이 조건을 나타냅니다. DB 인스턴스가 incompatible-parameters 상태인 동안 일부 작업이 차단됩니다. 예를 들어 엔진 버전을 업그레이드할 수 없습니다.

일반적으로 메모리 부족 오류로 인해 DB 인스턴스가 자주 다시 시작되면 이 상태가 됩니다. Aurora는 이러한 유형의 재시작이 발생할 때 이벤트를 기록합니다. Amazon RDS 이벤트 보기 프로시저에 따라 이벤트를 볼 수 있습니다. 성능 개선 도우미 및 IAM 인증과 같은 설정을 켜는 오버헤드로 인해 비정상

적으로 높은 메모리 사용량이 발생할 수 있습니다. 또한 DB 인스턴스의 과중한 워크로드 또는 많은 수의 스키마 객체와 연결된 메타데이터 관리로 인해 발생할 수 있습니다.

메모리 압력이 낮아져 DB 인스턴스가 최대 용량에 자주 도달하지 않는 경우 Aurora는 자동으로 DB 인스턴스 상태를 다시 `available`로 변경합니다.

이 조건에서 복구하려면 다음 작업 중 일부 또는 전부를 수행할 수 있습니다.

- 클러스터의 최소 Aurora 용량 단위)(ACU) 값을 변경하여 B1 DB 인스턴스의 용량 하한을 늘립니다. 이렇게 하면 유휴 데이터베이스가 클러스터에서 켜진 기능에 필요한 것보다 적은 메모리를 가진 용량으로 축소되는 문제를 피할 수 있습니다. 클러스터의 ACU 설정을 변경한 후 Aurora Serverless v2 DB 인스턴스를 재부팅합니다. 이렇게 하면 Aurora가 상태를 다시 `available`로 재설정할 수 있는지 여부를 평가합니다.
- 클러스터의 최대 ACU 값을 변경하여 Aurora Serverless v2 DB 인스턴스의 용량 상한을 늘립니다. 이렇게 하면 사용량이 많은 데이터베이스가 클러스터 및 데이터베이스 워크로드에서 켜진 기능을 위한 충분한 메모리로 용량을 확장할 수 없는 문제를 피할 수 있습니다. 클러스터의 ACU 설정을 변경한 후 Aurora Serverless v2 DB 인스턴스를 재부팅합니다. 이렇게 하면 Aurora가 상태를 다시 `available`로 재설정할 수 있는지 여부를 평가합니다.
- 메모리 오버헤드가 필요한 구성 설정을 끕니다. 예를 들어 AWS Identity and Access Management(IAM), 성능 개선 도우미 또는 Aurora MySQL 바이너리 로그 복제와 같은 기능이 켜져 있지만 사용하지 않는다고 가정합니다. 이 경우 끌 수 있습니다. 또는 클러스터의 최소 및 최대 용량 값을 더 높게 조정하여 해당 기능에서 사용하는 메모리를 고려할 수 있습니다. 최소 및 최대 용량 설정 선택에 대한 지침은 [Aurora 클러스터의 Aurora Serverless v2 용량 범위 선택](#) 페이지를 참조하세요.
- DB 인스턴스의 워크로드를 줄입니다. 예를 들어 클러스터에 리더 DB 인스턴스를 추가하여 읽기 전용 쿼리의 로드를 더 많은 DB 인스턴스에 분산할 수 있습니다.
- 더 적은 리소스를 사용하도록 애플리케이션에서 사용하는 SQL 코드를 튜닝합니다. 예를 들어 쿼리 계획을 검사하거나 느린 쿼리 로그를 확인하거나 테이블의 인덱스를 조정할 수 있습니다. 기존에 있는 다른 종류의 SQL 튜닝을 수행할 수도 있습니다.

Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표

Aurora Serverless v2 DB 인스턴스용 Amazon CloudWatch를 시작하려면 Amazon CloudWatch에서 Aurora Serverless v2 로그 보기 페이지를 참조하세요. CloudWatch를 통해 Aurora DB 클러스터를 모니터링하는 방법에 대한 자세한 내용은 Amazon CloudWatch에서 로그 이벤트 모니터링 섹션을 참조하세요.

CloudWatch에서 Aurora Serverless v2 DB 인스턴스를 보고 `ServerlessDatabaseCapacity` 지표를 통해 각 DB 인스턴스에서 소비한 용량을 모니터링할 수 있습니다. `DatabaseConnections` 및 `Queries`와 같은 모든 표준 Aurora CloudWatch 지표를 모니터링할 수도 있습니다. Aurora에 대해 모니터링할 수 있는 CloudWatch 지표의 전체 목록은 Amazon Aurora에 대한 Amazon CloudWatch 지표 페이지를 참조하세요. 메트릭은 Amazon Aurora에 대한 클러스터 수준 지표 및 Amazon Aurora에 대한 인스턴스 수준 지표에서 클러스터 수준 및 인스턴스 수준 메트릭으로 나뉩니다.

다음 CloudWatch 인스턴스 수준 지표는 Aurora Serverless v2 DB 인스턴스가 확장 및 축소되는 방식을 이해하기 위해 모니터링하는 데 중요합니다. 이러한 모든 지표는 매초마다 계산됩니다. 그렇게 하면 Aurora Serverless v2 DB 인스턴스의 현재 상태를 모니터링할 수 있습니다. Aurora Serverless v2 DB 인스턴스가 용량과 관련된 지표 임계값에 도달하면 알림을 받도록 경보를 설정할 수 있습니다. 최소 및 최대 용량 설정이 적절한지 또는 조정이 필요한지 결정할 수 있습니다. 데이터베이스의 효율성을 최적화하기 위해 집중할 부분을 결정할 수 있습니다.

- `ServerlessDatabaseCapacity`. 인스턴스 수준 지표로서 현재 DB 인스턴스 용량이 나타내는 ACU 수를 보고합니다. 클러스터 수준의 지표로 클러스터 내 모든 `ServerlessDatabaseCapacity` DB 인스턴스의 Aurora Serverless v2 값 평균을 나타냅니다. 이 지표는 Aurora Serverless v1의 클러스터 수준 지표일 뿐입니다. Aurora Serverless v2에서는 DB 인스턴스 수준과 클러스터 수준에서 사용할 수 있습니다.
- `ACUUtilization`. 이 지표는 Aurora Serverless v2의 새로운 기능입니다. 이 값은 백분율로 표시됩니다. `ServerlessDatabaseCapacity` 지표의 값을 DB 클러스터의 최대 ACU 값으로 나눈 값으로 계산됩니다. 이 지표를 해석하고 조치를 취하려면 다음 지침을 고려하세요.
  - 이 지표가 `100.0` 값에 가까워지면 DB 인스턴스가 가능한 한 높게 확장된 것입니다. 클러스터에 대한 최대 ACU 설정을 늘리는 것이 좋습니다. 이렇게 하면 라이터와 리더 DB 인스턴스 모두 더 높은 용량으로 확장할 수 있습니다.
  - 읽기 전용 워크로드로 인해 리더 DB 인스턴스가 `100.0`의 `ACUUtilization`에 접근하는 반면 라이터 DB 인스턴스는 최대 용량에 근접하지 않는다고 가정합니다. 이 경우 클러스터에 리더 DB 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 DB 인스턴스에 분산시켜 각 리더 DB 인스턴스의 부하를 줄일 수 있습니다.
  - 성능과 확장성이 주요 고려 사항인 프로덕션 애플리케이션을 실행하고 있다고 가정합니다. 이 경우 클러스터의 최대 ACU 값을 높은 값으로 설정할 수 있습니다. 목표는 `ACUUtilization` 지표가 항상 `100.0` 아래에 있도록 하는 것입니다. 최대 ACU 값이 높으면 데이터베이스 활동이 예기치 않게 급증하는 경우에 충분한 공간이 확보되어 있다는 확신을 가질 수 있습니다. 실제로는 사용한 데이터베이스 용량에 대해서만 요금이 청구됩니다.
- `CPUUtilization`. 이 지표는 프로비저닝된 DB 인스턴스와 Aurora Serverless v2에서 다르게 해석됩니다. Aurora Serverless v2의 경우 이 값은 현재 사용 중인 CPU의 양을 DB 클러스터의 최대 ACU

값 아래에서 사용 가능한 CPU 용량으로 나눈 값입니다. Aurora는 이 값을 자동으로 모니터링하고 DB 인스턴스가 CPU 용량의 높은 비율을 지속적으로 사용하는 경우 Aurora Serverless v2 DB 인스턴스를 확장합니다.

이 지표가 `100.0` 값에 가까워지면 DB 인스턴스가 최대 CPU 용량에 도달한 것입니다. 클러스터에 대한 최대 ACU 설정을 늘리는 것이 좋습니다. 이 지표가 리더 DB 인스턴스에서 `100.0` 값에 가까워지는 경우 클러스터에 리더 DB 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 DB 인스턴스에 분산시켜 각 리더 DB 인스턴스의 로드를 줄일 수 있습니다.

- `FreeableMemory`. 이 값은 Aurora Serverless v2 DB 인스턴스가 최대 용량으로 확장될 때 사용 가능한 미사용 메모리의 양을 나타냅니다. 현재 용량이 최대 용량 미만인 모든 ACU에 대해 이 값은 약 2GiB씩 증가합니다. 따라서 DB 인스턴스가 가능한 한 높게 확장될 때까지 이 지표는 0에 접근하지 않습니다.

  이 지표가 `0` 값에 접근하면 DB 인스턴스가 가능한 한 확장되었으며 사용 가능한 메모리 한도에 근접한 것입니다. 클러스터에 대한 최대 ACU 설정을 늘리는 것이 좋습니다. 이 지표가 리더 DB 인스턴스에서 `0` 값에 가까워지는 경우 클러스터에 리더 DB 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 DB 인스턴스에 분산하여 각 리더 DB 인스턴스의 메모리 사용량을 줄일 수 있습니다.

- `TempStorageIOPS`. DB 인스턴스에 연결된 로컬 스토리지에서 수행된 IOPS 수입니다. 여기에는 읽기와 쓰기에 모두 대한 IOPS가 포함됩니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. 이 지표는 Aurora Serverless v2의 새로운 지표입니다. 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표] 페이지을 참조하세요.
- `TempStorageThroughput`. DB 인스턴스와 연결된 로컬 스토리지에서 전송되는 데이터의 양입니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. 이 지표는 Aurora Serverless v2의 새로운 지표입니다. 자세한 내용은 [Amazon Aurora에 대한 인스턴스 수준 지표] 페이지을 참조하세요.

일반적으로 Aurora Serverless v2 DB 인스턴스에 대한 대부분의 확장은 메모리 사용량과 CPU 활동으로 인해 발생합니다. `TempStorageIOPS` 및 `TempStorageThroughput` 지표는 DB 인스턴스와 로컬 스토리지 디바이스 간의 전송에 대한 네트워크 활동이 예상치 못한 용량 증가의 원인이 되는 드문 경우를 진단하는 데 도움이 될 수 있습니다. 다른 네트워크 활동을 모니터링하려면 다음과 같은 기존 지표를 사용할 수 있습니다.

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`

- StorageNetworkThroughput
- StorageNetworkTransmitThroughput

Aurora가 데이터베이스 로그 중 일부 또는 전체를 Amazon CloudWatch Logs에 게시하게 할 수 있습니다. 관련 지침은 사용 중인 데이터베이스 엔진에 따라 다음을 참조하세요.

- the section called “CloudWatch Logs에 Aurora PostgreSQL 로그 게시”
- the section called “CloudWatch Logs에 Aurora MySQL 로그 게시”

Aurora Serverless v2 지표가 AWS 청구서에 적용되는 방식

AWS 청구서의 Aurora Serverless v2 요금은 모니터링할 수 있는 동일한 `ServerlessDatabaseCapacity` 지표를 기반으로 계산됩니다. Aurora Serverless v2 용량을 한 시간의 일부만 사용하는 경우 청구 메커니즘은 이 지표에 대해 계산된 CloudWatch 평균과 다를 수 있습니다. 시스템 문제로 인해 CloudWatch 지표를 짧은 기간 동안 사용할 수 없는 경우에도 다를 수 있습니다. 따라서 청구서에 ACU-시간 값이 `ServerlessDatabaseCapacity` 평균값에서 직접 계산하는 경우와 약간 다를 수 있습니다.

Aurora Serverless v2 지표에 대한 CloudWatch 명령의 예

다음 AWS CLI 예제는 Aurora Serverless v2와 관련된 가장 중요한 CloudWatch 지표를 모니터링하는 방법을 보여줍니다. 각 경우에 `--dimensions` 파라미터의 `Value=` 문자열을 고유한 Aurora Serverless v2 DB 인스턴스의 식별자로 바꿉니다.

다음 Linux 예제는 1시간 동안 10분마다 측정된 DB 인스턴스의 최소, 최대 및 평균 용량 값을 표시합니다. Linux `date` 명령은 현재 날짜 및 시간을 기준으로 시작 및 종료 시간을 지정합니다. `--query` 파라미터의 `sort_by` 함수는 `Timestamp` 필드를 기준으로 결과를 시간순으로 정렬합니다.

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

다음 Linux 예제는 클러스터에 있는 각 DB 인스턴스의 용량 모니터링을 보여줍니다. 각 DB 인스턴스의 최소, 최대 및 평균 용량 사용률을 측정합니다. 측정은 3시간 동안 매시간에 한 번씩 측정됩니다. 이 예에서는 고정된 수의 ACU를 나타내는 `ServerlessDatabaseCapacity` 대신 ACU에 대한 상한의

백분율을 나타내는 ACUUtilization 지표를 사용합니다. 이렇게 하면 용량 범위에서 최소 및 최대 ACU 값의 실제 수치를 알 필요가 없습니다. 0에서 100 사이의 백분율을 볼 수 있습니다.

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

다음 Linux 예제는 이전 예제와 유사한 측정을 수행합니다. 이 경우 측정은 B1 지표에 대한 것입니다. 측정은 1시간 동안 10분마다 수행됩니다. 이 수치는 DB 인스턴스의 최대 용량 설정까지 사용 가능한 CPU 리소스를 기준으로 사용된 사용 가능한 CPU의 백분율을 나타냅니다.

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

다음 Linux 예제는 이전 예제와 유사한 측정을 수행합니다. 이 경우 측정은 B1 지표에 대한 것입니다. 측정은 1시간 동안 10분마다 수행됩니다.

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

성능 개선 도우미로 Aurora Serverless v2 성능 모니터링

성능 개선 도우미를 사용하여 Aurora Serverless v2 DB 인스턴스의 성능을 모니터링할 수 있습니다. 성능 개선 도우미 프로시저는 성능 개선 도우미를 통한 Amazon Aurora 모니터링 페이지를 참조하세요.

다음과 같은 새로운 성능 개선 도우미 카운터가 Aurora Serverless v2 DB 인스턴스에 적용됩니다.

- `os.general.serverlessDatabaseCapacity` - ACU에 있는 DB 인스턴스의 현재 용량입니다. DB 인스턴스에 대한 `ServerlessDatabaseCapacity` CloudWatch 지표에 해당하는 값입니다.
- `os.general.acuUtilization` - 구성된 최대 용량에서 현재 용량의 백분율입니다. DB 인스턴스에 대한 `ACUUtilization` CloudWatch 지표에 해당하는 값입니다.
- `os.general.maxConfiguredAcu` - 이 Aurora Serverless v2 DB 인스턴스에 대해 구성한 최대 용량입니다. ACU로 측정됩니다.
- `os.general.minConfiguredAcu` - 이 Aurora Serverless v2 DB 인스턴스에 대해 구성한 최소 용량입니다. ACU로 측정됩니다.

성능 개선 도우미 카운터의 전체 목록은 성능 개선 도우미 카운터 페이지를 참조하세요.

성능 개선 도우미에서 Aurora Serverless v2 DB 인스턴스에 대해 vCPU 값이 표시되는 경우 해당 값은 DB 인스턴스에 대한 ACU 값을 기반으로 한 추정치를 나타냅니다. 기본 간격인 1분 간격으로 모든 분수 vCPU 값은 가장 가까운 정수로 반올림됩니다. 더 긴 간격의 경우 표시된 vCPU 값은 분당 정수 vCPU 값의 평균입니다.

Aurora Serverless v2 용량 문제 해결

데이터베이스에 부하가 없는데도 Aurora Serverless v2이 최소값으로 축소되지 않는 경우가 있습니다. 원인은 다음과 같습니다.

- 특정 기능을 사용하면 리소스 사용량을 늘리고, 데이터베이스가 최소 용량으로 스케일 다운되는 것을 방지할 수 있습니다. 이러한 기능은 다음과 같습니다.
  - Aurora Global Database
  - CloudWatch 로그 내보내기
  - Aurora PostgreSQL 호환 DB 클러스터에서 `pg_audit` 활성화
  - 확장 모니터링
  - 성능 개선 도우미

자세한 내용은 클러스터에 대한 최소 Aurora Serverless v2 용량 설정 선택 섹션을 참조하세요.

- 리더 인스턴스가 최소 용량으로 축소되지 않고 라이터 인스턴스와 같거나 더 높은 용량을 유지하는 경우, 리더 인스턴스의 우선 순위 등급을 확인하십시오. Aurora Serverless v2 0 또는 1 등급 리더 DB 인스턴스는 라이터 DB 인스턴스보다 같거나 높은 최소 용량으로 유지됩니다. 리더의 우선 순위 등급을 2 이상으로 변경하여 라이터와 별개로 크기를 늘리거나 줄이십시오. 자세한 내용은 Aurora Serverless v2 리더에 대한 승격 티어 선택 섹션을 참조하세요.
- 공유 메모리 크기에 영향을 주는 데이터베이스 파라미터를 기본값으로 설정합니다. 기본값보다 높은 값을 설정하면 공유 메모리 요구 사항이 증가하고 데이터베이스가 최소 용량으로 축소되지 않습니다. 대표적인 예는 `max_connections` 및 `max_locks_per_transaction`입니다.

> **Note**
>
> 공유 메모리 파라미터를 업데이트하면 데이터베이스를 다시 시작해야 변경 사항이 적용됩니다.

- 데이터베이스 워크로드가 많으면 리소스 사용량이 증가할 수 있습니다.
- 데이터베이스 볼륨이 크면 리소스 사용량이 증가할 수 있습니다.

  Amazon Aurora는 DB 클러스터 관리에 메모리와 CPU 리소스를 사용합니다. 데이터베이스 볼륨이 큰 DB 클러스터를 관리하려면 Aurora에 더 많은 CPU와 메모리가 필요합니다. 클러스터의 최소 용량이 클러스터 관리에 필요한 최소 용량보다 작으면 클러스터가 최소 용량으로 스케일 다운되지 않습니다.
- 제거와 같은 백그라운드 프로세스도 리소스 사용량을 높일 수 있습니다.

그래도 데이터베이스가 구성된 최소 용량까지 축소되지 않는다면, 데이터베이스를 중지하고 다시 시작하여 시간이 지남에 따라 형성된 메모리 조각을 회수하십시오. 데이터베이스를 중지하고 다시 시작하면 다운타임이 발생하므로 이 작업은 최소한으로 수행하는 것이 좋습니다.

Aurora Serverless v2에 대해 자동 일시 중지 및 재개를 사용하여 0ACU로 크기 조정

지정된 기간 내에 사용자 활동으로 시작된 연결이 없는 경우 Aurora Serverless v2 DB 인스턴스가 0ACU로 스케일 다운되고 자동으로 일시 중지되도록 지정할 수 있습니다. DB 클러스터에 대해 최소 ACU 값을 0으로 지정하면 됩니다. 인스턴스가 일시 중지된 상태인 동안에는 인스턴스 용량에 대한 요금이 부과되지 않습니다. 가볍게 사용되거나 장시간 활동이 없는 Aurora 클러스터에 대해 자동 일시 중지 및 재개 기능(자동 일시 중지)을 활성화하면 데이터베이스 플릿의 비용을 관리하는 데 도움이 될 수 있습니다.

> **Note**
>
> 자동 일시 중지 기능은 Aurora PostgreSQL과 Aurora MySQL 모두에서 Aurora Serverless v2에 사용할 수 있습니다. 이 기능을 활용하려면 Aurora 데이터베이스 엔진 버전을 업그레이드해야 할 수 있습니다. 최소 용량을 0ACU로 지정할 수 있는 엔진 버전은 Aurora Serverless v2 용량 섹션을 참조하세요.

주제

- Aurora Serverless v2 자동 일시 중지 기능 개요
- Aurora Serverless v2 자동 일시 중지 기능에 대한 사전 조건 및 제한 사항
- 자동 일시 중지 기능 켜기 및 끄기
- Aurora Serverless v2 자동 일시 중지 기능의 작동 방식
- 다양한 유형의 Aurora 클러스터에서 Aurora Serverless v2 자동 일시 중지의 작동 방식
- 자동 일시 중지를 사용하는 Aurora 클러스터 모니터링
- Aurora Serverless v2 자동 일시 중지 문제 해결
- Aurora Serverless v2 자동 일시 중지 기능을 위한 애플리케이션 설계 고려 사항

Aurora Serverless v2 자동 일시 중지 기능 개요

Aurora Serverless v2 DB 인스턴스는 사용자 연결 없이 일정 기간이 지나면 자동으로 일시 중지되고 연결 요청이 도착하면 자동으로 재개될 수 있습니다. Aurora Serverless v2 자동 일시 중지/재개 기능은 엄격한 서비스 수준 목표(SLO)가 없는 시스템의 비용을 관리하는 데 도움이 됩니다. 예를 들어 개발 및 테스트에 사용되는 클러스터에 이 기능을 활성화하거나 데이터베이스가 재개되는 동안 잠시 일시

중지할 수 있는 내부 애플리케이션에 이 기능을 활성화할 수 있습니다. 워크로드에 비활성 기간이 있고 인스턴스가 재개되는 동안 연결 지연을 약간 허용할 수 있는 경우 Aurora Serverless v2 인스턴스에 자동 일시 중지를 사용하여 비용을 절감하는 것을 고려해 보세요.

클러스터의 Aurora Serverless v2 DB 인스턴스의 자동 일시 중지 가능 여부와 각 인스턴스가 일시 중지되기 전에 유휴 상태여야 하는 기간을 지정하여 이 동작을 제어합니다. Aurora 클러스터의 모든 Aurora Serverless v2 DB 인스턴스에 대해 자동 일시 중지 동작을 활성화하려면 클러스터의 최소 용량 값을 0ACU로 설정하세요.

이전에 비활성 기간이 지난 후 0ACU로 크기 조정되는 Aurora Serverless v1 기능을 활용한 경우 Aurora Serverless v2로 업그레이드하고 해당 자동 일시 중지 기능을 사용할 수 있습니다.

자동 일시 중지 기능을 통한 비용 절감의 이점은 클러스터 중지/시작 기능을 사용하는 것과 유사합니다. Aurora Serverless v2에 대한 자동 일시 중지의 경우 추가적인 이점은 중지된 클러스터를 시작하는 것보다 재개가 더 빠르다는 것과 각 DB 인스턴스를 일시 중지하고 재개할 시기를 결정하는 프로세스를 자동화한다는 것입니다.

또한 자동 일시 중지 기능을 사용하면 클러스터 내의 컴퓨팅 리소스 비용을 더 세부적으로 제어할 수 있습니다. 클러스터의 라이터 인스턴스와 다른 리더는 항상 활성 상태를 유지하는 동안에도 일부 리더 인스턴스가 일시 중지되도록 할 수 있습니다. 이렇게 하려면 다른 인스턴스와 독립적으로 일시 중지할 수 있는 리더 인스턴스에 2~15 범위의 장애 조치 우선순위를 할당합니다.

라이터 인스턴스와 장애 조치 우선순위가 0 및 1인 모든 리더 인스턴스는 항상 동시에 일시 중지되고 재개됩니다. 따라서 이 그룹의 인스턴스는 지정된 시간 간격 동안 연결이 없는 경우 일시 중지됩니다.

Aurora DB 클러스터에는 라이터 및 리더 DB 인스턴스와 프로비저닝된 DB 인스턴스 및 Aurora Serverless v2 DB 인스턴스의 조합이 포함될 수 있습니다. 따라서 이 기능을 효과적으로 사용하려면 자동 일시 중지 메커니즘의 다음 측면을 이해하는 것이 좋습니다.

- DB 인스턴스가 자동으로 일시 중지될 수 있는 상황
- DB 인스턴스가 일시 중지되지 않을 수 있는 경우. 예를 들어, 일부 Aurora 기능을 활성화하거나 클러스터에서 특정 종류의 작업을 수행하면 인스턴스에 연결하지 않아도 인스턴스가 일시 중지되지 않을 수 있습니다.
- 인스턴스가 일시 중지된 동안 모니터링 및 결제에 미치는 영향
- DB 인스턴스가 처리를 재개하게 하는 작업
- 일시 중지 및 재개 이벤트가 발생할 때 DB 인스턴스의 용량이 변경되는 방식
- DB 인스턴스가 일시 중지되기 전에 유휴 간격을 제어하는 방법
- DB 인스턴스가 처리를 재개하는 동안 기간을 처리하도록 애플리케이션 로직을 코딩하는 방법

Aurora Serverless v2 자동 일시 중지 기능에 대한 사전 조건 및 제한 사항

자동 일시 중지 기능을 사용하기 전에 사용할 엔진 버전을 확인하세요. 또한 자동 일시 중지가 사용하려는 다른 Aurora 기능과 함께 작동하는지 확인하세요. 자동 일시 중지를 지원되지 않는 엔진 버전을 사용하는 경우 자동 일시 중지를 켤 수 없습니다. 호환되지 않는 기능의 경우 자동 일시 중지와 함께 사용해도 오류가 발생하지 않습니다. 클러스터가 호환되지 않는 기능 또는 설정을 사용하는 경우 Aurora Serverless v2 인스턴스가 자동으로 일시 중지되지 않습니다.

- Aurora PostgreSQL을 사용하는 경우 데이터베이스 엔진이 버전 16.3, 15.7, 14.12 또는 13.15 이상에서 실행 중이어야 합니다.
- Aurora MySQL을 사용하는 경우 데이터베이스 엔진이 버전 3.08.0 이상을 실행 중이어야 합니다.
- 이 기능을 사용할 수 있는 엔진 버전 및 AWS 리전의 전체 목록은 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.
- Aurora Serverless v2 인스턴스가 재개되면 인스턴스가 일시 중지되었을 때보다 용량이 낮을 수 있습니다. 세부 정보는 Aurora Serverless v2와 Aurora Serverless v1의 자동 일시 중지 동작 차이을 참조하세요.

특정 조건 또는 설정으로 인해 Aurora Serverless v2 인스턴스가 자동으로 일시 중지되지 않습니다. 자세한 내용은 Aurora Serverless v2가 자동 일시 중지되지 않는 상황 단원을 참조하십시오.

자동 일시 중지 기능 켜기 및 끄기

클러스터 수준에서 자동 일시 중지 기능을 켜거나 끌 수 있습니다. 이렇게 하려면 클러스터의 최소 및 최대 용량을 조정할 때와 동일한 프로시저를 사용합니다. 자동 일시 중지 기능은 최소 용량인 0ACU로 표시됩니다.

주제

클러스터의 Aurora Serverless v2 인스턴스에 대해 자동 일시 중지 켜기

클러스터의 Aurora Serverless v2 용량 설정의 프로시저를 따르세요. 최소 용량으로 0ACU를 선택합니다. 최소 용량을 0ACU로 선택하면 인스턴스가 자동으로 일시 중지되기 전의 유휴 상태 시간도 지정할 수 있습니다.

다음 CLI 예시에서는 자동 일시 중지 기능이 활성화되고 자동 일시 중지 간격이 10분(600초)으로 설정된 Aurora 클러스터를 만드는 방법을 보여줍니다.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=4,SecondsUntilAutoPause=600 \
    --master-username myuser \
    --manage-master-user-password
```

다음 CLI 예시에서는 기존 Aurora 클러스터에 대해 자동 일시 중지 기능을 켤 수 있는 방법을 보여줍니다. 이 예시에서는 자동 일시 중지 간격을 1시간(3,600초)으로 설정합니다.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=80,SecondsUntilAutoPause=3600

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 0,
    "MaxCapacity": 80.0,
    "SecondsUntilAutoPause": 3600
}
```

구성 가능한 Aurora Serverless v2 자동 일시 중지 제한 시간 간격

제한 시간 간격은 Aurora 클러스터의 `ServerlessV2ScalingConfiguration` 속성에 표시됩니다. 최소 용량이 0ACU로 설정된 경우 Aurora 클러스터를 만들거나 수정할 때 AWS Management Console에서이 간격을 지정할 수 있습니다. Aurora 클러스터를 만들거나 수정할 때 `--serverless-v2-scaling-configuration` 파라미터를 사용하여 AWS CLI에서 이 간격을 지정할 수 있습니다.

Aurora 클러스터를 만들거나 수정할 때 `ServerlessV2ScalingConfiguration` 파라미터를 사용하여 RDS API에서 이 간격을 지정할 수 있습니다.

설정할 수 있는 최소 간격은 300초(5분)입니다. 간격을 지정하지 않으면 이 값이 기본값입니다. 설정할 수 있는 최대 간격은 86,400초(1일)입니다.

클러스터의 최소 용량을 0이 아닌 값으로 변경하여 클러스터의 자동 일시 중지 기능을 끄는 경우를 가정해 보겠습니다. 이 경우 간격 속성이 `ServerlessV2ScalingConfiguration` 속성에서 제거됩니다. 해당 속성이 없으면 해당 클러스터에 대해 자동 일시 중지 기능이 꺼져 있음을 추가로 확인할 수 있습니다. 나중에 자동 일시 중지를 다시 켜면 그 시점에 사용자 지정 간격을 다시 지정할 수 있습니다.

자동 일시 중지된 Aurora Serverless v2 인스턴스 재개

- 일시 중지된 Aurora Serverless v2 인스턴스에 연결하면 인스턴스가 자동으로 재개되고 연결을 수락합니다.
- 유효한 자격 증명이 포함되지 않은 연결을 시도하더라도 DB 인스턴스가 재개됩니다.
- 라이터 엔드포인트를 통해 연결하면 Aurora는 라이터 DB 인스턴스가 자동 일시 중지된 경우 재개합니다. 동시에 Aurora는 장애 조치 우선순위가 0 또는 1인 자동 일시 중지된 리더 인스턴스를 재개합니다. 즉, 용량은 라이터 인스턴스의 용량과 연결됩니다.
- 리더 엔드포인트를 통해 연결하는 경우 Aurora는 리더 인스턴스를 무작위로 선택합니다. 해당 리더 인스턴스가 일시 중지되면 Aurora가 인스턴스를 재개합니다. 또한 Aurora는 라이터 인스턴스를 먼저 재개합니다. 활성 상태인 리더 인스턴스가 있으면 라이터 인스턴스가 항상 활성 상태여야 하기 때문입니다. Aurora가 해당 라이터 인스턴스를 재개하면 장애 조치 승격 계층이 0 및 1인 리더 인스턴스도 재개됩니다.
- RDS 데이터 API를 통해 클러스터에 요청을 보내면 라이터 인스턴스가 일시 중지된 경우 Aurora가 재개합니다. 그런 다음 Aurora는 데이터 API 요청을 처리합니다.
- DB 클러스터 파라미터 그룹에서 구성 파라미터의 값을 변경하면 Aurora는 해당 클러스터 파라미터 그룹을 사용하는 클러스터에서 일시 중지된 Aurora Serverless v2 인스턴스를 자동으로 재개합니다. 마찬가지로 DB 파라미터 그룹에서 파라미터 값을 변경하면 Aurora는 해당 DB 파라미터 그룹을 사용하는 일시 중지된 Aurora Serverless v2 인스턴스를 자동으로 재개합니다. 다른 클러스터 파라미터 그룹을 할당하도록 클러스터를 수정하거나 다른 DB 파라미터 그룹을 할당하도록 인스턴스를 수정할 때도 동일한 자동 재개 동작이 적용됩니다.
- 역추적 요청을 수행하면 Aurora Serverless v2 라이터 인스턴스가 일시 중지된 경우 자동으로 재개됩니다. Aurora는 라이터 인스턴스가 재개된 후 역추적 요청을 처리합니다. Aurora Serverless v2 인스턴스가 일시 중지된 시간으로 역추적할 수 있습니다.
- 클러스터 스냅샷을 만들거나 스냅샷을 삭제해도 Aurora Serverless v2 인스턴스가 재개되지 않습니다.

- Aurora 복제본을 만들면 Aurora가 복제되는 클러스터의 라이터 인스턴스를 재개합니다.
- 일시 중지된 인스턴스가 재개를 완료하기 전에 많은 수의 연결 요청을 수신하면 일부 세션이 연결되지 않을 수 있습니다. 자동 일시 중지가 활성화된 일부 Aurora Serverless v2 인스턴스가 있는 Aurora 클러스터에 연결을 위한 재시도 로직을 구현하는 것이 좋습니다. 예를 들어 실패한 연결을 세 번 재시도할 수 있습니다.
- Aurora는 인스턴스를 깨우지 않고도 일부 유형의 사소한 내부 유지 관리를 수행할 수 있습니다. 그러나 클러스터의 유지 관리 기간 동안 발생하는 일부 유형의 유지 관리에서는 Aurora가 인스턴스를 재개해야 합니다. 유지 관리가 완료되면 지정된 간격 이후에 더 이상 활동이 없으면 인스턴스가 자동으로 일시 중지됩니다.

> **Note**
>
> Aurora는 PostgreSQL `pg_cron` 확장 또는 MySQL 이벤트 스케줄러와 같은 엔진별 예약 작업을 위해 일시 중지된 인스턴스를 자동으로 재개하지 않습니다. 이러한 작업이 실행되도록 하려면 예약된 시간 전에 인스턴스에 수동으로 연결을 시작하세요. Aurora는 DB 인스턴스가 일시 중지되어 있는 동안 예약된 시간이 도래하는 작업을 대기열에 추가하지 않습니다. 인스턴스가 나중에 재개되면 이러한 작업은 건너뜁니다.

- Aurora Serverless v2 인스턴스가 자동 일시 중지된 동안 Aurora 클러스터가 장애 조치를 거치는 경우 Aurora는 인스턴스를 재개한 다음 해당 인스턴스를 라이터로 승격할 수 있습니다. 인스턴스가 일시 중지된 동안 클러스터에서 하나 이상의 DB 인스턴스가 제거되는 경우에도 마찬가지일 수 있습니다. 이 경우 인스턴스가 재개되면 즉시 라이터가 됩니다.
- 클러스터의 속성을 변경하는 작업으로 인해서도 자동 일시 중지된 Aurora Serverless v2 인스턴스가 재개됩니다. 예를 들어 자동 일시 중지된 인스턴스는 다음과 같은 작업을 위해 재개됩니다.
  - 클러스터의 크기 조정 범위 변경
  - 클러스터의 엔진 버전 업그레이드
  - 일시 중지된 인스턴스에서 로그 파일 설명 또는 다운로드. CloudWatch로의 로그 업로드를 활성화하고 CloudWatch를 통해 로그를 분석하여 일시 중지된 인스턴스의 기록 로그 데이터를 검사할 수 있습니다.

클러스터의 Aurora Serverless v2 인스턴스에 대해 자동 일시 중지 끄기

클러스터의 Aurora Serverless v2 용량 설정의 프로시저를 따르세요. 최소 용량에서 0.5 이상의 값을 선택합니다. 자동 일시 중지 기능을 끄면 인스턴스가 유휴 상태가 되는 간격이 재설정됩니다. 자동 일시 중지를 다시 켜면 새로운 제한 시간 간격을 지정하게 됩니다.

다음 CLI 예시에서는 기존 Aurora 클러스터에 대해 자동 일시 중지 기능을 끌 수 있는 방법을 보여줍니다. describe-db-clusters 출력은 최소 용량이 0이 아닌 값으로 설정된 경우 SecondsUntilAutoPause 속성이 제거됨을 보여줍니다.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=2,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 2,
    "MaxCapacity": 80.0
}
```

Aurora Serverless v2 자동 일시 중지 기능의 작동 방식

다음 정보를 사용하여 자동 일시 중지 기능의 사용을 계획할 수 있습니다. 인스턴스가 일시 중지되거나 재개되거나 활성 상태를 유지하는 상황을 이해하면 가용성, 응답성 및 비용 절감 간의 균형을 유지하는 데 도움이 될 수 있습니다.

주제

Aurora Serverless v2 인스턴스가 일시 중지되면 발생하는 상황

DB Aurora Serverless v2 인스턴스가 연결 없이 일정 기간 후에 일시 중지되는 경우:

- Aurora는 인스턴스에 있는 ACU 수와 관계없이 인스턴스에 대한 연결 없이 지정된 기간이 경과하면 인스턴스 일시 중지를 시작합니다.
- 일시 중지 메커니즘은 즉각적이지 않습니다. 자동 일시 중지될 Aurora Serverless v2 인스턴스는 잠시 기다렸다가 Aurora 스토리지의 모든 변경 사항을 따라잡을 수 있습니다.

- 해당 인스턴스에 대한 인스턴스 요금은 보류됩니다. 인스턴스가 일시 중지된 동안 `ServerlessV2Usage` 지표의 값은 0입니다.
- 인스턴스의 상태 값은 변경되지 않습니다. 상태는 여전히 '사용 가능'으로 표시됩니다.
- 인스턴스가 데이터베이스 로그 파일에 대한 쓰기를 중지합니다. `CPUUtilization` 및 `ACUUtilization`의 경우 0%, `ServerlessDatabaseCapacity`의 경우 0을 등록하는 것 외에는 CloudWatch로 지표 전송을 중지합니다.
- Aurora는 Aurora Serverless v2 DB 인스턴스가 일시 중지를 시작하고, 일시 중지를 완료하고, 일시 중지 메커니즘이 중단되거나 실패할 때 이벤트를 내보냅니다. 이러한 이벤트에 대한 자세한 내용은 DB 인스턴스 이벤트 섹션을 참조하세요.

자동 일시 중지된 Aurora Serverless v2 인스턴스가 재개되면 발생하는 상황

Aurora Serverless v2 DB 인스턴스가 자동으로 일시 중지된 후 재개되면 다음 조건이 적용됩니다.

- `pending-reboot` 변경 사항에 포함된 파라미터 변경 사항은 인스턴스가 재개될 때 적용됩니다.
- Aurora는 각 Aurora Serverless v2 DB 인스턴스가 재개를 시작하고, 재개를 완료하고, 어떤 이유로든 인스턴스를 재개할 수 없을 때 인스턴스 수준 이벤트를 내보냅니다. 이러한 이벤트에 대한 자세한 내용은 DB 인스턴스 이벤트 섹션을 참조하세요.
- 요청된 연결은 DB 인스턴스가 재개를 완료한 후 설정됩니다. 일반적인 재개 소요 시간은 약 15초일 수 있으므로 클라이언트 제한 시간 설정을 15초보다 길게 조정하는 것이 좋습니다. 예를 들어 JDBC 드라이버 설정에서 `connectTimeout` 및 `sslResponseTimeout` 설정의 값을 15초보다 길게 조정할 수 있습니다.

> **Note**
>
> Aurora Serverless v2 인스턴스가 24시간 이상 일시 중지된 상태로 남아 있는 경우 Aurora는 인스턴스를 재개하는 데 더 오래 걸리는 더 깊은 휴면 상태로 전환할 수 있습니다. 이 경우 재개 소요 시간은 30초 이상일 수 있으며, 이는 인스턴스를 재부팅하는 데 소요되는 시간과 거의 동일합니다. 데이터베이스에 하루 이상 지속되는 비활성 기간이 있는 경우 연결 제한 시간을 30초 이상으로 설정하는 것이 좋습니다.

자동 일시 중지된 Aurora Serverless v2 클러스터에 대한 인스턴스 요금 청구 작동 방식

Aurora Serverless v2 인스턴스가 자동 일시 중지된 동안 인스턴스 요금은 0입니다. `ServerlessV2Usage` 지표는 이 기간 동안 0입니다. AWS는 해당 DB 인스턴스에 연결되지 않은 클러스터의 Aurora 스토리지 및 기타 측면에 대해서는 여전히 요금을 청구합니다.

Aurora Serverless v2가 자동 일시 중지되지 않는 상황

- DB 클러스터의 최소 용량 값이 0ACU보다 높으면 클러스터의 Aurora Serverless v2 인스턴스가 자동으로 일시 중지되지 않습니다. 자동 일시 중지 기능을 사용할 수 있기 전부터 있던 Aurora Serverless v2 인스턴스가 포함된 기존 클러스터의 경우 최소 용량 설정은 0.5ACU였습니다. 이러한 클러스터에서 자동 일시 중지 기능을 사용하려면 최소 용량 설정을 0ACU로 수정하세요.
- 사용자가 시작한 연결이 Aurora Serverless v2 인스턴스에 열려 있으면 인스턴스가 일시 중지되지 않습니다. 또한 패치 적용 및 업그레이드와 같은 활동이 진행되는 동안에는 인스턴스가 일시 중지되지 않습니다. Aurora가 상태 확인에 사용하는 관리 연결은 활동으로 간주되지 않으며 인스턴스 일시 중지를 방지하지 않습니다.
- 논리적 복제가 활성화된 Aurora PostgreSQL 클러스터 또는 binlog 복제가 활성화된 Aurora MySQL 클러스터에서 라이터 인스턴스와 장애 조치 승격 계층이 0 및 1인 리더 인스턴스는 자동으로 일시 중지되지 않습니다. Aurora는 복제 연결의 상태를 확인하기 위해 지속적으로 최소한의 활동을 수행합니다.

  복제가 활성화된 클러스터의 경우 소스 클러스터에 자동 일시 중지되는 Aurora 리더 인스턴스를 여전히 보유할 수 있습니다. 이렇게 하려면 장애 조치 우선순위를 0 또는 1 이외의 값으로 설정합니다. 그러면 라이터 인스턴스와 독립적으로 일시 중지될 수 있습니다.
- Aurora 클러스터에 연결된 RDS Proxy가 있는 경우 프록시는 클러스터의 각 DB 인스턴스에 대해 열린 연결을 유지합니다. 따라서 이러한 클러스터의 Aurora Serverless v2 인스턴스는 자동으로 일시 중지되지 않습니다.
- 클러스터가 Aurora 글로벌 데이터베이스의 기본 클러스터인 경우 Aurora는 Aurora Serverless v2 라이터 인스턴스를 자동으로 일시 중지하지 않습니다. 글로벌 데이터베이스의 다른 클러스터를 관리하기 위해 라이터 인스턴스에 일정한 수준의 활동이 필요하기 때문입니다. 라이터 인스턴스는 활성 상태로 유지되므로 장애 조치 우선순위가 0 또는 1인 Aurora Serverless v2 리더 인스턴스도 자동 일시 중지되지 않습니다.
- Aurora 글로벌 데이터베이스의 보조 클러스터에 있는 Aurora Serverless v2 인스턴스는 자동으로 일시 중지되지 않습니다. DB 클러스터가 독립 실행형 클러스터로 승격될 경우 다른 모든 조건이 충족되면 자동 일시 중지 기능이 적용됩니다.
- Redshift와 연결된 제로 ETL 통합이 있는 클러스터에서 라이터 인스턴스와 장애 조치 승격 계층이 0 및 1인 리더 인스턴스는 자동으로 일시 중지되지 않습니다.

- Aurora PostgreSQL 인스턴스에 Babelfish 기능이 활성화된 경우 인스턴스의 기본 데이터베이스 포트 활동 외에도 T-SQL 포트의 연결 및 활동은 인스턴스가 자동으로 일시 중지되는 것을 방지합니다.

클러스터 중지/시작 기능을 사용한 자동 일시 중지의 작동 방식

자동 일시 중지 기능이 활성화된 경우 Aurora 클러스터를 중지하고 시작할 수 있습니다. 일부 인스턴스가 일시 중지되어 있어도 상관없습니다. 클러스터를 다시 시작하면 일시 중지된 Aurora Serverless v2 인스턴스가 자동으로 재개됩니다.

Aurora 클러스터가 중지된 동안 일시 중지된 Aurora Serverless v2 인스턴스는 연결 시도에 따라 자동으로 재개되지 않습니다. 클러스터가 재개되면 Aurora Serverless v2 인스턴스 일시 중지 및 재개를 위한 일반적인 메커니즘이 적용됩니다.

자동 일시 중지된 Aurora Serverless v2 클러스터에 대한 유지 관리 및 업그레이드의 작동 방식

- Aurora Serverless v2 인스턴스가 자동으로 일시 중지된 동안 Aurora 클러스터를 업그레이드하려고 하면 Aurora가 인스턴스를 재개하고 업그레이드합니다.
- Aurora는 마이너 버전 업그레이드 및 파라미터 그룹과 같은 속성 변경과 같은 유지 관리를 수행하기 위해 자동 일시 중지된 Aurora Serverless v2 인스턴스를 주기적으로 재개합니다.
- 업그레이드 또는 유지 관리 적용과 같은 관리 작업을 위해 Aurora Serverless v2 인스턴스가 재개되면 Aurora는 해당 인스턴스를 다시 일시 중지하기 전에 20분 이상 기다립니다. 백그라운드 작업이 있는 경우 마무리될 시간을 주는 것입니다. 또한 이 20분의 시간이 주어지기 때문에 인스턴스가 여러 관리 작업을 연속으로 거치는 경우 인스턴스가 여러 번 일시 중지와 재개를 반복하지 않아도 됩니다.

Aurora Serverless v2와 Aurora Serverless v1의 자동 일시 중지 동작 차이

- 재개 소요 시간은 Aurora Serverless v1에 비해 Aurora Serverless v2에서 개선되었습니다. 인스턴스가 일시 중지된 기간이 24시간 미만이면 재개 소요 시간은 일반적으로 약 15초입니다. 인스턴스가 일시 중지된 기간이 24시간을 초과하면 재개 소요 시간이 더 길어질 수 있습니다.
- Aurora Serverless v2가 다중 AZ 클러스터에 적용되는 방식으로 인해 클러스터의 일부 DB 인스턴스가 일시 중지되고 다른 DB 인스턴스는 활성 상태일 수 있습니다. 라이터 인스턴스는 클러스터 내에서 특정 활동을 조정하는 데 필요하므로 실행 중인 리더 인스턴스가 있을 때마다 재개됩니다. Aurora Serverless v1은 리더 인스턴스를 사용하지 않으므로 전체 클러스터가 항상 일시 중지 상태이거나 활성 상태입니다.

- 리더 엔드포인트가 연결할 리더 인스턴스를 무작위로 선택하면 해당 리더 인스턴스가 이미 활성 상태이거나 자동 일시 중지되었을 수 있습니다. 따라서 리더 인스턴스에 액세스하는 데 소요되는 시간은 다양하고 예측하기 어려울 수 있습니다. 따라서 Aurora Serverless v2 및 자동 일시 중지를 사용하는 다중 AZ 클러스터의 경우 모든 읽기 전용 세션을 리더 엔드포인트로 전달하는 대신 특정 읽기 전용 사용 사례에 대한 사용자 지정 엔드포인트를 설정하는 것이 도움이 될 수 있습니다.
- Aurora Serverless v2 인스턴스는 프로비저닝된 인스턴스와 동일한 빈도로 유지 관리 작업을 거칩니다. Aurora는 이러한 유지 관리가 필요할 때 인스턴스를 자동으로 재개하므로 Aurora Serverless v1 클러스터보다 Aurora Serverless v2 인스턴스가 더 자주 재개되는 것을 볼 수 있습니다.

다양한 유형의 Aurora 클러스터에서 Aurora Serverless v2 자동 일시 중지의 작동 방식

자동 일시 중지 기능에 대한 고려 사항은 Aurora 클러스터에 있는 인스턴스 수, 리더 인스턴스의 장애 조치 승격 계층, 모든 인스턴스가 Aurora Serverless v2인지 또는 Aurora Serverless v2와 프로비저닝의 조합인지에 따라 달라집니다.

주제

자동 일시 중지 사용 시 권장되는 Aurora 클러스터 레이아웃

자동 일시 중지 기능이 활성화되면 사용 사례에 맞게 고가용성, 빠른 응답 및 확장성의 적절한 균형을 맞추기 위해 Aurora 클러스터를 조정할 수 있습니다. 이렇게 하려면 클러스터의 DB 인스턴스에 대해 Aurora Serverless v2 인스턴스, 프로비저닝된 인스턴스 및 장애 조치 승격 계층의 조합을 선택합니다.

다음 유형의 구성은 클러스터에 대한 고가용성과 비용 최적화 간의 절충점을 보여줍니다.

- 개발 및 테스트 시스템의 경우 Aurora Serverless v2 DB 인스턴스를 사용하여 단일 AZ DB 클러스터를 설정할 수 있습니다. 단일 인스턴스가 모든 읽기 및 쓰기 요청을 처리합니다. 클러스터를 상당한 시간 동안 사용하지 않으면 DB 인스턴스가 일시 중지됩니다. 이때 클러스터의 DB 컴퓨팅 요금 부과도 일시 중지됩니다.
- 고가용성이 우선순위이지만 클러스터에 여전히 완전히 유휴 상태인 기간이 있는 애플리케이션을 실행하는 시스템의 경우 라이터 및 리더 DB 인스턴스가 모두 Aurora Serverless v2인 다중 AZ 클러스

터를 설정할 수 있습니다. 리더 인스턴스를 장애 조치 우선순위 0 또는 1로 설정하여 라이터와 리더 인스턴스가 동시에 일시 중지되고 재개되도록 합니다. 이제 클러스터가 활성화되어 있는 동안 빠른 장애 조치의 이점을 누릴 수 있습니다. 클러스터가 자동 일시 중지 임곗값보다 오래 유휴 상태로 유지되면 두 인스턴스 모두에 대한 DB 인스턴스 요금 부과가 일시 중지됩니다. 클러스터가 처리를 재개하면 첫 번째 데이터베이스 세션이 연결하는 데 잠시 시간이 걸립니다.

- 클러스터가 최소한의 활동으로 지속적으로 활성 상태이고 모든 연결에 대해 빠른 응답이 필요하다고 가정해 보겠습니다. 이 경우 둘 이상의 Aurora Serverless v2 리더 인스턴스로 클러스터를 만들고 일부 리더 인스턴스의 용량을 라이터에서 분리할 수 있습니다. 라이터 인스턴스와 리더 인스턴스 하나에 대해 장애 조치 우선순위 0 또는 1을 지정합니다. 다른 리더 인스턴스에 대해 1보다 큰 우선순위를 지정합니다. 이렇게 하면 라이터와 리더 중 하나가 활성 상태를 유지하는 동안에도 우선순위가 높은 계층의 리더 인스턴스가 자동으로 일시 중지될 수 있습니다.

  이 경우 몇 가지 다른 기법을 사용하여 유휴 기간 동안 여전히 낮은 용량으로 스케일 다운하면서 클러스터를 지속적으로 사용 가능하도록 할 수 있습니다.

  - 라이터 및 우선순위가 0 또는 1인 리더에 프로비저닝된 인스턴스를 사용할 수 있습니다. 이렇게 하면 두 DB 인스턴스는 절대 자동 일시 중지되지 않으며 데이터베이스 트래픽을 처리하고 장애 조치를 수행하는 데 항상 사용 가능합니다.
  - 더 높은 우선순위 계층의 Aurora Serverless v2 인스턴스가 포함되지만 라이터 또는 승격 계층이 0 또는 1인 리더는 포함되지 않는 사용자 지정 엔드포인트를 설정할 수 있습니다. 이렇게 하면 지연 시간에 민감하지 않은 읽기 전용 세션을 자동 일시 중지될 수 있는 리더에게 전달할 수 있습니다. Aurora가 항상 깨어 있는 리더 인스턴스 또는 자동 일시 중지된 인스턴스 중 하나로 리더 엔드포인트 연결을 전달할 수 있으므로 이러한 요청에 리더 엔드포인트를 사용하지 않아도 됩니다. 사용자 지정 엔드포인트를 사용하면 빠른 응답 또는 추가 크기 조정 용량에 대한 기본 설정에 따라 다양한 인스턴스 그룹으로 연결을 전달할 수 있습니다.

DB 클러스터의 라이터 인스턴스에 대한 Aurora Serverless v2 자동 일시 중지의 작동 방식

Aurora DB 클러스터에 단일 DB 인스턴스만 포함된 경우 DB 인스턴스를 자동으로 일시 중지하고 재개하는 메커니즘은 간단합니다. 이 메커니즘은 라이터 인스턴스의 활동에만 달려 있습니다. 개발 및 테스트에 사용되는 클러스터 또는 고가용성이 중요하지 않은 애플리케이션 실행에 대해 이러한 구성이 있을 수 있습니다. 단일 인스턴스 클러스터에서 Aurora는 리더 엔드포인트를 통해 라이터 DB 인스턴스로 연결을 전달합니다. 따라서 단일 인스턴스 DB 클러스터의 경우 리더 엔드포인트에 연결을 시도하면 자동 일시 중지된 라이터 DB 인스턴스가 재개됩니다.

DB 인스턴스가 여러 개인 Aurora 클러스터에는 다음과 같은 추가 요소가 적용됩니다.

- Aurora DB 클러스터 내에서 라이터 DB 인스턴스는 일반적으로 자주 액세스됩니다. 따라서 하나 이상의 리더 DB 인스턴스가 자동으로 일시 중지된 동안에도 라이터 DB 인스턴스가 활성 상태로 유지될 수 있습니다.
- 리더 DB 인스턴스의 특정 활동을 수행하려면 라이터 DB 인스턴스를 사용할 수 있어야 합니다. 따라서 모든 리더 인스턴스가 일시 중지될 때까지 라이터 DB 인스턴스를 일시 중지할 수 없습니다. 리더 인스턴스를 재개하면 애플리케이션이 라이터 인스턴스에 직접 액세스하지 않더라도 라이터 인스턴스가 자동으로 재개됩니다.
- 장애 조치 승격 계층이 0 및 1인 Aurora Serverless v2 리더 인스턴스는 용량이 라이터 인스턴스와 동기화되도록 크기가 조정됩니다. 따라서 Aurora Serverless v2 라이터 인스턴스가 재개되면 승격 계층이 0 또는 1인 Aurora Serverless v2 리더 인스턴스도 재개됩니다.

다중 AZ 클러스터에서 Aurora Serverless v2 자동 일시 중지의 작동 방식

라이터와 하나 이상의 리더 DB 인스턴스가 모두 포함된 Aurora DB 클러스터 내에서 일부 Aurora Serverless v2 DB 인스턴스는 다른 DB 인스턴스가 활성 상태인 동안 일시 중지될 수 있습니다. 라이터 인스턴스와 장애 조치 우선순위가 0 및 1인 리더 인스턴스는 항상 모두 동시에 일시 중지되고 재개됩니다. 우선순위가 0 또는 1이 아닌 리더 인스턴스는 다른 인스턴스와 별개로 일시 중지되고 재개될 수 있습니다.

이 기능을 여러 리더 인스턴스가 있는 클러스터에 사용하면 고가용성을 유지하면서 비용을 관리할 수 있습니다. 라이터 인스턴스와 또 다른 1~2개의 리더 인스턴스는 항상 활성 상태를 유지할 수 있으며, 그 외의 리더 인스턴스는 대용량 읽기 트래픽을 처리하는 데 필요하지 않은 경우 일시 중지될 수 있습니다.

리더 인스턴스가 자동으로 일시 중지될 수 있는지는 용량이 독립적으로 크기 조정될 수 있는지 또는 용량이 라이터 DB 인스턴스의 용량과 연결되어 있는지에 따라 달라집니다. 이 크기 조정 속성은 리더 DB 인스턴스의 장애 조치 우선순위에 따라 달라집니다. 리더의 우선순위가 0 또는 1인 경우 리더의 용량은 라이터 DB 인스턴스의 용량을 추적합니다. 따라서 가장 광범위한 상황에서 리더 DB 인스턴스가 자동으로 일시 중지되도록 하려면 우선순위를 1보다 높은 값으로 설정하세요.

리더 인스턴스를 재개하는 시간이 라이터 인스턴스를 재개하는 시간보다 약간 더 걸릴 수 있습니다. 인스턴스가 일시 중지될 수 있는 경우 가장 빠른 응답을 위해 클러스터 엔드포인트에 연결하세요.

프로비저닝된 인스턴스가 있는 클러스터에서 Aurora Serverless v2 자동 일시 중지의 작동 방식

Aurora DB 클러스터에 프로비저닝된 DB 인스턴스는 자동으로 일시 중지되지 않습니다. db.serverless 인스턴스 클래스를 사용하는 Aurora Serverless v2 DB 인스턴스만 자동 일시 중지 기능을 사용할 수 있습니다.

Aurora 클러스터에 프로비저닝된 DB 인스턴스가 포함된 경우 Aurora Serverless v2 라이터 인스턴스는 자동으로 일시 중지되지 않습니다. 이는 리더 인스턴스가 활성 상태일 때 라이터 인스턴스를 계속 사용할 수 있어야 한다는 요구 사항 때문입니다. Aurora Serverless v2 라이터가 활성 상태로 유지된다는 것은 장애 조치 우선순위가 0 및 1인 모든 Aurora Serverless v2 리더 인스턴스가 프로비저닝된 인스턴스가 포함된 하이브리드 클러스터에서 자동 일시 중지되지 않는다는 의미이기도 합니다.

자동 일시 중지를 사용하는 Aurora 클러스터 모니터링

Aurora를 모니터링하려면 Amazon CloudWatch로 Amazon Aurora 지표 모니터링의 모니터링 프로시저와 Amazon Aurora용 지표 참조에 나열된 CloudWatch 지표에 이미 익숙한 상태여야 합니다. 자동 일시 중지 기능을 사용하는 Aurora 클러스터를 모니터링할 때 몇 가지 특별한 고려 사항이 있습니다.

- 인스턴스가 일시 중지되어 있어서 Aurora Serverless v2 인스턴스가 로그 데이터와 대부분의 지표를 기록하지 않는 기간이 있을 수 있습니다. 인스턴스가 일시 중지된 동안 CloudWatch로 전송되는 유일한 지표는 `CPUUtilization` 및 `ACUUtilization`의 경우 0%, `ServerlessDatabaseCapacity`의 경우 0입니다.
- Aurora Serverless v2 인스턴스가 예상보다 더 높은 빈도로 중지되는지 아니면 더 낮은 빈도로 일시 중지되는지 확인할 수 있습니다. 그러려면 `ServerlessDatabaseCapacity` 지표가 0이 아닌 값에서 0으로 변경되는 빈도와 0인 상태로 유지되는 기간을 확인하세요. 인스턴스가 예상한 기간 만큼 일시 중지 상태로 유지되지 않으면 비용을 최대한 절감할 수 없습니다. 인스턴스가 의도한 것보다 더 자주 일시 중지되었다가 재개되면 연결 요청에 응답할 때 클러스터에 불필요한 지연 시간이 발생할 수 있습니다. Aurora Serverless v2 인스턴스 일시 중지 가능 여부와 빈도에 영향을 미치는 요인에 대한 자세한 내용은 Aurora Serverless v2 자동 일시 중지 기능에 대한 사전 조건 및 제한 사항, Aurora Serverless v2가 자동 일시 중지되지 않는 상황 및 Aurora Serverless v2 자동 일시 중지 문제 해결 섹션을 참조하세요.
- Aurora Serverless v2 인스턴스에 대한 자동 일시 중지 및 재개 작업을 기록하는 로그 파일도 검사할 수 있습니다. 제한 시간 간격이 만료된 후 인스턴스가 일시 중지되지 않은 경우 이 로그 파일에 자동 일시 중지가 발생하지 않은 이유도 포함됩니다. 자세한 내용은 ??? 단원을 참조하십시오.

주제

Aurora Serverless v2 인스턴스가 일시 중지되었는지 확인

Aurora Serverless v2 인스턴스가 일시 중지 상태인지 확인하려면 인스턴스에 대한 `ACUUtilization` 지표를 관찰하면 됩니다. 인스턴스가 일시 중지된 동안 해당 지표의 값은 0입니다.

Aurora Serverless v2 인스턴스가 일시 중지된 동안 상태 값은 여전히 사용 가능으로 표시됩니다. 일시 중지된 Aurora Serverless v2 인스턴스가 재개되는 동안에도 마찬가지입니다. 이는 연결이 약간 지연되더라도 이러한 인스턴스에 성공적으로 연결할 수 있기 때문입니다.

Aurora 인스턴스의 가용성과 관련된 모든 지표는 인스턴스가 일시 중지된 기간을 인스턴스가 사용 가능한 시간으로 간주합니다.

자동 일시 중지 및 자동 재개 작업을 위한 이벤트

Aurora는 자동 일시 중지 및 자동 재개 작업이 시작, 완료 또는 취소될 때 Aurora Serverless v2 인스턴스에 대한 이벤트를 내보냅니다. 자동 일시 중지 기능과 관련된 이벤트는 `RDS-EVENT-0370`부터 `RDS-EVENT-0374`입니다. 이러한 이벤트에 대한 자세한 내용은 DB 인스턴스 이벤트 섹션을 참조하세요.

성능 개선 도우미 및 향상된 모니터링을 사용할 경우 자동 일시 중지의 작동 방식

Aurora Serverless v2 인스턴스가 일시 중지된 동안 Aurora는 성능 개선 도우미 또는 향상된 모니터링을 통해 해당 인스턴스에 대한 모니터링 정보를 수집하지 않습니다. 인스턴스가 재개되면 Aurora가 이러한 모니터링 정보 수집을 재개하기 전에 잠시 지연이 있을 수 있습니다.

Aurora Serverless v2 자동 일시 중지 기능이 Aurora 지표와 상호 작용하는 방법

Aurora Serverless v2 인스턴스가 일시 중지된 동안에는 대부분의 CloudWatch 지표를 내보내지 않고 데이터베이스 로그에 정보를 기록하지 않습니다. 인스턴스가 일시 중지된 동안 CloudWatch로 전송되는 유일한 지표는 `CPUUtilization` 및 `ACUUtilization`의 경우 0%, `ServerlessDatabaseCapacity`의 경우 0입니다.

CloudWatch가 인스턴스 또는 클러스터 가용성 및 가동 시간과 관련된 통계를 계산하는 경우 Aurora Serverless v2 인스턴스는 일시 중지된 시간 동안 사용 가능한 것으로 간주됩니다.

일시 중지된 Aurora Serverless v2 인스턴스의 로그를 설명하거나 다운로드하기 위해 AWS CLI 또는 RDS API 작업을 시작하면 인스턴스가 자동으로 재개되어 로그 정보를 사용 가능하게 합니다.

CloudWatch 지표의 예

다음 AWS CLI 예시에서는 시간이 지남에 따라 변화하는 인스턴스의 용량을 관찰하는 방법을 보여줍니다. 이 기간 동안 인스턴스는 자동으로 일시 중지되었다가 재개됩니다. 일시 중지된 동안 `ServerlessDatabaseCapacity` 지표는 0의 값을 보고합니다. 해당 기간의 특정 시점에 인스턴스가 일시 중지된 상태였는지 확인하기 위해 해당 기간 동안의 최소 용량이 0이었는지 확인합니다.

다음 Linux 예시는 일정 시간 동안 자동으로 일시 중지된 Aurora Serverless v2 인스턴스를 나타냅니다. 3분에 걸쳐 1분마다 `ServerlessDatabaseCapacity`를 샘플링합니다. 최소 ACU 값인 0.0은 인스턴스가 매분 특정 시점에 일시 중지 상태였음을 알려줍니다.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '3 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:11:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:12:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
```

다음으로 일시 중지된 Aurora Serverless v2 인스턴스에 연결하려고 시도합니다. 이 예시에서는 연결 시도가 성공하지 않도록 의도적으로 잘못된 암호를 사용합니다. 실패하더라도 연결 시도로 인해 Aurora가 일시 중지된 인스턴스를 재개합니다.

```
$ mysql -h my_cluster_endpoint.rds.amazonaws.com -u admin -pwrong-password

ERROR 1045 (28000): Access denied for user 'admin'@'ip_address' (using password: YES)
```

다음 Linux 예시는 일시 중지된 인스턴스가 재개되고 약 5분 동안 유휴 상태로 유지된 다음 일시 중지 상태로 돌아간 것을 보여줍니다. 인스턴스가 2.0ACU의 용량에서 재개되었습니다. 그런 다음 내부 정리를 수행하는 등의 작업을 위해 약간 스케일 업되었습니다. 5분 제한 시간 내에 사용자 연결 시도를 수신하지 못했기 때문에 4.0ACU에서 곧바로 일시 중지 상태로 전환되었습니다.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '8 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
DATAPOINTS 2.0 2024-08-02T22:14:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:15:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:16:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:17:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:18:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:19:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:20:00+00:00 None
```

보고서가 인스턴스가 워크로드를 처리하기 위해 얼마나 빨리 스케일 업되었는지 보여주기 위한 것이라면 통계에 `Minimum` 대신 `Maximum`을 지정할 수 있습니다. 용량 계획 및 비용 추정을 위해 더 긴 기간을 지정하고 `Average` 통계를 사용할 수 있습니다. 이렇게 하면 활동이 많은 기간, 활동이 적은 기간, 일시 중지 상태인 기간 동안의 일반적인 용량 값을 확인할 수 있습니다. 일시 중지 및 재개와 관련된 정확한 시간 동안의 동작을 조사하기 위해 1초의 기간을 지정하고 더 짧은 시간 간격을 살펴볼 수 있습니다. `2024-08-02T22:13:00+00:00`과 같은 출력의 타임스탬프 값은 `--start-time` 및 `--end-time` 옵션에 대한 정확한 파라미터를 지정하는 형식을 보여줍니다.

Aurora Serverless v2 자동 일시 중지 문제 해결

Aurora Serverless v2 인스턴스가 예상만큼 자주 일시 중지되지 않는 경우 다음과 같은 가능한 원인을 확인하세요.

- 실행 중인 Aurora 버전이 최소 용량으로 0ACU를 지원하는지 확인합니다. Aurora 버전별 용량 범위는 Aurora Serverless v2 용량 섹션을 참조하세요.
- 클러스터의 최소 용량 값이 0ACU로 설정되어 있는지 확인합니다.
- 해당 인스턴스가 프로비저닝된 인스턴스 클래스 중 하나가 아닌 Aurora Serverless v2 인스턴스 클래스 `db.serverless`를 실제로 사용하고 있는지 확인합니다.
- 클러스터가 Aurora Serverless v2 자동 일시 중지 기능에 대한 사전 조건 및 제한 사항의 호환되지 않는 기능 또는 설정을 사용하지 않는지 확인합니다.

- Aurora Serverless v2 인스턴스가 일시 중지 또는 재개된 시점 또는 Aurora가 어떤 이유로 인스턴스를 일시 중지 또는 재개할 수 없었던 시점을 보여주는 로그 파일을 검사합니다. 자세한 내용은 ??? 단원을 참조하십시오.
- 클라이언트 또는 애플리케이션이 오랜 기간 동안 연결을 열린 상태로 유지하고 있는지 확인합니다. 반대로 RDS 데이터 API 또는 Lambda 함수를 사용하는 애플리케이션이 빈번한 요청을 보내고 있어 인스턴스가 일시 중지할 만큼 오랫동안 유휴 상태가 발생하지 않는지 확인합니다. `ConnectionAttempts` 및 `DatabaseConnections`과 같은 CloudWatch 지표를 검사할 수 있습니다. 자세한 내용은 Amazon Aurora에 대한 인스턴스 수준 지표 단원을 참조하십시오.
- 리더 인스턴스가 거의 일시 중지되지 않는 경우 장애 조치 우선순위를 확인합니다. 리더를 장애 조치 시 대기가 아닌 읽기 크기 조정에 사용하는 경우 2~15 범위의 우선순위로 설정합니다.
- 라이터 인스턴스가 거의 일시 중지되지 않는 경우 리더 인스턴스의 사용량을 확인합니다. 라이터는 전체 클러스터가 유휴 상태인 경우에만 일시 중지할 수 있습니다. 리더 인스턴스에 연결하면 라이터가 재개되기 때문입니다.
- Aurora Serverless v2 인스턴스가 재개되는 동안 애플리케이션이 연결 요청 중에 제한 시간이 초과되면 애플리케이션 코드 또는 기본 데이터베이스 프레임워크에서 사용하는 제한 시간 간격을 늘리는 것을 고려해 보세요. 연결 제한 시간이 길어지면 Aurora Serverless v2 인스턴스가 재개되는 동안 연결이 실패할 가능성이 줄어듭니다. 그러나 제한 시간이 길수록 애플리케이션이 클러스터의 가용성 문제를 감지하는 속도가 느려질 수도 있습니다.

  반대로 애플리케이션이 일시 중지된 인스턴스에 자주 노출되지 않도록 자동 일시 중지 간격을 늘리는 것을 고려해 보세요.

  자동 일시 중지 동작과 애플리케이션의 클러스터 응답성 간에 논리적 균형이 없는 경우 해당 클러스터는 자동 일시 중지 기능을 사용하기에 적합하지 않을 수 있습니다.
- Aurora Serverless v2 인스턴스가 일시 중지되는 기간을 추정하는 경우 정확한 예측을 수행하는 것을 어렵게 하는 요인이 있다는 점에 유의하세요.
  - 인스턴스는 유지 관리나 마이너 버전 업그레이드를 수행하거나 파라미터 그룹에 변경 사항을 적용하기 위해 주기적으로 재개될 수 있습니다.
  - 다중 AZ 클러스터의 경우 인스턴스 하나를 재개하면 다른 인스턴스도 재개되는 경우가 있습니다. 리더를 재개하면 항상 라이터가 재개됩니다. 라이터를 재개하면 항상 장애 조치 우선순위가 0 및 1인 리더가 재개됩니다.

  현실적인 워크로드를 사용하여 며칠 동안 실제 일시 중지된 시간을 측정하는 것이 좋습니다. 그런 다음 이러한 측정값을 사용하여 인스턴스가 일시 중지될 것으로 예상할 수 있는 시간 비율에 대한 기준을 설정합니다.

- MySQL 제거, MySQL 이벤트 스케줄러, PostgreSQL autovacuum 또는 pg_cron 확장을 통해 예약된 PostgreSQL 작업과 같은 내부 작업이 실행되지 않거나 완료되지 않을 수 있습니다. 인스턴스는 사용자가 데이터베이스에 연결하지 않는 이러한 작업을 수행하기 위해 자동으로 재개되지 않습니다. 자동 일시 중지 제한 시간이 만료될 때 이러한 내부 작업이 진행 중인 경우 MySQL 제거 및 PostgreSQL autovacuum과 같은 유지 관리 작업이 취소됩니다. MySQL 이벤트 스케줄러 또는 PostgreSQL pg_cron 확장에서 예약된 작업도 Aurora가 일시 중지 작업을 시작할 때 진행 중인 경우 취소됩니다.

  예약된 작업을 수행하기 위해 인스턴스가 주기적으로 깨어 있는지 확인해야 하는 경우 작업 시작 시간 전에 인스턴스를 재개하기 위한 연결을 시작할 수 있습니다. 사용자 활동이 완료된 후 autovacuum과 같은 작업이 더 오래 실행되도록 자동 일시 중지 제한 시간 간격을 늘릴 수도 있습니다. Lambda 함수와 같은 메커니즘을 사용하여 필요에 따라 인스턴스를 자동으로 재개하도록 일정에 따라 데이터베이스 작업을 수행할 수도 있습니다.

Aurora Serverless v2 자동 일시 중지 기능을 위한 애플리케이션 설계 고려 사항

Aurora Serverless v2 DB 인스턴스가 자동으로 일시 중지된 후 재개되면 비교적 작은 용량으로 시작하고 거기서부터 스케일 업됩니다. 이 시작 용량은 DB 인스턴스가 자동으로 일시 중지되기 직전에 용량이 약간 더 높았더라도 적용됩니다.

연결을 설정하는 동안 약 15초의 간격을 허용할 수 있는 애플리케이션에서 이 기능을 사용하세요. 이는 하나 또는 소수의 수신 연결로 인해 Aurora Serverless v2 인스턴스가 재개되는 일반적인 경우를 감안한 것입니다. 인스턴스가 일시 중지된 기간이 24시간을 초과하면 재개 소요 시간이 더 길어질 수 있습니다.

애플리케이션이 이미 Aurora Serverless v1 및 자동 일시 중지 기능을 사용하고 있는 경우 연결 시도에 이러한 제한 시간 간격이 이미 있을 수 있습니다. Aurora 중지/시작 클러스터 기능과 함께 Aurora Serverless v2를 이미 사용 중인 경우 자동 일시 중지된 Aurora Serverless v2 인스턴스의 재개 소요 시간은 일반적으로 중지된 클러스터를 시작하는 시간보다 훨씬 짧습니다.

애플리케이션에서 연결 로직을 코딩할 때 첫 번째 시도에서 일시적인 원인이 있는 오류가 반환되면 연결을 다시 시도하세요. (인증 실패로 인한 오류인 경우 다시 시도하기 전에 자격 증명을 수정하세요.) 재개 직후 발생하는 오류는 제한 시간 또는 데이터베이스 한도와 관련된 오류일 수 있습니다. 재시도는 동시 연결 요청 수가 많아서 Aurora Serverless v2 인스턴스가 재개되는 드문 경우에 문제를 처리할 수 있습니다. 이 경우 일부 연결이 처리되는 데 평소보다 오래 걸리거나 첫 번째 시도에서 동시 연결 한도를 초과할 수 있습니다.

애플리케이션 개발 및 디버깅 중에 데이터베이스에 대한 연결이 열려 있는 클라이언트 세션 또는 프로그래밍 도구를 그대로 두지 마세요. Aurora는 연결이 SQL 문 또는 트랜잭션을 실행하고 있지 않더라도 사용자가 시작한 연결이 열려 있는 경우 인스턴스를 일시 중지하지 않습니다. Aurora 클러스터의 한 Aurora Serverless v2 인스턴스를 일시 중지할 수 없는 경우 클러스터의 다른 인스턴스도 일시 중지되지 않을 수 있습니다. 자세한 내용은 Aurora Serverless v2가 자동 일시 중지되지 않는 상황 단원을 참조하십시오.

Aurora는 Aurora Serverless v2 DB 인스턴스가 재개를 시작하고, 재개를 완료하고, 어떤 이유로든 인스턴스를 재개할 수 없을 때 이벤트를 내보냅니다. 이러한 이벤트에 대한 자세한 내용은 DB 인스턴스 이벤트 섹션을 참조하세요.

Aurora Serverless v2로 마이그레이션

Aurora Serverless v2를 사용하도록 기존 DB 클러스터를 변환하려면 다음을 수행합니다.

- 프로비저닝된 Aurora DB 클러스터에서 업그레이드합니다.
- Aurora Serverless v1 클러스터에서 업그레이드합니다.
- 온프레미스 데이터베이스에서 Aurora Serverless v2 클러스터로 마이그레이션합니다.

업그레이드된 클러스터가 Aurora Serverless v2 요구 사항 및 제한 사항에 나열된 적절한 엔진 버전을 실행 중이면 여기에 Aurora Serverless v2 DB 인스턴스를 추가할 수 있습니다. 업그레이드된 클러스터에 추가하는 첫 번째 DB 인스턴스는 프로비저닝된 DB 인스턴스여야 합니다. 그런 다음 쓰기 워크로드, 읽기 워크로드 또는 둘 다에 대한 처리를 Aurora Serverless v2 DB 인스턴스로 전환할 수 있습니다.

목차

  - Aurora MySQL 호환 DB 클러스터
  - Aurora PostgreSQL 호환 DB 클러스터
- Aurora Serverless v2로 온프레미스 데이터베이스 마이그레이션

> **Note**
> 이 주제에서는 기존 DB 클러스터를 변환하는 방법을 설명합니다. 새로운 Aurora Serverless v2 DB 클러스터 생성에 대한 자세한 내용은 Aurora Serverless v2를 사용하는 DB 클러스터 생성 섹션을 참조하세요.

Aurora Serverless v2를 사용하도록 기존 클러스터 업그레이드 또는 전환

프로비저닝된 클러스터에 Aurora Serverless v2를 지원하는 엔진 버전이 있는 경우 Aurora Serverless v2로 전환할 때 업그레이드가 필요하지 않습니다. 이 경우 Aurora Serverless v2 DB 인스턴스를 원래 클러스터에 추가할 수 있습니다. 모든 Aurora Serverless v2 DB 인스턴스를 사용하도록 클러스터를 전환할 수 있습니다. 동일한 DB 클러스터에서 Aurora Serverless v2와 프로비저닝된 DB 인스턴스를 조합하여 사용할 수도 있습니다. Aurora Serverless v2을 지원하는 Aurora 엔진 버전은 Aurora Serverless v2를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

Aurora Serverless v2를 지원하지 않는 낮은 엔진 버전을 실행 중인 경우 다음과 같은 일반적인 단계를 수행합니다.

1. 클러스터를 업그레이드합니다.
2. 업그레이드된 클러스터에 대해 프로비저닝된 라이터 DB 인스턴스를 생성합니다.
3. Aurora Serverless v2 DB 인스턴스를 사용하도록 클러스터를 수정합니다.

> **Important**
> 스냅샷 복원 또는 복제를 사용하여 Aurora Serverless v2 호환 버전으로 메이저 버전 업그레이드를 수행하는 경우 새 클러스터에 추가하는 첫 번째 DB 인스턴스는 프로비저닝된 DB 인스턴스여야 합니다. 이 추가는 업그레이드 프로세스의 마지막 단계를 시작합니다.
> 최종 단계가 완료될 때까지 클러스터에는 Aurora Serverless v2 지원에 필요한 인프라가 없습니다. 따라서 이러한 업그레이드된 클러스터는 항상 프로비저닝된 라이터 DB 인스턴스로 시작

합니다. 그런 다음 프로비저닝된 DB 인스턴스를 Aurora Serverless v2 인스턴스로 변환하거나 장애 조치를 수행할 수 있습니다.

Aurora Serverless v1에서 Aurora Serverless v2로 업그레이드하려면 프로비저닝된 클러스터를 중간 단계로 생성해야 합니다. 그런 다음 프로비저닝된 클러스터로 시작할 때와 동일한 업그레이드 단계를 수행합니다.

Aurora Serverless v2를 사용하기 위한 MySQL 호환 클러스터의 업그레이드 경로

원래 클러스터에서 Aurora MySQL을 실행 중인 경우 클러스터의 엔진 버전 및 엔진 모드에 따라 적절한 프로시저를 선택합니다.

원래 Aurora MySQL 클러스터가 다음과 같은 경우	Aurora Serverless v2로 전환하려면 이렇게 하세요.
MySQL 8.0과 호환되는 Aurora MySQL 버전 3을 실행하는 프로비저닝된 클러스터	이는 기존 Aurora MySQL 클러스터에서 모든 변환을 위한 마지막 단계입니다. 필요한 경우 버전 3.02.0 이상으로 마이너 버전 업그레이드를 수행합니다. 라이터 DB 인스턴스에 프로비저닝된 DB 인스턴스를 사용합니다. 하나의 Aurora Serverless v2 리더 DB 인스턴스를 추가합니다. 라이터 DB 인스턴스를 만들기 위해 장애 조치를 수행합니다. (선택 사항) 클러스터의 다른 프로비저닝된 DB 인스턴스를 Aurora Serverless v2로 변환합니다. 또는 새 Aurora Serverless v2 DB 인스턴스를 추가하고 프로비저닝된 DB 인스턴스를 제거합니다. 전체 프로시저 및 예는 프로비저닝된 클러스터에서 Aurora Serverless v2로 전환 섹션을 참조하세요.
MySQL 5.7과 호환되는 Aurora MySQL 버전 2를 실행하는 프로비저닝된 클러스터	Aurora MySQL 버전 3.02.0 이상으로 메이저 버전 업그레이드를 수행합니다. 그런 다음 Aurora MySQL 버전 3에 대한 프로시저에 따라 Aurora

원래 Aurora MySQL 클러스터가 다음과 같은 경우	Aurora Serverless v2로 전환하려면 이렇게 하세요.
	Serverless v2 DB 인스턴스를 사용하도록 클러스터를 전환합니다.
MySQL 5.7과 호환되는 Aurora MySQL 버전 2를 실행하는 Aurora Serverless v1 클러스터	Aurora Serverless v1에서 전환을 계획하는 데 도움을 받으려면 먼저 Aurora Serverless v2 및 Aurora Serverless v1 비교에 문의하세요. 그런 다음 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드의 절차를 따릅니다.

Aurora Serverless v2를 사용하기 위한 PostgreSQL 호환 클러스터의 업그레이드 경로

원래 클러스터에서 Aurora PostgreSQL을 실행 중인 경우 클러스터의 엔진 버전 및 엔진 모드에 따라 적절한 프로시저를 선택합니다.

원래 Aurora PostgreSQL 클러스터가 다음과 같은 경우	Aurora Serverless v2로 전환하려면 이렇게 하세요.
Aurora PostgreSQL 버전 13을 실행하는 프로비저닝된 클러스터	이는 기존 PostgreSQL 클러스터에서 모든 변환을 위한 마지막 단계입니다. 필요한 경우 버전 13.6 이상으로 마이너 버전 업그레이드를 수행합니다. 라이터 DB 인스턴스에 대해 프로비저닝된 DB 인스턴스를 하나 추가합니다. 하나의 Aurora Serverless v2 리더 DB 인스턴스를 추가합니다. 해당 Aurora Serverless v2 인스턴스를 라이터 DB 인스턴스로 만들기 위해 장애 조치를 수행합니다. (선택 사항) 클러스터의 다른 프로비저닝된 DB 인스턴스를 Aurora Serverless v2로 변환합니다. 또는 새 Aurora Serverless v2 DB 인스턴스를 추가하고 프로비저닝된 DB 인스턴스를 제거합니다.

원래 Aurora PostgreSQL 클러스터가 다음과 같은 경우	Aurora Serverless v2로 전환하려면 이렇게 하세요.
	전체 프로시저 및 예는 프로비저닝된 클러스터에서 Aurora Serverless v2로 전환 섹션을 참조하세요.
Aurora PostgreSQL 버전 11 또는 12를 실행하는 프로비저닝된 클러스터	Aurora PostgreSQL 버전 13.6 이상으로 메이저 버전 업그레이드를 수행합니다. 그런 다음 Aurora PostgreSQL 버전 13에 대한 절차에 따라 Aurora Serverless v2 DB 인스턴스를 사용하도록 클러스터를 전환합니다.
Aurora PostgreSQL 버전 11 또는 13을 실행하는 Aurora Serverless v1 클러스터	Aurora Serverless v1에서 전환을 계획하는 데 도움을 받으려면 먼저 Aurora Serverless v2 및 Aurora Serverless v1 비교에 문의하세요. 그런 다음 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드의 절차를 따릅니다.

프로비저닝된 클러스터에서 Aurora Serverless v2로 전환

Aurora Serverless v2를 사용하도록 프로비저닝된 클러스터를 전환하려면 다음 단계를 따르세요.

1. 프로비저닝된 클러스터를 Aurora Serverless v2 DB 인스턴스와 함께 사용하기 위해 업그레이드해야 하는지 확인하세요. Aurora Serverless v2와 호환되는 Aurora 버전은 Aurora Serverless v2 요구 사항 및 제한 사항 섹션을 참조하세요.

   프로비저닝된 클러스터가 Aurora Serverless v2에 사용할 수 없는 엔진 버전을 실행 중인 경우 클러스터의 엔진 버전을 업그레이드하세요.

   - MySQL 5.7 호환 프로비저닝된 클러스터가 있는 경우 Aurora MySQL 버전 3에 대한 업그레이드 지침을 따릅니다. 현재 위치 업그레이드 수행 방법의 프로시저를 따르세요.
   - PostgreSQL 버전 11~12를 실행하는 PostgreSQL 호환 프로비저닝된 클러스터가 있는 경우 Aurora PostgreSQL 버전 13에 대한 업그레이드 지침을 따르세요. 메이저 버전 업그레이드 수행의 프로시저를 따르세요.

2. Aurora Serverless v2 요구 사항 및 제한 사항의 Aurora Serverless v2 요구 사항과 일치하도록 다른 클러스터 속성을 구성합니다.

3. 클러스터에 대한 크기 조정 구성을 구성합니다. 클러스터의 Aurora Serverless v2 용량 설정의 프로시저를 따르세요.

4. 클러스터에 하나 이상의 Aurora Serverless v2 DB 인스턴스를 추가합니다. DB 클러스터에 Aurora 복제본 추가의 일반 절차를 따르세요. 각각의 새 DB 인스턴스에 대해 AWS Management Console에서 특수 DB 인스턴스 클래스 이름을 Serverless로 지정하거나 AWS CLI 또는 Amazon RDS API에서 `db.serverless`를 지정합니다.

   경우에 따라 클러스터에 이미 하나 이상의 프로비저닝된 리더 DB 인스턴스가 있을 수 있습니다. 그렇다면 새 DB 인스턴스를 생성하는 대신 리더 중 하나를 Aurora Serverless v2 DB 인스턴스로 변환할 수 있습니다. 이 작업을 수행하려면 프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환의 프로시저를 따르세요.

5. 장애 조치 작업을 수행하여 Aurora Serverless v2 DB 인스턴스 중 하나를 클러스터의 라이터 DB 인스턴스로 만듭니다.

6. (선택 사항) 프로비저닝된 DB 인스턴스를 Aurora Serverless v2로 변환하거나 클러스터에서 제거합니다. 프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환 또는 Aurora DB 클러스터에서 DB 인스턴스 삭제의 일반 절차를 따르세요.

> **Tip**
>
> 프로비저닝된 DB 인스턴스를 꼭 제거해야 하는 것은 아닙니다. Aurora Serverless v2 및 프로비저닝된 DB 인스턴스를 모두 포함하는 클러스터를 설정할 수 있습니다. 그러나 Aurora Serverless v2 DB 인스턴스의 성능 및 조정 특성에 익숙해질 때까지는 모두 동일한 유형의 DB 인스턴스로 클러스터를 구성하는 것이 좋습니다.

다음 AWS CLI 예제는 Aurora MySQL 버전 3.02.0을 실행하는 프로비저닝된 클러스터를 사용하는 전환 프로세스를 보여줍니다. 클러스트 이름은 `mysql-80`입니다. 클러스터는 라이터와 리더인 `provisioned-instance-1`과 `provisioned-instance-2`라는 두 개의 프로비저닝된 DB 인스턴스로 시작합니다. 둘 다 `db.r6g.large` DB 인스턴스 클래스를 사용합니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
```

```
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

일부 데이터가 포함된 테이블을 만들었습니다. 이렇게 하면 전환 전후에 클러스터의 데이터와 동작이 동일한지 확인할 수 있습니다.

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

먼저 클러스터에 용량 범위를 추가합니다. 그렇지 않으면 클러스터에 Aurora Serverless v2 DB 인스턴스를 추가할 때 오류가 발생합니다. 이 절차에 AWS Management Console을 사용하는 경우 첫 번째 Aurora Serverless v2 DB 인스턴스를 추가할 때 해당 단계는 자동입니다.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
```

```
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

원본 DB 인스턴스를 대신할 두 개의 Aurora Serverless v2 리더를 만듭니다. 이렇게 하려면 새로운 `db.serverless` DB 인스턴스에 DB 인스턴스 클래스를 지정합니다.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

장애 조치 작업을 수행하여 Aurora Serverless v2 DB 인스턴스 중 하나를 클러스터의 새 라이터로 만듭니다.

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
```

```
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

변경 사항이 적용되려면 몇 초가 걸립니다. 그 시점에서 Aurora Serverless v2 라이터와 Aurora Serverless v2 리더가 있습니다. 따라서 원래 프로비저닝된 DB 인스턴스가 필요하지 않습니다.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

전환 절차의 마지막 단계에서는 프로비저닝된 DB 인스턴스를 모두 삭제합니다.

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
```

```
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

최종 확인으로 Aurora Serverless v2 라이터 DB 인스턴스에서 원본 테이블에 액세스하고 쓸 수 있는지 확인합니다.

```
mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
+---------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
| And it finished with a Serverless v2 writer.      |
+---------------------------------------------------+
2 rows in set (0.01 sec)
```

또한 Aurora Serverless v2 리더 DB 인스턴스에 연결하여 새로 작성된 데이터도 사용할 수 있는지 확인합니다.

```
mysql> select * from serverless_v2_demo.demo;
+---------------------------------------------------+
| s                                                 |
+---------------------------------------------------+
| This cluster started with a provisioned writer.   |
| And it finished with a Serverless v2 writer.      |
+---------------------------------------------------+
2 rows in set (0.01 sec)
```

Aurora Serverless v2 및 Aurora Serverless v1 비교

이미 Aurora Serverless v1을 사용하고 있다면 Aurora Serverless v1과 Aurora Serverless v2의 주요 차이점을 배울 수 있습니다. 리더 DB 인스턴스 지원과 같은 아키텍처상의 차이점은 새로운 유형의 사용 사례를 보여줍니다.

다음 테이블에서 Aurora Serverless v2과 Aurora Serverless v1의 가장 중요한 차이점을 이해할 수 있습니다.

주제

Aurora Serverless v2 및 Aurora Serverless v1의 요구 사항 비교

다음 테이블에는 Aurora Serverless v2 또는 Aurora Serverless v1를 사용하여 데이터베이스를 실행하기 위한 다양한 요구 사항이 요약되어 있습니다. Aurora Serverless v2는 Aurora Serverless v1보다 더 높은 버전의 Aurora MySQL 및 Aurora PostgreSQL DB 엔진을 제공합니다.

특징	Aurora Serverless v2 요구 사항	Aurora Serverless v1 요구 사항
DB 엔진	Aurora MySQL, Aurora PostgreSQL	Aurora MySQL, Aurora PostgreSQL

특징	Aurora Serverless v2 요구 사항	Aurora Serverless v1 요구 사항
지원되는 Aurora MySQL 버전	Aurora MySQL을 사용하는 Aurora Serverless v2을(를) 참조하세요.	Aurora MySQL을 사용하는 Aurora Serverless v1을(를) 참조하세요.
지원되는 Aurora PostgreSQL 버전	Aurora PostgreSQL을 사용하는 Aurora Serverless v2을(를) 참조하세요.	Aurora PostgreSQL을 사용하는 Aurora Serverless v1을(를) 참조하세요.
DB 클러스터 업그레이드	프로비저닝된 DB 클러스터에서처럼, Aurora가 DB 클러스터를 업그레이드할 때까지 기다리지 않고 수동으로 업그레이드를 수행할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요. Note Aurora PostgreSQL 호환 DB 클러스터를 13.x에서 14.x 또는 15.x로 메이저 버전 업그레이드하려면 클러스터의 최대 용량이 2 ACU 이상이어야 합니다.	마이너 버전 업그레이드는 출시되는 즉시 자동으로 적용됩니다. 자세한 내용은 Aurora Serverless v1 및 Aurora 데이터베이스 엔진 버전 섹션을 참조하세요. 메이저 버전 업그레이드는 수동으로 수행할 수 있습니다. 자세한 내용은 Aurora Serverless v1 DB 클러스터 수정 섹션을 참조하세요.

특징	Aurora Serverless v2 요구 사항	Aurora Serverless v1 요구 사항
프로비저닝된 DB 클러스터에서 변환	다음 방법을 사용할 수 있습니다. • 기존 프로비저닝된 클러스터에 하나 이상의 Aurora Serverless v2 리더 DB 인스턴스를 추가합니다. 라이터에 Aurora Serverless v2를 사용하려면 Aurora Serverless v2 DB 인스턴스 중 하나로 장애 조치를 수행합니다. 전체 클러스터에서 Aurora Serverless v2 DB 인스턴스를 사용하려면 Aurora Serverless v2 DB 인스턴스를 라이터로 승격한 후 프로비저닝된 라이터 DB 인스턴스를 제거하세요. • 적절한 DB 엔진 및 엔진 버전으로 새 클러스터를 생성합니다. 표준 방법을 사용합니다. 예를 들어 클러스터 스냅샷을 복원하거나 기존 클러스터의 복제본을 생성합니다. 새 클러스터의 일부 또는 모든 DB 인스턴스에 대해 Aurora Serverless v2를 선택합니다. 복제를 통해 새 클러스터를 생성하는 경우 엔진 버전을 동시에 업그레이드할 수 없습니다. 원래 클러스터가 이미 Aurora Serverless v2와	프로비저닝된 클러스터의 스냅샷을 복원하여 새 Aurora Serverless v1 클러스터를 생성합니다.

특징	Aurora Serverless v2 요구 사항	Aurora Serverless v1 요구 사항
	호환되는 엔진 버전을 실행 중인지 확인합니다.	
Aurora Serverless v1 클러스터에서 변환	Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드의 프로시저를 따르세요.	해당 사항 없음
사용 가능한 DB 인스턴스 클래스	특수 DB 인스턴스 클래스 B1 AWS Management Console에서는 Serverless 레이블이 지정됩니다.	해당 사항 없음. Aurora Serverless v1은 `serverless` 엔진 모드를 사용합니다.
Port	MySQL 또는 PostgreSQL과 호환되는 모든 포트	기본 MySQL 또는 PostgreSQL 포트만
퍼블릭 IP 주소는 허용되나요?	예	아니요
Virtual Private Cloud(VPC)가 필요한가요?	예	예. 각 Aurora Serverless v1 클러스터는 VPC에 할당된 2개의 인터페이스와 게이트웨이 로드 밸런서 엔드포인트를 사용합니다.

Aurora Serverless v2 및 Aurora Serverless v1의 확장성과 가용성 비교

다음 테이블에는 확장성 및 가용성 측면에서 Aurora Serverless v2과 Aurora Serverless v1의 차이점이 요약되어 있습니다.

Aurora Serverless v2 확장은 Aurora Serverless v1의 확장보다 반응성이 더 크고 세분화되어 있으며 방해가 덜합니다. Aurora Serverless v2은 DB 인스턴스의 크기를 변경하고 DB 클러스터에 더 많은 DB 인스턴스를 추가하여 확장할 수 있습니다. 다른 AWS 리전의 클러스터를 Aurora 글로벌 데이터베이스에 추가하여 확장할 수도 있습니다. 이와 달리 Aurora Serverless v1은 라이터의 용량을 늘리거나 줄이는 방식으로만 확장됩니다. Aurora Serverless v1 클러스터의 모든 컴퓨팅은 단일 가용 영역과 단일 AWS 리전에서 실행됩니다.

크기 조정 및 고가용성 기능	Aurora Serverless v2 동작	Aurora Serverless v1 동작
최소 Aurora 용량 단위(ACU)(Aurora MySQL)	클러스터가 실행 중이면 0.5, 클러스터가 일시 중지되면 0입니다.	클러스터가 실행 중이면 1, 클러스터가 일시 중지되면 0입니다.
최소 ACU(Aurora PostgreSQL)	클러스터가 실행 중이면 0.5, 클러스터가 일시 중지되면 0입니다.	클러스터가 실행 중이면 2, 클러스터가 일시 중지되면 0입니다.
최대 ACU(Aurora MySQL)	256	256
최대 ACU(Aurora PostgreSQL)	256	384
클러스터 중지	프로비저닝된 클러스터와 [동일한 클러스터 중지 및 시작 기능]을 사용하여 클러스터를 수동으로 중지 및 시작할 수 있습니다.	클러스터는 시간 초과 후 자동으로 일시 중지됩니다. 활동이 재개되면 사용할 수 있게 되기까지 시간이 걸립니다.
DB 인스턴스 크기 조정	최소 0.5 ACU 증분으로 스케일 업 및 다운합니다.	ACU를 두 배로 늘리거나 반으로 줄여 스케일 업 및 다운합니다.
DB 인스턴스 개수	프로비저닝된 클러스터와 동일: 1개의 라이터 DB 인스턴스, 최대 15개의 리더 DB 인스턴스	읽기 및 쓰기를 모두 처리하는 DB 인스턴스 1개
SQL 문이 실행되는 동안 확장이 발생할 수 있나요?	예. Aurora Serverless v2는 조용한 지점을 기다릴 필요가 없습니다.	아니요. 예를 들어 확장은 장기 실행 트랜잭션, 임시 테이블 및 테이블 잠금이 완료될 때까지 기다립니다.
리더 DB 인스턴스는 라이터와 함께 확장됩니다.	선택 사항	해당 사항 없음
최대 스토리지	128TiB	128TiB

크기 조정 및 고가용성 기능	Aurora Serverless v2 동작	Aurora Serverless v1 동작
확장 시 버퍼 캐시가 보관됨	예. 버퍼 캐시는 동적으로 크기가 조정됩니다.	아니요. 버퍼 캐시는 크기 조정 후 다시 예열됩니다.
장애 조치	예. 프로비저닝된 클러스터와 동일합니다.	최상의 노력만, 용량 가용성에 따라 다름 Aurora Serverless v2에서보다 느림
다중 AZ 기능	예. 예, 프로비저닝된 것과 동일합니다. 다중 AZ 클러스터에는 두 번째 가용 영역(AZ)에 리더 DB 인스턴스가 필요합니다. 다중 AZ 클러스터의 경우 Aurora는 AZ 장애가 발생할 경우 다중 AZ 장애 조치를 수행합니다.	Aurora Serverless v1 클러스터는 단일 AZ에서 모든 컴퓨팅을 실행합니다. AZ 장애 발생 시 최선을 다해 복구하지만 용량 가용성에 따라 달라질 수 있습니다.
Aurora Global Database	예	아니요
메모리 압력에 따라 크기 조정	예	아니요
CPU 로드에 따라 크기 조정	예	예
네트워크 트래픽에 따라 크기 조정	예. 네트워크 트래픽의 메모리 및 CPU 오버헤드를 기준으로 합니다. B1 파라미터는 축소 시 연결 끊김을 방지하기 위해 일정하게 유지됩니다.	예. 연결 수에 따라 다릅니다.
크기 조정 이벤트에 대한 시간 초과 작업	아니요	예
B1를 통해 클러스터에 새 DB 인스턴스 추가	해당 사항 없음. 프로모션 티어 2~15에서 B1 리더 DB 인스턴스를 생성하고 낮은 용량으로 축소된 상태로 둘 수 있습니다.	아니요. 리더 DB 인스턴스는 사용할 수 없습니다.

Aurora Serverless v2 및 Aurora Serverless v1 기능 지원 비교

다음 테이블에는 이에 대한 내용이 요약되어 있습니다.

- Aurora Serverless v2에서는 사용할 수 있지만 Aurora Serverless v1에서는 사용할 수 없는 기능
- Aurora Serverless v1과 Aurora Serverless v2에서 다르게 작동하는 기능
- 현재 Aurora Serverless v2에서 사용할 수 없는 기능

Aurora Serverless v2에는 Aurora Serverless v1에서 사용할 수 없는 프로비저닝된 클러스터의 많은 기능이 포함되어 있습니다.

Feature	Aurora Serverless v2 지원	Aurora Serverless v1 지원
클러스터 토폴로지	Aurora Serverless v2는 개별 DB 인스턴스의 속성입니다. 클러스터에는 여러 Aurora Serverless v2 DB 인스턴스 또는 Aurora Serverless v2와 프로비저닝된 DB 인스턴스의 조합이 포함될 수 있습니다.	Aurora Serverless v1 클러스터는 DB 인스턴스의 개념을 사용하지 않습니다. 클러스터를 생성한 후에는 Aurora Serverless v1 속성을 변경할 수 없습니다.
구성 파라미터	프로비저닝된 클러스터에서와 거의 동일한 파라미터를 수정할 수 있습니다. 자세한 내용은 [Aurora Serverless v2에 대한 파라미터 그룹 작업](#) 페이지를 참조하세요.	파라미터의 하위 집합만 수정할 수 있습니다.
파라미터 그룹	클러스터 파라미터 그룹 및 DB 파라미터 그룹 `SupportedEngineModes` 속성에 `provisioned` 값이 있는 파라미터를 사용할 수 있습니다. Aurora Serverless v1보다 개수가 더 많은 파라미터입니다.	클러스터 파라미터 그룹만 해당 `SupportedEngineModes` 속성에 `serverless` 값이 있는 파라미터를 사용할 수 있습니다.

Feature	Aurora Serverless v2 지원	Aurora Serverless v1 지원
클러스터 볼륨에 대한 암호화	선택 사항	필수 사항입니다. Amazon Aurora 암호화된 DB 클러스터의 제한의 제한 사항은 모든 Aurora Serverless v1 클러스터에 적용됩니다.
리전 간 스냅샷	예	스냅샷은 자신의 AWS Key Management Service(AWS KMS) 키로 암호화해야 합니다.
DB 클러스터 삭제 후 자동 백업 유지	예	아니요
TLS/SSL	예. 지원은 프로비저닝된 클러스터와 동일합니다. 사용 정보는 Aurora Serverless v2에서 TLS/SSL 사용 단원을 참조하세요.	예. 프로비저닝된 클러스터에 대한 TLS 지원과 몇 가지 차이점이 있습니다. 사용 정보는 Aurora Serverless v1에서 TLS/SSL 사용 단원을 참조하세요.
복제	Aurora Serverless v2와 호환되는 DB 엔진 버전에서/버전으로만 가능합니다. 복제를 사용하여 Aurora Serverless v1 또는 프로비저닝된 클러스터의 이전 버전에서 업그레이드할 수 없습니다.	Aurora Serverless v1와 호환되는 DB 엔진 버전에서/버전으로만 가능합니다.
Amazon S3와 통합	예	예
AWS Secrets Manager와의 통합	예	아니요
S3로 DB 클러스터 스냅샷 내보내기	예	아니요
IAM 역할 연결	예	아니요

Feature	Aurora Serverless v2 지원	Aurora Serverless v1 지원
Amazon CloudWatch에 로그 업로드	선택 사항. 켤 로그와 CloudWatch에 업로드할 로그를 선택합니다.	켜져 있는 모든 로그는 CloudWatch에 자동으로 업로드됩니다.
데이터 API 사용 가능	예	예
쿼리 편집기 사용 가능	예	예
성능 개선 도우미	예	아니요
Amazon RDS 프록시 사용 가능	예	아니요
Babelfish for Aurora PostgreSQL 사용 가능	예. Babelfish 및 Aurora Serverless v2 모두와 호환되는 Aurora PostgreSQL 버전에서 지원됩니다.	아니요

Aurora Serverless v1 사용 사례를 Aurora Serverless v2에 적용하기

Aurora Serverless v1의 사용 사례에 따라 다음과 같이 Aurora Serverless v2 기능을 활용하기 위해 해당 접근 방식을 조정할 수 있습니다.

로드가 적은 Aurora Serverless v1 클러스터가 있고 우선 순위가 비용을 최소화하면서 지속적인 가용성을 유지하는 것이라고 가정합니다. Aurora Serverless v2를 사용하면 Aurora Serverless v1의 최소 ACU 1개와 비교하여 0.5의 더 작은 최소 ACU 설정을 구성할 수 있습니다. 리더 DB 인스턴스에도 최소 0.5개의 ACU가 있는 다중 AZ 구성을 생성하여 가용성을 높일 수 있습니다.

개발 및 테스트 시나리오에서 사용하는 Aurora Serverless v1 클러스터가 있다고 가정합니다. 이 경우 비용도 우선순위가 높지만 클러스터가 항상 사용 가능할 필요는 없습니다. Aurora Serverless v2는 각 인스턴스가 완전히 유휴 상태일 때 각 인스턴스를 자동으로 일시 중지할 수 있습니다. Aurora Serverless v2에 대해 자동 일시 중지 및 재개를 사용하여 0ACU로 크기 조정에 설명된 대로 클러스터에 최소 ACU 용량을 0으로 지정하면 됩니다. 클러스터가 필요하지 않을 때 수동으로 중지하고 다음 테스트 또는 개발 주기가 되면 시작할 수도 있습니다.

워크로드가 많은 B1 클러스터가 있다고 가정합니다. Aurora Serverless v2를 사용하는 동등한 클러스터는 더 세분화하여 확장할 수 있습니다. 예를 들어 Aurora Serverless v1은 용량을 두 배로 늘려 확장

합니다(예: 64에서 128 ACU로 확장). 반대로 Aurora Serverless v2 DB 인스턴스는 0.5ACU 단위로 스케일 인할 수 있습니다.

워크로드에 Aurora Serverless v1에서 사용 가능한 것보다 더 많은 총 용량이 필요하다고 가정합니다. 여러 Aurora Serverless v2 리더 DB 인스턴스를 사용하여 라이터 DB 인스턴스에서 워크로드의 읽기 집약적인 부분을 오프로드할 수 있습니다. 읽기 집약적 워크로드를 여러 리더 DB 인스턴스로 나눌 수도 있습니다.

쓰기 집약적인 워크로드의 경우 대규모 프로비저닝된 DB 인스턴스를 라이터로 사용하여 클러스터를 구성할 수 있습니다. 이때 하나 이상의 Aurora Serverless v2 리더 DB 인스턴스와 함께 사용할 수 있습니다.

Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

DB 클러스터를 Aurora Serverless v1에서 Aurora Serverless v2로 업그레이드하는 프로세스는 여러 단계로 이루어집니다. Aurora Serverless v1에서 Aurora Serverless v2로 직접 변환할 수 없기 때문입니다. Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환하는 중간 단계가 항상 있습니다.

Aurora MySQL 호환 DB 클러스터

Aurora Serverless v1 DB 클러스터를 프로비저닝된 DB 클러스터로 변환한 다음 블루/그린 배포를 사용하여 업그레이드하고 다시 Aurora Serverless v2 DB 클러스터로 변환할 수 있습니다. 프로덕션 환경에서 이 절차를 진행하는 것이 좋습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 섹션을 참조하세요.

블루/그린 배포를 사용하여 Aurora MySQL 버전 2(MySQL 5.7 호환)를 실행하는 Aurora Serverless v1 클러스터를 업그레이드하는 방법

1. Aurora Serverless v1 DB 클러스터를 프로비저닝된 Aurora MySQL 버전 2 클러스터로 변환합니다. Aurora Serverless v1에서 프로비저닝으로 변환의 프로시저를 따르세요.
2. 블루/그린 배포를 생성합니다. Amazon Aurora에서 블루/그린 배포 생성의 프로시저를 따르세요.
3. 그린 클러스터에는 Aurora Serverless v2와 호환되는 Aurora MySQL 버전을 선택하세요(예: 3.04.1).

   호환 가능한 버전은 Aurora MySQL을 사용하는 Aurora Serverless v2 섹션을 참조하세요.
4. 그린 클러스터의 라이터 DB 인스턴스를 수정하여 Serverless v2(db.serverless) DB 인스턴스 클래스를 사용하도록 합니다.

   자세한 내용은 프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환을 참조하세요.
5. 업그레이드된 Aurora Serverless v2 DB 클러스터를 사용할 수 있게 되면 블루 클러스터에서 그린 클러스터로 전환합니다.

Aurora PostgreSQL 호환 DB 클러스터

Aurora Serverless v1 DB 클러스터를 프로비저닝된 DB 클러스터로 변환한 다음 블루/그린 배포를 사용하여 업그레이드하고 다시 Aurora Serverless v2 DB 클러스터로 변환할 수 있습니다. 프로덕션 환경에서 이 절차를 진행하는 것이 좋습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon Aurora 블루/그린 배포 사용 섹션을 참조하세요.

블루/그린 배포를 사용하여 Aurora PostgreSQL 버전 11을 실행하는 Aurora Serverless v1 클러스터를 업그레이드하는 방법

1. Aurora Serverless v1 DB 클러스터를 프로비저닝된 Aurora PostgreSQL 클러스터로 변환합니다. Aurora Serverless v1에서 프로비저닝으로 변환의 프로시저를 따르세요.
2. 블루/그린 배포를 생성합니다. Amazon Aurora에서 블루/그린 배포 생성의 프로시저를 따르세요.
3. 그린 클러스터에는 Aurora Serverless v2와 호환되는 Aurora PostgreSQL 버전을 선택하세요(예: 15.3).

   호환 가능한 버전은 Aurora PostgreSQL을 사용하는 Aurora Serverless v2 섹션을 참조하세요.
4. 그린 클러스터의 라이터 DB 인스턴스를 수정하여 Serverless v2(db.serverless) DB 인스턴스 클래스를 사용하도록 합니다.

자세한 내용은 프로비저닝된 라이터 또는 리더를 Aurora Serverless v2로 변환을 참조하세요.

5. 업그레이드된 Aurora Serverless v2 DB 클러스터를 사용할 수 있게 되면 블루 클러스터에서 그린 클러스터로 전환합니다.

Aurora PostgreSQL 버전 11에서 버전 13으로 Aurora Serverless v1 DB 클러스터를 직접 업그레이드하고 프로비저닝된 DB 클러스터로 변환한 다음, 프로비저닝된 클러스터를 Aurora Serverless v2 DB 클러스터로 변환할 수도 있습니다.

Aurora PostgreSQL 버전 11을 실행하는 Aurora Serverless v1 클러스터를 업그레이드하고 변환하는 방법

1. Aurora Serverless v1 DB 클러스터를 프로비저닝된 Aurora PostgreSQL 클러스터로 변환합니다. Aurora Serverless v1에서 프로비저닝으로 변환의 프로시저를 따르세요.
2. Aurora Serverless v1 클러스터를 Aurora Serverless v2와 호환되는 Aurora PostgreSQL 버전 13 버전(예: 13.12)으로 업그레이드합니다. 메이저 버전 업그레이드의 프로시저를 따르세요.

   호환 가능한 버전은 Aurora PostgreSQL을 사용하는 Aurora Serverless v2 섹션을 참조하세요.
3. 클러스터에 Aurora Serverless v2 리더 DB 인스턴스를 추가합니다. 자세한 내용은 Aurora Serverless v2 리더 추가 섹션을 참조하세요.
4. Aurora Serverless v2 DB 인스턴스에 대한 장애 조치:

   a. DB 클러스터의 라이터 DB 인스턴스를 선택합니다.
   b. 작업(Actions)으로 장애 조치(Failover)를 선택합니다.
   c. 확인 페이지에서 장애 조치를 선택합니다.

Aurora PostgreSQL 버전 13을 실행하는 Aurora Serverless v1 DB 클러스터의 경우, Aurora Serverless v1 클러스터를 프로비저닝된 DB 클러스터로 변환한 다음, 프로비저닝된 클러스터를 Aurora Serverless v2 DB 클러스터로 변환합니다.

Aurora PostgreSQL 버전 13을 실행하는 Aurora Serverless v1 클러스터를 업그레이드하려면

1. Aurora Serverless v1 DB 클러스터를 프로비저닝된 Aurora PostgreSQL 클러스터로 변환합니다. Aurora Serverless v1에서 프로비저닝으로 변환의 프로시저를 따르세요.
2. 클러스터에 Aurora Serverless v2 리더 DB 인스턴스를 추가합니다. 자세한 내용은 Aurora Serverless v2 리더 추가 섹션을 참조하세요.
3. Aurora Serverless v2 DB 인스턴스에 대한 장애 조치:

   a. DB 클러스터의 라이터 DB 인스턴스를 선택합니다.
   b. 작업(Actions)으로 장애 조치(Failover)를 선택합니다.
   c. 확인 페이지에서 장애 조치를 선택합니다.
4. 리더 인스턴스를 제거합니다.

Aurora Serverless v2로 온프레미스 데이터베이스 마이그레이션

프로비저닝된 Aurora MySQL 및 Aurora PostgreSQL과 마찬가지로 온프레미스 데이터베이스를 Aurora Serverless v2로 마이그레이션할 수 있습니다.

- MySQL 데이터베이스의 경우 `mysqldump` 명령을 사용할 수 있습니다. 자세한 내용은 가동 중지 시간을 단축하여 Amazon RDS MySQL MariaDB DB 인스턴스로 데이터 가져오기를 참조하세요.
- PostgreSQL 데이터베이스의 경우 `pg_dump` 및 `pg_restore` 명령을 사용할 수 있습니다. 자세한 내용은 블로그 게시물 PostgreSQL 데이터베이스를 Amazon RDS 및 Amazon Aurora로 마이그레이션하기 위한 모범 사례를 참조하세요.

Amazon Aurora Serverless v1 사용

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Amazon Aurora Serverless v1(Amazon Aurora Serverless 버전 1)은 Amazon Aurora에 대한 온디맨드 방식의 자동 크기 조정 구성입니다. Aurora Serverless v1 DB 클러스터는 애플리케이션의 요구 사항에 따라 컴퓨팅 용량을 확장 및 축소하는 DB 클러스터입니다. 이는 용량을 수동으로 관리하는 Aurora 프로비저닝된 DB 클러스터와 대조적입니다. Aurora Serverless v1은 빈도가 낮거나 간헐적이거나 예측할 수 없는 워크로드에 대해 상대적으로 단순하고 비용 효율적인 옵션을 제공합니다. 자동으로 시작하고 애플리케이션의 사용량에 맞춰 컴퓨팅 용량을 확장하고, 사용하지 않는 경우 종료되기 때문에 비용 효율적입니다.

요금에 대한 자세한 내용은 Amazon Aurora pricing 페이지의 MySQL 호환 버전 또는 PostgreSQL 호환 버전에서 Serverless 요금을 참조하세요.

Aurora Serverless v1 클러스터는 프로비저닝된 DB 클러스터에서 사용하는 것과 동일한 유형의 분산된 고가용성 대용량 스토리지 볼륨을 갖습니다.

Aurora Serverless v1 클러스터의 경우 클러스터 볼륨은 항상 암호화됩니다. 암호화 키를 선택할 수 있지만 암호화를 비활성화할 수는 없습니다. 즉, 암호화된 스냅샷에서 수행할 수 있는 것과 동일한 작업을 Aurora Serverless v1에서 수행할 수 있습니다. 자세한 내용은 Aurora Serverless v1 및 스냅샷 섹션을 참조하세요.

주제

- Aurora Serverless v1의 구성 요구 사항
- Aurora Serverless v1에서 TLS/SSL 사용
- Aurora Serverless v1 작동 방식
- Aurora Serverless v1 DB 클러스터 생성
- Aurora Serverless v1 DB 클러스터 복원
- Aurora Serverless v1 DB 클러스터 수정
- 수동으로 Aurora Serverless v1 DB 클러스터 용량 확장
- Aurora Serverless v1 DB 클러스터 보기
- Aurora Serverless v1 DB 클러스터 삭제
- Aurora Serverless v1 및 Aurora 데이터베이스 엔진 버전

Aurora Serverless v1의 리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. Aurora와 Aurora Serverless v1 버전 및 리전 가용성에 대한 자세한 정보는 Aurora Serverless v1 섹션을 참조하세요.

Aurora Serverless v1의 장점

Aurora Serverless v1의 장점은 다음과 같습니다.

- 프로비저닝보다 간단 - Aurora Serverless v1은 DB 인스턴스 및 용량 관리의 복잡성을 크게 줄입니다.
- 확장성 - Aurora Serverless v1은 클라이언트 연결을 중단하지 않고 필요에 따라 컴퓨팅 및 메모리 용량을 원활하게 확장합니다.
- 비용 효율성 - Aurora Serverless v1을 사용하는 경우 사용한 데이터베이스 리소스에 대해서만 초 단위로 요금을 지불합니다.
- 고가용성 스토리지 - Aurora Serverless v1은 6방향 복제가 가능하고 Aurora와 동일한 내결함성 분산 스토리지 시스템을 사용해 데이터 손실을 방지합니다.

Aurora Serverless v1 사용 사례

Aurora Serverless v1는 다음과 같은 사용 사례를 위해 설계되었습니다.

- 자주 사용하지 않는 애플리케이션 – 하루 또는 일주일에 고작 몇 분씩 몇 차례만 사용하는 애플리케이션이 있습니다(예: 사용자가 적은 블로그 사이트). Aurora Serverless v1를 사용하는 경우 사용한 데이터베이스 리소스에 대해서만 초 단위로 요금을 지불합니다.
- 새 애플리케이션 – 새 애플리케이션을 배포하는 중인데, 필요한 인스턴스 크기를 잘 모릅니다. Aurora Serverless v1를 사용하면 데이터베이스 엔드포인트를 생성해 애플리케이션의 용량 요구 사항에 따라 데이터베이스가 자동으로 확장되게 할 수 있습니다.
- 가변성 높은 워크로드 – 피크가 하루에 몇 번 안 되거나 1년에 몇 번에 불과하고 30분~몇 시간에 불과한 사용량이 낮은 애플리케이션을 실행하고 있습니다. 인사 관리, 예산 작성 및 운영 보고 애플리케이션을 예로 들 수 있습니다. Aurora Serverless v1를 사용하면 더 이상 피크 또는 평균 용량에 맞추어 프로비저닝하지 않아도 됩니다.
- 예측 불가능한 워크로드 – 작업의 증가가 갑작스럽고 예측할 수 없는 일일 워크로드를 실행하고 있습니다. 비가 내리기 시작하면 활동이 급증하는 트래픽 사이트를 예로 들 수 있습니다. Aurora Serverless v1를 사용하면 애플리케이션의 피크 로드를 처리하는 데 필요한 용량을 충족하도록 데이터베이스가 용량을 자동으로 확장하고 활동이 급증하는 시점이 지나면 용량이 다시 줄입니다.
- 개발 및 테스트 데이터베이스 – 개발자가 작업 시간 중에는 데이터베이스를 사용하지만 밤이나 주말에는 데이터베이스를 사용하지 않습니다. Aurora Serverless v1를 사용하면 데이터베이스가 사용 중이 아닌 경우 자동으로 종료됩니다.
- 다중 테넌트 애플리케이션 - Aurora Serverless v1을 사용하면 플릿에서 각 애플리케이션에 대한 데이터베이스 용량을 개별적으로 관리할 필요가 없습니다. Aurora Serverless v1은 사용자를 대신하여 개별 데이터베이스 용량을 관리해 줍니다.

Aurora Serverless v1의 제한 사항

Aurora Serverless v1에는 다음과 같은 제한 사항이 적용됩니다.

- > **Important**
  >
  > AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.
- Aurora Serverless v1는 다음과 같은 기능을 지원하지 않습니다.

  - Aurora Global Database
  - Aurora 복제본
  - AWS Identity and Access Management(IAM) 데이터베이스 인증
  - Aurora 내 역추적
  - 데이터베이스 활동 스트림
  - Kerberos 인증
  - 성능 개선 도우미
  - RDS 프록시
  - AWS Management Console에서 로그 보기
- Aurora Serverless v1 DB 클러스터에 대한 연결이 하루 이상 열려 있으면 자동으로 닫힙니다.
- 모든 Aurora Serverless v1 DB 클러스터에는 다음과 같은 제한 사항이 있습니다.
  - Aurora Serverless v1 스냅샷을 Amazon S3 버킷으로 내보낼 수 없습니다.
  - AWS Database Migration Service 및 변경 데이터 캡처(CDC)를 Aurora Serverless v1 DB 클러스터와 함께 사용할 수 없습니다. 프로비저닝된 Aurora DB 클러스터만 소스로서 AWS DMS와 함께 CDC를 지원합니다.
  - Amazon S3의 텍스트 파일에 데이터를 저장하거나 S3의 텍스트 파일 데이터를 Aurora Serverless v1에 로드할 수 없습니다.
  - IAM 역할을 Aurora Serverless v1 DB 클러스터에 첨부할 수 없습니다. 하지만 Aurora Serverless v1 함수 및 `aws_s3` 파라미터와 함께 `aws_s3.table_import_from_s3` 확장을 사용하여 `credentials`에서 Amazon S3로 데이터를 로드할 수 있습니다. 자세한 내용은 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터 섹션을 참조하세요.
  - 쿼리 편집기를 사용할 때 DB 보안 인증 정보가 데이터베이스에 액세스할 수 있도록 Secrets Manager 암호가 생성됩니다. 쿼리 편집기에서 보안 인증 정보를 삭제하면 Secrets Manager에서도 관련 암호가 삭제됩니다. 이러한 암호가 삭제된 후에는 복구할 수 없습니다.
- Aurora Serverless v1을 실행하는 Aurora MySQL 기반 DB 클러스터는 다음을 지원하지 않습니다.
  - Aurora MySQL DB 클러스터 내에서 AWS Lambda 함수 호출. 하지만 AWS Lambda 함수는 Aurora Serverless v1 DB 클러스터를 호출할 수 있습니다.
  - Aurora MySQL 또는 RDS for MySQL이 아닌 DB 인스턴스에서 스냅샷 복원.
  - 이진 로그(binlogs) 기반의 복제를 사용한 데이터 복제. 이 제한은 Aurora MySQL 기반 DB 클러스터 Aurora Serverless v1이 복제의 소스인지 대상인지에 관계없이 적용됩니다. Amazon EC2에서 실행되는 것과 같이 Aurora 외부의 MySQL DB 인스턴스에서 Aurora Serverless v1 DB 클러스터

로 데이터를 복제하려면 AWS Database Migration Service를 사용하는 것이 좋습니다. 자세한 내용은 AWS Database Migration Service 사용 설명서를 참조하세요.

- 호스트 기반 액세스 권한을 가진 사용자 생성('*username*'@'*IP_address*'). 이는 Aurora Serverless v1이 원활한 확장을 위해 클라이언트와 데이터베이스 호스트 간에 라우터 플릿을 사용하기 때문입니다. Aurora Serverless DB 클러스터에 표시되는 IP 주소는 클라이언트가 아닌 라우터 호스트의 IP 주소입니다. 자세한 내용은 Aurora Serverless v1 아키텍처 섹션을 참조하세요.

  대신, 와일드카드('*username*'@'%')를 사용합니다.

- Aurora Serverless v1을 실행하는 Aurora PostgreSQL 기반 DB 클러스터의 경우 다음과 같은 제한 사항이 있습니다.
  - Aurora PostgreSQL 쿼리 계획 관리(`apg_plan_management` 확장)은 지원되지 않습니다.
  - Amazon RDS PostgreSQL 및 Aurora PostgreSQL에서 제공되는 논리적 복제 기능은 지원되지 않습니다.
  - Amazon RDS for PostgreSQL 확장에 의해 활성화된 아웃바운드 통신과 같은 아웃바운드 통신은 지원되지 않습니다. 예를 들어 `postgres_fdw/dblink` 확장을 사용하여 외부 데이터에 액세스할 수 없습니다. RDS PostgreSQL 확장에 대한 자세한 내용은 RDS 사용 설명서에서 Amazon RDS의 PostgreSQL을 참조하세요.
  - 현재 특정 SQL 쿼리 및 명령은 권장되지 않습니다. 여기에는 세션 수준 공지 잠금, 임시 관계, 비동기 알림(`LISTEN`) 및 hold가 있는 커서(`DECLARE` *name* `... CURSOR WITH HOLD FOR` *query*)가 포함됩니다. 또한 `NOTIFY` 명령은 확장을 방지하므로 권장되지 않습니다.

    자세한 내용은 Aurora Serverless v1에서의 Auto Scaling 섹션을 참조하세요.
- Aurora Serverless v1 DB 클러스터의 기본 자동 백업 기간을 설정할 수 없습니다.
- Aurora Serverless v1 DB 클러스터의 유지 관리 기간을 설정할 수 있습니다. 자세한 내용은 기본 DB 클러스터 유지 관리 기간 조정 섹션을 참조하세요.

Aurora Serverless v1의 구성 요구 사항

Aurora Serverless v1 DB 클러스터를 생성할 때는 다음 요구 사항에 주의하세요.

- 각 DB 엔진에 대해 다음과 같은 특정 포트 번호를 사용합니다.
  - Aurora MySQL – `3306`
  - Aurora PostgreSQL – `5432`
- Amazon VPC 서비스 기반의 Virtual Private Cloud(VPC)에서 Aurora Serverless v1 DB 클러스터를 생성합니다. VPC에서 Aurora Serverless v1 DB 클러스터를 생성할 때 VPC에 할당된 50개의 인터

페이스 및 게이트웨이 로드 밸런서 엔드포인트 중 2개가 사용됩니다. 이러한 엔드포인트는 자동으로 생성됩니다. 할당량을 늘리려면 지원에 연락하면 됩니다. 자세한 내용은 Amazon VPC할당량을 참조하세요.

- Aurora Serverless v1 DB 클러스터에는 퍼블릭 IP 주소를 부여할 수 없습니다. VPC 내에서만 Aurora Serverless v1 DB 클러스터에 액세스할 수 있습니다.
- Aurora Serverless v1 DB 클러스터에 사용하는 DB 서브넷 그룹에 대해 서로 다른 가용 영역에 서브넷을 생성합니다. 즉, 동일한 가용 영역에 서브넷을 2개 이상 보유할 수 없습니다.
- Aurora Serverless v1 DB 클러스터에서 사용하는 서브넷 그룹에 대한 변경 사항은 클러스터에 적용되지 않습니다.
- Aurora Serverless v1에서 AWS Lambda DB 클러스터에 액세스할 수 있습니다. 그러려면 Lambda 함수가 Aurora Serverless v1 DB 클러스터와 동일한 VPC에서 실행되도록 구성해야 합니다. AWS Lambda 작업에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Amazon VPC의 리소스에 액세스하도록 Lambda 함수 구성을 참조하세요.

Aurora Serverless v1에서 TLS/SSL 사용

기본적으로 Aurora Serverless v1는 TLS/SSL(Transport Layer Security/Secure Sockets Layer) 프로토콜을 사용하여 클라이언트와 Aurora Serverless v1 DB 클러스터 간의 통신을 암호화합니다. TLS/SSL 버전 1.0, 1.1 및 1.2를 지원합니다. TLS/SSL을 사용하도록 Aurora Serverless v1 DB 클러스터를 구성할 필요가 없습니다.

단, 다음 제한 사항이 적용됩니다.

- Aurora Serverless v1 DB 클러스터에 대한 TLS/SSL 지원은 현재 중국(베이징) AWS 리전에서 사용할 수 없습니다.
- Aurora MySQL 기반 Aurora Serverless v1 DB 클러스터의 데이터베이스 사용자를 생성할 때는 SSL 권한에 `REQUIRE` 절을 사용해서는 안 됩니다. 이렇게 하면 사용자가 Aurora DB 인스턴스에 연결할 수 없습니다.
- MySQL 클라이언트 및 PostgreSQL 클라이언트 유틸리티 모두, 클라이언트와 Aurora Serverless v1 간에 TLS/SSL을 사용할 때 다른 환경에서 사용하는 세션 변수는 아무런 영향을 미치지 않습니다.
- MySQL 클라이언트의 경우 TLS/SSL `VERIFY_IDENTITY` 모드로 연결할 때 현재, MySQL 8.0 호환 `mysql` 명령을 사용해야 합니다. 자세한 내용은 MySQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기를 참조하세요.

Aurora Serverless v1 DB 클러스터에 연결하는 데 사용하는 클라이언트에 따라 암호화된 연결을 설정하기 위해 TLS/SSL을 지정하지 않아도 됩니다. 예를 들어 Aurora PostgreSQL 호환 버전을 실행 중인 Aurora Serverless v1 DB 클러스터에 PostgreSQL 클라이언트를 사용하여 연결하려면 평소와 같이 연결합니다.

```
psql -h endpoint -U user
```

암호를 입력하면 PostgreSQL 클라이언트에서 TLS/SSL 버전 및 암호를 포함한 연결 세부 정보를 볼 수 있습니다.

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

Important

Aurora Serverless v1은 클라이언트 애플리케이션에서 SSL/TLS를 사용 중지 하지 않는 한 전송 계층 보안/보안 소켓 계층(TLS/SSL) 프로토콜을 사용하여 기본적으로 연결을 암호화합니다. TLS/SSL 연결은 라우터 플릿에서 종료됩니다. 라우터 플릿과 Aurora Serverless v1 DB 클러스터 간의 통신은 서비스의 내부 네트워크 경계 내에서 발생합니다.

클라이언트 연결 상태를 확인하여 Aurora Serverless v1에 대한 연결이 TLS/SSL로 암호화되었는지 여부를 확인할 수 있습니다. PostgreSQL `pg_stat_ssl` 및 `pg_stat_activity` 테이블과 해당하는 `ssl_is_used` 함수는 클라이언트 애플리케이션과 Aurora Serverless v1 간의 통신에 대한 TLS/SSL 상태를 표시하지 않습니다. 마찬가지로 MySQL `status` 문에서 TLS/SSL 상태를 도출할 수 없습니다.

Aurora 클러스터 파라미터(PostgreSQL의 경우 `force_ssl`이고 MySQL의 경우 `require_secure_transport`)는 이전에 Aurora Serverless v1에 대해 지원되지 않았습니다. 이 파라미터는 현재 Aurora Serverless v1에 대해 사용할 수 있습니다. Aurora Serverless v1에서 지원하는 전체 파라미터 목록을 보려면 DescribeEngineDefaultClusterParameters API 작업을 호출하세요. 파라미터 그룹 및 Aurora Serverless v1에 대한 자세한 내용은 Aurora Serverless v1 파라미터 그룹 단원을 참조하세요.

Aurora MySQL 호환 버전을 실행 중인 Aurora Serverless v1 DB 클러스터에 MySQL 클라이언트를 사용하여 연결하려면 요청에 TLS/SSL을 지정합니다. 다음 예에는 Amazon Trust Services에서 다운로드한 Amazon 루트 CA 1 신뢰할 수 있는 스토어가 포함되어 있습니다. Trust Services는 이 연결에 성공하기 위해 필요합니다.

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

메시지가 표시되면 암호를 입력합니다. 곧 MySQL 모니터가 열립니다. `status` 명령을 사용하여 세션이 암호화되었는지 확인할 수 있습니다.

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

MySQL 클라이언트를 사용하여 Aurora MySQL 데이터베이스에 연결하는 방법에 대한 자세한 내용은 MySQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기를 참조하세요.

Aurora Serverless v1는 다음 표에 나열된 것을 포함하여 MySQL 클라이언트(`mysql`) 및 PostgreSQL 클라이언트(`psql`)에서 사용 가능한 모든 TLS/SSL 모드를 지원합니다.

TLS/SSL 모드에 대한 설명	mysql	psql
TLS/SSL을 사용하지 않고 연결합니다.	비활성화됨	disable
먼저 TLS/SSL을 사용하여 연결을 시도하지만 필요한 경우 비 SSL로 폴백합니다.	PREFERRED	prefer(기본값)
TLS/SSL을 사용하도록 합니다.	필수	require
TLS/SSL을 적용하고 CA를 확인합니다.	VERIFY_CA	[verify-ca]
TLS/SSL을 적용하고 CA를 확인한 다음 CA 호스트 이름을 확인합니다.	VERIFY_IDENTITY	[verify-full]

Aurora Serverless v1는 와일드카드 인증서를 사용합니다. TLS/SSL을 사용할 때 "CA 확인" 또는 "CA 및 CA 호스트 이름 확인" 옵션을 지정하는 경우 먼저 Amazon Trust Services에서 Amazon 루트 CA 1 신뢰할 수 있는 스토어를 다운로드합니다. 이렇게 하면 클라이언트 명령에서 이 PEM 형식의 파일을 식별할 수 있습니다. PostgreSQL 클라이언트를 사용하여 이렇게 하려면 다음을 수행합니다.

대상 LinuxmacOS, 또는Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Postgres 클라이언트를 사용하여 Aurora PostgreSQL 데이터베이스 작업을 수행하는 방법에 대한 자세한 내용은 PostgreSQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기를 참조하세요.

일반적인 Aurora DB 클러스터에 연결하는 방법에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

Aurora Serverless v1 DB 클러스터에 연결을 위해 지원되는 암호 그룹

구성 가능한 암호 그룹을 사용하면 데이터베이스 연결의 보안을 더 잘 제어할 수 있습니다. 데이터베이스에 대한 클라이언트 TLS/SSL 연결을 보호하도록 허용할 암호 그룹 목록을 지정할 수 있습니다. 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. 이렇게 하면 안전하지 않거나 더 이상 사용되지 않는 암호의 사용을 방지할 수 있습니다.

Aurora MySQL 기반 Aurora Serverless v1 DB 클러스터는 Aurora MySQL 프로비저닝된 DB 클러스터와 동일한 암호 그룹을 지원합니다. 이러한 암호 그룹에 대한 자세한 내용은 Aurora MySQL DB 클러스터 연결을 위한 암호 그룹 구성 섹션을 참조하세요.

Aurora PostgreSQL 기반 Aurora Serverless v1 DB 클러스터는 암호 그룹을 지원하지 않습니다.

Aurora Serverless v1 작동 방식

Important

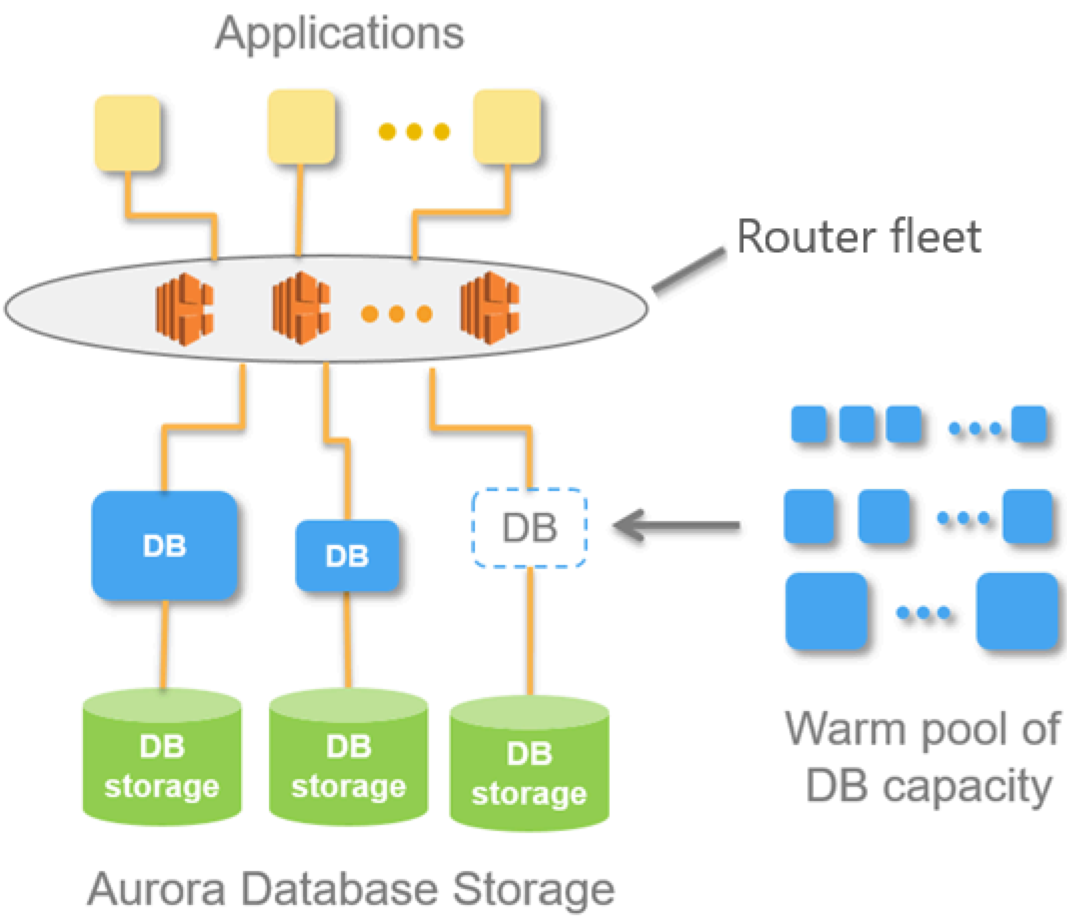
AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

다음으로 Aurora Serverless v1 작동 방식을 알아볼 수 있습니다.

주제

Aurora Serverless v1 아키텍처

다음 이미지는 Aurora Serverless v1 아키텍처의 개요를 보여줍니다.



데이터베이스 서버를 프로비저닝 및 관리하는 대신 ACU(Aurora 용량 단위)를 지정합니다. 각 ACU는 약 2GB 메모리, 해당 CPU 및 네트워킹의 조합입니다. 데이터베이스 스토리지는 표준 Aurora DB 클러스터의 스토리지와 동일한 10기비바이트(GiB)~128 tebibytes (TiB) 범위에서 자동으로 규모를 조정합니다.

최소 및 최대 ACU를 지정할 수 있습니다. 최소 Aurora 용량 단위는 DB 클러스터를 축소할 수 있는 가장 낮은 ACU입니다. 최대 Aurora 용량 단위는 DB 클러스터를 확장할 수 있는 가장 높은 ACU입니다. Aurora Serverless v1는 설정에 따라 CPU 사용률, 연결 및 가용 메모리 임계값에 대한 조정 규칙을 자동으로 생성합니다.

Aurora Serverless v1는 조정 시간을 최소화하기 위해 AWS 리전에서 리소스의 웜 풀을 관리합니다. Aurora Serverless v1이 Aurora DB 클러스터에 새 리소스를 추가하면 라우터 플릿을 사용하여 활성 클라이언트 연결을 새 리소스로 전환합니다. 특정 시간에 Aurora DB 클러스터에서 사용 중인 ACU에 대해서 비용이 청구됩니다.

Aurora Serverless v1에서의 Auto Scaling

Aurora Serverless v1 DB 클러스터에 할당된 용량은 클라이언트 애플리케이션에서 생성된 로드에 따라 원활하게 확장 및 축소됩니다. 여기서 부하는 CPU 사용률과 연결 수입니다. 이 중 하나에 의해 용량이 제한되는 경우, Aurora Serverless v1가 확장됩니다. 또한 Aurora Serverless v1는 이를 통해 해결할 수 있는 성능 문제를 감지한 경우에도 확장됩니다.

Aurora Serverless v1의 AWS Management Console 클러스터에 대한 크기 조정 이벤트를 볼 수 있습니다. Auto Scaling 동안 Aurora Serverless v1는 `EngineUptime` 지표를 재설정합니다. 재설정 지표 값은 원활한 크기 조정에 문제가 있다거나 Aurora Serverless v1 연결이 끊어졌음을 의미하지 않습니다. 이는 단순히 새로운 용량에서의 가동 시간 시작점일 뿐입니다. 지표에 대해 자세히 알아보려면 Amazon Aurora 클러스터에서 지표 모니터링 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터에 활성 연결이 없는 경우 용량을 0(ACU 0)으로 축소할 수 있습니다. 자세한 내용은 Aurora Serverless v1 일시 중지 및 다시 시작 단원을 참조하십시오.

크기 조정 작업을 수행해야 하는 경우, Aurora Serverless v1는 먼저 쿼리가 처리되지 않는 순간을 의미하는 조정점을 찾으려 합니다. 다음과 같은 이유로 Aurora Serverless v1가 조정점을 찾지 못할 수 있습니다.

- 장기 실행 쿼리
- 진행 중인 트랜잭션
- 임시 테이블 또는 테이블 잠금

조정점을 찾을 때 Aurora Serverless v1 DB 클러스터의 성공률을 높이려면 장기 실행 쿼리와 장기 실행 트랜잭션을 피하는 것이 좋습니다. 크기 조정을 저해하는 작업과 이러한 작업을 방지하는 방법에 대한 자세한 내용은 Aurora Serverless v1 작업 모범 사례를 참조하세요.

기본값으로 Aurora Serverless v1은(는) 5분(300초) 동안 크기 조정점을 찾으려고 시도합니다. 클러스터를 생성하거나 수정할 때 다른 제한 시간을 지정할 수 있습니다. 제한 시간은 60초에서 10분(600초) 사이일 수 있습니다. Aurora Serverless v1이(가) 지정 기간 내에 크기 조정점을 찾지 못하면 자동 크기 조정 작업 시간이 초과됩니다.

Auto Scaling이 시간 초과되기 전에 조정점을 찾지 못할 경우, 기본적으로 Aurora Serverless v1는 클러스터를 현재 용량으로 유지합니다. 이 기본 동작은 Aurora Serverless v1 DB 클러스터를 생성하거나 수정할 때 용량 변경 강제 적용 옵션을 선택하여 변경할 수 있습니다. 자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.

용량 변경을 위한 제한 시간 조치

크기 조정점 찾기로 인해 자동 크기 조정이 시간 초과되면 기본적으로 Aurora는 현재 용량을 유지합니다. [용량 변경 강제 적용(Force capacity change)] 옵션을 활성화하여 Aurora가 용량 변경을 강제 적용하도록 선택할 수 있습니다. 이 옵션은 클러스터를 생성할 때 데이터베이스 생성(Create database) 페이지의 자동 크기 조정 시간 초과 및 작업(Autoscaling timeout and action) 섹션에서 사용할 수 있습니다.

기본적으로 [용량 변경 강제 적용(Force capacity change)] 옵션은 선택되어 있지 않습니다. 크기 조정 지점을 찾지 못하고 크기 조정 작업이 시간 초과될 경우 Aurora Serverless v1 DB 클러스터의 용량이 변경되지 않도록 하려면 이 옵션을 선택하지 않은 상태로 둡니다.

이 옵션을 선택하면 조정점 없이도 Aurora Serverless v1 DB 클러스터에 용량 변경이 적용됩니다. 이 옵션을 선택하기 전에 이 선택의 결과를 고려해야 합니다.

- 진행 중인 트랜잭션이 중단되고 다음과 같은 오류 메시지가 나타납니다.

  Aurora MySQL 버전 2 - 오류 1105(HY000): 원활한 크기 조정으로 인해 마지막 트랜잭션이 중단되었습니다. 다시 시도하세요.

  Aurora Serverless v1 DB 클러스터를 사용할 수 있게 되는 즉시 트랜잭션을 다시 제출할 수 있습니다.
- 임시 테이블 및 잠금에 대한 연결이 삭제됩니다.

  끊긴 연결이나 불완전한 트랜잭션으로부터 애플리케이션을 복구할 수 있는 경우에만 [용량 변경 강제 적용(Force capacity change)] 옵션을 선택하는 것이 좋습니다.

AWS Management Console DB 클러스터를 생성할 때 Aurora Serverless v1에서 선택하는 옵션은 `ScalingConfigurationInfo` 객체의 `SecondsBeforeTimeout` 및 `TimeoutAction` 속성에 저장됩니다. `TimeoutAction` 속성의 값은 클러스터를 생성할 때 다음 값 중 하나로 설정됩니다.

- `RollbackCapacityChange` - 이 값은 용량 변경 롤백 옵션을 선택할 때 설정됩니다. 이는 기본 설정 동작입니다.
- `ForceApplyCapacityChange` - 이 값은 용량 변경 강제 수행 옵션을 선택할 때 설정됩니다.

다음과 같이 describe-db-clusters Aurora Serverless v1 명령을 사용하여 기존 AWS CLI DB 클러스터에서 이 속성 값을 가져올 수 있습니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

Windows의 경우:

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

예를 들어 다음은 미국 서부(캘리포니아 북부) 리전의 `west-coast-sles`(이)라는 Aurora Serverless v1 DB 클러스터에 대한 쿼리 및 응답을 보여줍니다.

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

응답에서 알 수 있듯이, 이 Aurora Serverless v1 DB 클러스터는 기본 설정을 사용합니다.

자세한 내용은 Aurora Serverless v1 DB 클러스터 생성 섹션을 참조하세요. Aurora Serverless v1를 생성한 후 시간 초과 작업 및 기타 용량 설정을 언제든지 수정할 수 있습니다. 자세한 방법은 Aurora Serverless v1 DB 클러스터 수정 단원을 참조하십시오.

Aurora Serverless v1 일시 중지 및 다시 시작

활동 없이 지정된 시간이 경과하면 Aurora Serverless v1 DB 클러스터를 일시 중지하도록 선택할 수 있습니다. DB 클러스터를 일시 중지하기 전에 활동 없이 경과하는 시간을 지정합니다. 이 옵션을 선택하면 활동이 없는 기본 시간은 5분이지만 이 값을 변경할 수 있습니다. 이는 선택적 설정입니다.

DB 클러스터가 일시 중지되면 컴퓨팅 또는 메모리 활동이 발생하지 않고 스토리지에 대한 요금만 청구됩니다. Aurora Serverless v1 DB 클러스터가 일시 중지되었을 때 데이터베이스 연결이 요청되면 DB 클러스터가 자동으로 작동을 재개해 연결 요청을 처리합니다.

DB 클러스터가 활동을 재개하면 Aurora이 클러스터를 일시 중지했을 때와 동일한 용량을 갖게 됩니다. ACU의 수는 클러스터를 일시 중지하기 전에 Aurora이 클러스터를 확장하거나 축소한 정도에 따라 달라집니다.

> **Note**
>
> DB 클러스터가 8일 이상 일시 중지된 경우 스냅샷을 사용하여 DB 클러스터를 백업할 수 있습니다. 이 경우 Aurora은 연결 요청이 있을 때 스냅샷에서 DB 클러스터를 복원합니다.

Aurora Serverless v1에 대한 최대 데이터베이스 연결 수 결정

다음은 MySQL 5.7과 호환되는 Aurora Serverless v1 DB 클러스터에 대한 예입니다. 액세스를 구성한 경우 MySQL 클라이언트 또는 쿼리 편집기를 사용할 수 있습니다. 자세한 내용은 [쿼리 편집기에서 쿼리 실행](#) 섹션을 참조하세요.

최대 데이터베이스 연결 수를 확인하려면

1. AWS CLI를 사용하여 Aurora Serverless v1 DB 클러스터의 용량 범위를 확인합니다.

```
aws rds describe-db-clusters \
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

   그 결과 용량 범위가 1~4 ACU임을 알 수 있습니다.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. 다음 SQL 쿼리를 실행하여 최대 연결 수를 확인합니다.

```
select @@max_connections;
```

결과로 클러스터의 최소 용량인 1 ACU가 표시됩니다.

```
@@max_connections
90
```

3. 클러스터를 8~32 ACU로 확장합니다.

   조정에 대한 자세한 내용은 Aurora Serverless v1 DB 클러스터 수정 섹션을 참조하세요.

4. 용량 범위를 확인합니다.

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. 최대 연결 수를 확인합니다.

```
select @@max_connections;
```

결과로 클러스터의 최소 용량인 8 ACU가 표시됩니다.

```
@@max_connections
1000
```

6. 클러스터를 가능한 최대 범위인 256~256 ACU로 확장합니다.
7. 용량 범위를 확인합니다.

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. 최대 연결 수를 확인합니다.

```
select @@max_connections;
```

결과로 256 ACU가 표시됩니다.

```
@@max_connections
6000
```

> **Note**
>
> max_connections 값은 ACU 수에 따라 선형적으로 확장되지 않습니다.

9. 클러스터를 1~4 ACU로 다시 축소합니다.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

이번에는 max_connections 값이 4 ACU로 나타납니다.

```
@@max_connections
270
```

10. 클러스터를 2 ACU로 축소합니다.

```
@@max_connections
180
```

일정한 유휴 시간이 지나면 일시 중지하도록 클러스터를 구성한 경우 0 ACU로 축소됩니다. 단, max_connections는 1 ACU 값 이하로 줄어들지 않습니다.

```
@@max_connections
90
```

Aurora Serverless v1 파라미터 그룹

Aurora Serverless v1 DB 클러스터를 생성할 때 특정 Aurora DB 엔진과 관련 DB 클러스터 파라미터 그룹을 선택합니다. 프로비저닝된 Aurora DB 클러스터와 달리, Aurora Serverless v1 DB 클러스터에는 별도의 DB 파라미터 그룹이 없고 하나의 DB 클러스터 파라미터 그룹으로만 구성된 단일 읽기/쓰기 DB 인스턴스가 있습니다.— Auto Scaling 중에 Aurora Serverless v1는 증가 또는 감소된 용량에 가장 적합하도록 클러스터의 파라미터를 변경할 수 있어야 합니다. 즉, Aurora Serverless v1 DB 클러스터의 경우, 특정 DB 엔진 유형에 대한 파라미터 변경 사항 중 일부가 적용되지 않을 수 있습니다.

예를 들어 Aurora PostgreSQL–기반 Aurora Serverless v1 DB 클러스터는 프로비저닝된 Aurora PostgreSQL DB 클러스터에서 쿼리 계획 관리에 사용되는 `apg_plan_mgmt.capture_plan_baselines` 및 기타 파라미터를 사용할 수 없습니다.

[describe-engine-default-cluster-parameters](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-engine-default-cluster-parameters.html) CLI 명령을 사용하고 AWS 리전을 쿼리하여 다양한 Aurora DB 엔진의 기본 파라미터 그룹의 기본값 목록을 가져올 수 있습니다. `--db-parameter-group-family` 옵션에 사용할 수 있는 값은 다음과 같습니다.

Aurora MySQL 버전 2	`aurora-mysql5.7`
Aurora PostgreSQL 버전 11	`aurora-postgresql11`
Aurora PostgreSQL 버전 13	`aurora-postgresql13`

AWS CLI 명령을 사용하기 전에 AWS 액세스 키 ID 및 AWS 보안 액세스 키를 사용하여 AWS 리전를 구성하고 AWS CLI를 설정하는 것이 좋습니다. CLI 구성에 리전을 설정하면 명령을 실행할 때 `--region` 파라미터를 입력할 필요가 없습니다. AWS CLI 구성에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서에서 구성 기본 사항을 참조하세요.

다음 예시에서는 Aurora MySQL 버전 2의 기본 DB 클러스터 그룹에서 파라미터 목록을 가져옵니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

Windows의 경우:

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

Aurora Serverless v1의 파라미터 값 수정

Amazon Aurora의 파라미터 그룹에서 설명한 것처럼, 유형(DB 클러스터 파라미터 그룹, DB 파라미터 그룹)에 관계없이 기본 파라미터 그룹에서 값을 직접 변경할 수는 없습니다. 대신, 기본 DB 클러스터 파라미터 그룹을 기반으로 Aurora DB 엔진에 사용할 사용자 지정 파라미터 그룹을 생성하고 해당 파라미터 그룹에서 필요에 따라 설정을 변경합니다. 예를 들어 다음과 같이 Amazon CloudWatch에 쿼리를 로깅하거나 DB 엔진별 로그 업로드하도록 Aurora Serverless v1 DB 클러스터의 일부 설정을 변경할 수 있습니다.

사용자 지정 DB 클러스터 파라미터 그룹을 생성하려면

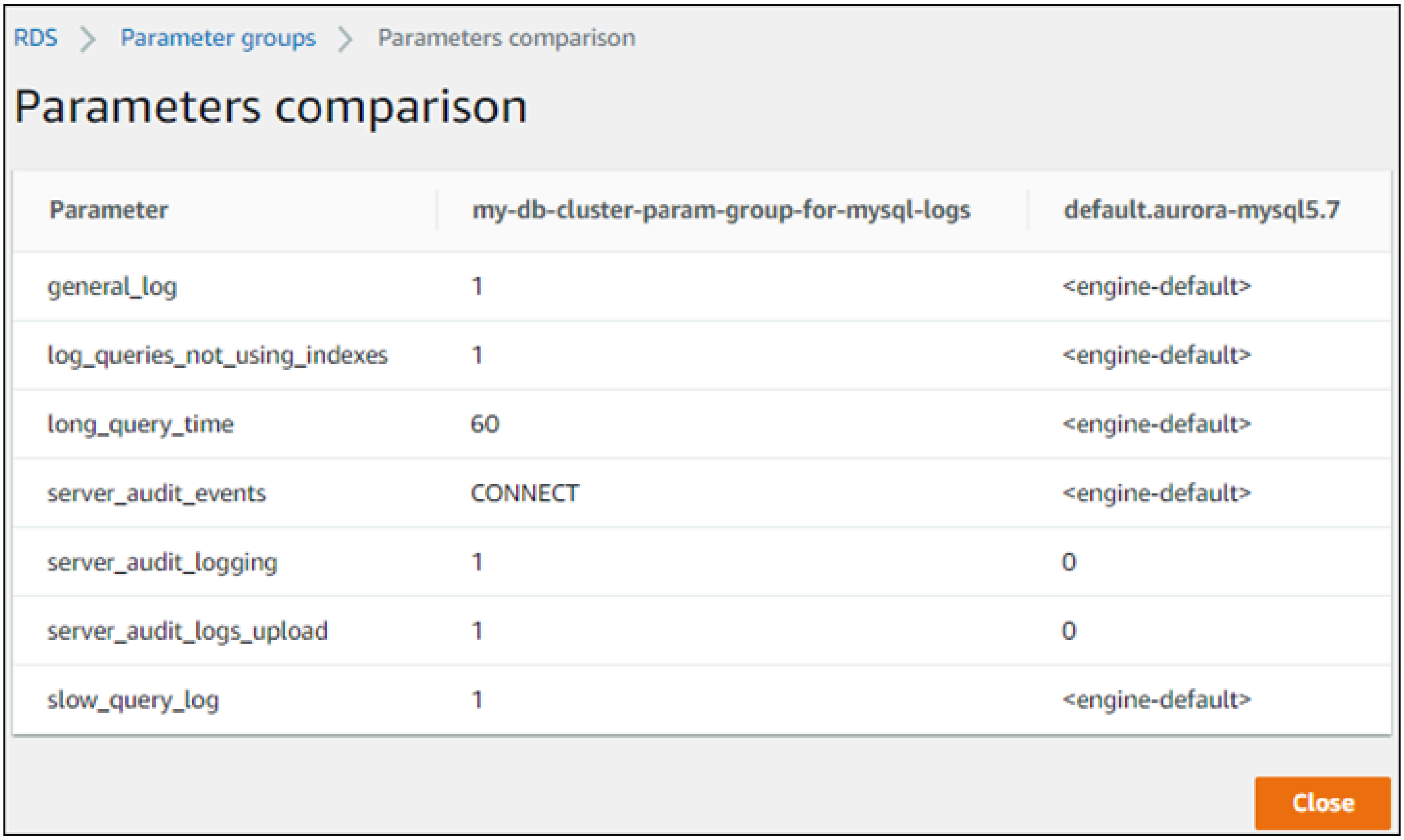
1. AWS Management Console에 로그인하고 ttps://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 파라미터 그룹을 선택합니다.
3. [파라미터 그룹 생성(Create parameter group)] 선택하여 파라미터 그룹 세부 정보 창을 엽니다.
4. DB 클러스터에 사용할 DB 엔진에 적합한 기본 Aurora Serverless v1 DB 클러스터 그룹을 선택합니다. 다음과 같은 옵션을 선택할 수 있습니다.

   a. [파라미터 그룹 패밀리(Parameter group family)]에서, 선택한 DB 엔진에 적합한 패밀리를 선택합니다. 선택 항목의 이름에 `aurora-` 접두어가 있어야 합니다.
   b. [Type]에서 [DB Cluster Parameter Group]을 선택합니다.
   c. [그룹 이름(Group name)] 및 [설명(Description)] 에 Aurora Serverless v1 DB 클러스터 및 해당 파라미터로 작업해야 하는 사용자가 알기 쉬운 이름을 입력합니다.
   d. 생성(Create)을 선택합니다.

사용자 지정 DB 클러스터 파라미터 그룹이 AWS 리전에서 사용할 수 있는 파라미터 그룹 목록에 추가됩니다. 새 DB 클러스터를 생성할 때 사용자 지정 Aurora Serverless v1 DB 클러스터 파라미터 그룹을 사용할 수 있습니다. 사용자 지정 Aurora Serverless v1 DB 클러스터 파라미터 그룹을 사용하도록 기

존 DB 클러스터를 수정할 수도 있습니다. Aurora Serverless v1 DB 클러스터가 사용자 지정 DB 클러스터 파라미터 그룹을 사용하기 시작하면 AWS Management Console 또는 AWS CLI를 사용하여 동적 파라미터의 값을 변경할 수 있습니다.

다음 스크린샷과 같이 콘솔을 사용하여 사용자 지정 DB 클러스터 파라미터 그룹의 값을 기본 DB 클러스터 파라미터 그룹의 값과 나란히 비교할 수도 있습니다.



Parameter	my-db-cluster-param-group-for-mysql-logs	default.aurora-mysql5.7
general_log	1	<engine-default>
log_queries_not_using_indexes	1	<engine-default>
long_query_time	60	<engine-default>
server_audit_events	CONNECT	<engine-default>
server_audit_logging	1	0
server_audit_logs_upload	1	0
slow_query_log	1	<engine-default>

활성 DB 클러스터의 파라미터 값을 변경하면 Aurora Serverless v1는 파라미터 변경 사항을 적용하기 위해 원활한 크기 조정을 시작합니다. Aurora Serverless v1 DB 클러스터가 일시 중지됨 상태인 경우 변경 작업을 수행할 수 있도록 클러스터가 다시 시작되어 크기 조정을 시작합니다. 파라미터 그룹의 크기 조정 작업은 항상 용량 변경을 강제로 적용하므로 크기 조정 기간 동안 조정점을 찾을 수 없는 경우 파라미터를 수정하면 연결이 끊어질 수 있습니다.

Aurora Serverless v1의 로깅

Aurora Serverless v1의 오류 로그는 기본적으로 활성화되고 Amazon CloudWatch에 자동으로 업로드됩니다. Aurora 데이터베이스 엔진별 로그를 CloudWatch에 DB 클러스터를 업로드하도록 할 수도 있습니다. 이를 위해 사용자 지정 DB 클러스터 파라미터 그룹에서 구성 파라미터를 활성화합니다. 그런 다음 Aurora Serverless v1 DB 클러스터는 사용 가능한 모든 로그를 Amazon CloudWatch에 업로드합

니다. 이 시점에서 CloudWatch를 통해 로그 데이터에 대한 분석을 수행할 수 있으며 경보를 만들고 지표를 볼 수 있습니다.

Aurora MySQL의 경우 다음 표에 나와 있는 로그를 활성화할 수 있습니다. 로그가 활성화되면 Aurora Serverless v1 DB 클러스터에서 Amazon CloudWatch로 자동 업로드됩니다.

Aurora MySQL 로그	설명
`general_log`	일반 로그를 생성합니다. 활성화하려면 1로 설정합니다. 기본값은 비활성화(0)입니다.
`log_queries_not_using_indexes`	인덱스를 사용하지 않는 느린 쿼리 로그에 모든 쿼리를 로깅합니다. 기본값은 비활성화(0)입니다. 이 로그를 활성화하려면 1로 설정합니다.
`long_query_time`	빠른 실행 쿼리가 느린 쿼리 로그에 로깅되지 않도록 합니다. 0에서 3,1536,000 사이의 부동 소수점 값으로 설정할 수 있습니다. 기본값은 0(비활성화)입니다.
`server_audit_events`	로그에 캡처할 이벤트 목록입니다. 지원되는 값은 `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` 및 `TABLE`입니다.
`server_audit_logging`	서버 감사 로깅을 활성화하려면 1로 설정합니다. 이 옵션을 설정하면 `server_audit_events` 매개 변수에 감사 이벤트를 CloudWatch 나열하여 보낼 감사 이벤트를 지정할 수 있습니다.
`slow_query_log`	느린 쿼리 로그를 생성합니다. 느린 쿼리 로그를 활성화하려면 1로 설정합니다. 기본값은 비활성화(0)입니다.

자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용](#) 섹션을 참조하세요.

Aurora PostgreSQL의 경우 다음 표에 나와 있는 로그를 활성화할 수 있습니다. 로그가 활성화되면 일반적인 오류 로그와 함께 Aurora Serverless v1 DB 클러스터에서 Amazon CloudWatch로 자동 업로드됩니다.

Aurora PostgreSQL 로그	설명
`log_connections`	기본적으로 켜지며 변경할 수 없습니다. 모든 새 클라이언트 연결에 대한 세부 정보를 로깅합니다.
`log_disconnections`	기본적으로 켜지며 변경할 수 없습니다. 모든 클라이언트 연결 해제를 로깅합니다.
`log_hostname`	기본적으로 비활성화 상태이며 값을 변경할 수 없습니다. 호스트 이름은 로깅되지 않습니다.
`log_lock_waits`	기본값은 0(비활성화)입니다. 잠금 대기를 로깅하려면 1로 설정합니다.
`log_min_duration_statement`	로깅하기 전에 문을 실행할 최소 시간(밀리초)입니다.
`log_min_messages`	기록되는 메시지 수준을 설정합니다. 지원되는 값은 `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`입니다. 성능 데이터를 `postgres` 로그에 로깅하려면 이 값을 `debug1`로 설정합니다.
`log_temp_files`	지정된 KB(킬로바이트)를 초과하는 임시 파일 사용을 로깅합니다.
`log_statement`	로깅되는 특정 SQL 문을 제어합니다. 지원되는 값은 `none`, `ddl`, `mod` 및 `all`입니다. 기본값은 `none`.

Aurora Serverless v1 DB 클러스터에 대해 Aurora MySQL 또는 Aurora PostgreSQL 로그를 활성화하고 나면, CloudWatch에서 로그를 볼 수 있습니다.

Amazon CloudWatch를 사용하여 Aurora Serverless v1 로그 보기

Aurora Serverless v1은 사용자 지정 DB 클러스터 파라미터 그룹에서 활성화한 모든 로그를 Amazon CloudWatch에 자동으로 업로드('게시')합니다. 따라서 로그 유형을 선택하거나 지정할 필요가 없습니다. 로그 구성 파라미터를 활성화하는 즉시 로그 업로드가 시작됩니다. 나중에 로그 파라미터를 비활성화하면 추가 업로드가 중지됩니다. 하지만 CloudWatch에 이미 게시된 모든 로그는 삭제할 때까지 유지됩니다.

Aurora MySQL 로그에 CloudWatch을 사용하는 방법에 대한 자세한 내용은 Amazon CloudWatch에서 로그 이벤트 모니터링 섹션을 참조하세요.

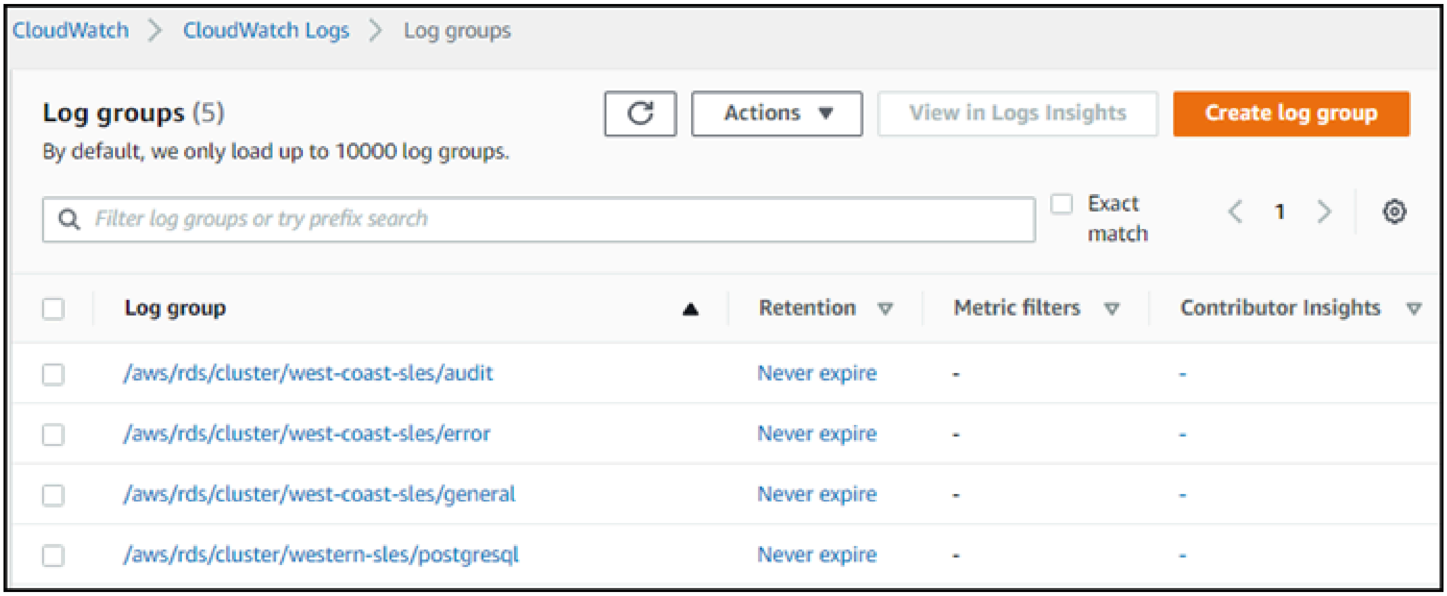
CloudWatch 및 Aurora PostgreSQL에 대한 자세한 내용은 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 단원을 참조하십시오.

Aurora Serverless v1 DB 클러스터에 대한 로그를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. AWS 리전를 선택합니다.
3. 로그 그룹을 선택합니다.
4. 목록에서 Aurora Serverless v1 DB 클러스터 로그를 선택합니다. 오류 로그의 경우 이름 지정 패턴은 다음과 같습니다.

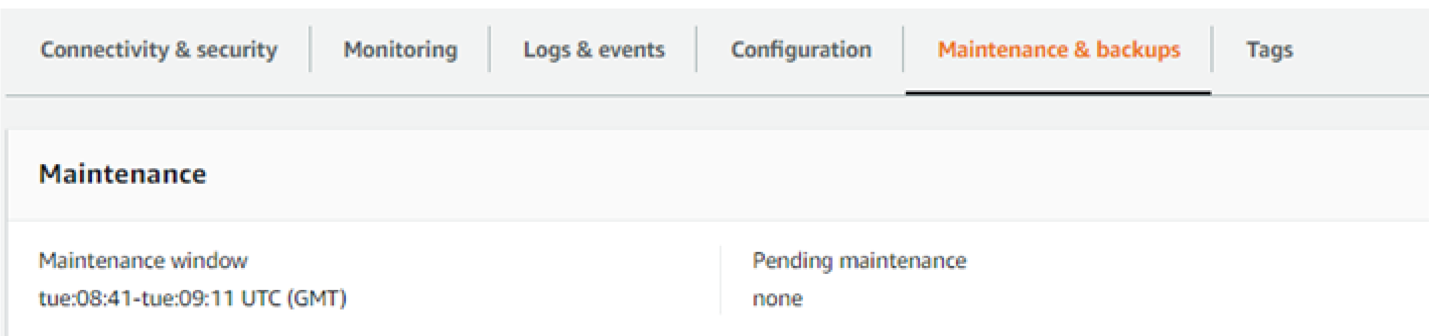
```
/aws/rds/cluster/cluster-name/error
```

예를 들어 다음 스크린샷에는 Aurora Serverless v1라는 Aurora PostgreSQL `western-sles` DB 클러스터에 대해 게시된 로그의 목록이 나와 있습니다. 또한 Aurora MySQL Aurora Serverless v1 DB 클러스터 `west-coast-sles`에 대한 몇 가지 목록이 나와 있습니다. 관심 있는 로그를 선택하여 해당 콘텐츠를 탐색할 수 있습니다.



Aurora Serverless v1 및 유지 관리

Aurora Serverless v1 DB 클러스터에 대한 유지 관리(예: 최신 기능, 수정, 보안 업데이트 적용)가 자동으로 수행됩니다. Aurora Serverless v1에는 유지 관리 기간이 있으며 Aurora Serverless v1 DB 클러스터에 대한 유지 관리 및 백업의 AWS Management Console에서 확인할 수 있습니다. 다음 그림과 같이 유지 관리가 수행될 수 있는 날짜 및 시간과 Aurora Serverless v1 DB 클러스터에 대해 대기 중인 유지 관리 작업을 확인할 수 있습니다.



Aurora Serverless v1 DB 클러스터를 생성할 때 유지 관리 기간을 설정하고 나중에 기간을 수정할 수 있습니다. 자세한 내용은 [기본 DB 클러스터 유지 관리 기간 조정](#) 섹션을 참조하세요.

유지 관리 기간은 예약된 메이저 버전 업그레이드에 사용됩니다. 마이너 버전 업그레이드 및 패치는 규모 조정 중에 즉시 적용됩니다. 규모 조정은 다음과 같은 `TimeoutAction` 설정에 따라 이루어집니다.

- `ForceApplyCapacityChange` - 변경 사항이 즉시 적용됩니다.
- `RollbackCapacityChange` - Aurora가 첫 번째 패치를 시도한 지 3일 후에 클러스터를 강제로 업데이트합니다.

적절한 조정점 없이 강제로 변경되는 여타의 경우와 마찬가지로, 이로 인해 워크로드가 중단될 수 있습니다.

Aurora Serverless v1는 가능하면 운영 중단 없이 유지 관리 작업을 수행합니다. 유지 관리가 필요한 경우 Aurora Serverless v1 DB 클러스터는 필요한 작업을 처리할 수 있도록 용량을 확장합니다. Aurora Serverless v1은 크기 조정 전에 크기 조정 시점을 찾습니다. 필요한 경우 최대 3일 동안 이를 수행합니다.

결국 Aurora Serverless v1가 조정점을 찾지 못하면 클러스터 이벤트를 생성합니다. 이 이벤트는 대기 중인 유지 관리 작업과 유지 관리를 수행하기 위해 확장해야 하는 필요성을 알려줍니다. 알림에는 Aurora Serverless v1가 DB 클러스터를 강제로 확장할 수 있는 날짜가 포함됩니다.

자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.

Aurora Serverless v1 및 장애 조치

Aurora Serverless v1 DB 클러스터의 DB 인스턴스를 사용할 수 없게 되거나 가용 영역(AZ)에 장애가 발생하면 Aurora가 다른 AZ에서 DB 인스턴스를 재생성합니다. 그러나 Aurora Serverless v1 클러스터는 다중 AZ 클러스터가 아닙니다. 이는 단일 AZ의 단일 DB 인스턴스로 구성되기 때문입니다. 이러한 장애 조치 메커니즘은 Aurora 프로비저닝된 클러스터보다 시간이 오래 걸립니다. Aurora Serverless v1 장애 조치 시간은 해당 AWS 리전 내 다른 AZ의 수요 및 용량 가용성에 따라 다르기 때문에 정의되어 있지 않습니다.

Aurora는 컴퓨팅 용량과 스토리지를 분리하기 때문에 클러스터의 스토리지 볼륨은 다중 AZ로 분산됩니다. 따라서 시스템 중단으로 DB 인스턴스 또는 연결된 AZ가 영향을 받더라도 데이터는 계속해서 사용할 수 있습니다.

Aurora Serverless v1 및 스냅샷

Aurora Serverless v1 클러스터의 클러스터 볼륨은 항상 암호화됩니다. 암호화 키를 선택할 수 있지만 암호화를 비활성화할 수는 없습니다. Aurora Serverless v1 클러스터의 스냅샷을 복사하거나 공유하려면 사용자 고유의 AWS KMS key을(를) 사용해 스냅샷을 암호화합니다. 자세한 내용은 DB 클러스터 스냅샷 복사 섹션을 참조하세요. 암호화 및 Amazon Aurora에 대한 자세한 내용은 Amazon Aurora DB 클러스터 암호화 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터 생성

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

다음 프로시저에서는 스키마 개체 또는 데이터 없이 Aurora Serverless v1 클러스터를 생성합니다. 기존 프로비저닝 또는 Aurora Serverless v1 클러스터의 복제본인 Aurora Serverless v1 클러스터를 생성하려는 경우 스냅샷 복원 또는 복제 작업을 대신 수행할 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 및 Aurora DB 클러스터에 대한 볼륨 복제 섹션을 참조하세요. 기존 프로비저닝된 클러스터를 Aurora Serverless v1으로 변환할 수 없습니다. 또한 기존 Aurora Serverless v1 클러스터를 다시 프로비저닝된 클러스터로 변환할 수 없습니다.

Aurora Serverless v1 DB 클러스터를 생성하는 경우 해당 클러스터에 대한 최소 및 최대 용량을 설정할 수 있습니다. 용량 단위는 특정 컴퓨팅 및 메모리 구성과 동일합니다. Aurora Serverless v1은 CPU 사용률, 연결 및 사용 가능한 메모리에 대한 임계값에 대한 조정 규칙을 생성하고 애플리케이션에 필요한 만큼 용량 단위 범위로 원활하게 확장합니다. 자세한 정보는 Aurora Serverless v1 아키텍처을 참조하세요.

Aurora Serverless v1 DB 클러스터에 대해 다음과 같은 특정 값을 설정할 수 있습니다.

- 최소 Aurora 용량 단위 - Aurora Serverless v1은 이 용량 단위까지 용량을 줄일 수 있습니다.
- 최대 Aurora 용량 단위 - Aurora Serverless v1은 이 용량 단위까지 용량을 늘릴 수 있습니다.

다음과 같은 선택적 조정 구성 옵션을 선택할 수도 있습니다.

- 제한 시간에 도달하면 용량을 지정된 값으로 강제 크기 조정 - 지정된 제한 시간 이전에 크기 조정 포인트를 찾을 수 없더라도 Aurora Serverless v1이 Aurora Serverless v1을 강제 크기 조정하도록 하려면 이 설정을 선택할 수 있습니다. 조정점을 찾을 수 없는 경우 Aurora Serverless v1가 용량 변경을 취소하도록 하려면 이 설정을 선택하지 않습니다. 자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.

- 다음 시간(분) 동안 활동이 없는 경우 컴퓨팅 용량 일시 중지 - 지정한 시간 동안 DB 클러스터에 활동이 없을 때 Aurora Serverless v1을 0으로 크기 조정하려는 경우 이 설정을 선택할 수 있습니다. 이 설정을 활성화하면 Aurora Serverless v1 DB 클러스터가 처리를 자동으로 다시 시작하고 데이터베이스 트래픽이 다시 시작될 때 워크로드를 처리하는 데 필요한 용량으로 조정합니다. 자세한 내용은 Aurora Serverless v1 일시 중지 및 다시 시작 단원을 참조하십시오.

Aurora Serverless v1 DB 클러스터를 생성하려면 먼저 AWS 계정이 필요합니다. 또한 Amazon Aurora 작업을 위해 설정 작업을 완료해야 합니다. 자세한 내용은 Amazon Aurora 환경 설정 섹션을 참조하세요. 또한 Aurora DB 클러스터를 생성하기 위한 다른 예비 단계도 완료해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

Aurora Serverless v1은 특정 AWS 리전와 특정 Aurora MySQL 및 Aurora PostgreSQL 버전에서만 사용할 수 있습니다. 자세한 내용은 Aurora Serverless v1 섹션을 참조하세요.

> **Note**
>
> Aurora Serverless v1 클러스터의 클러스터 볼륨은 항상 암호화됩니다. Aurora Serverless v1 DB 클러스터를 생성할 때 암호화를 해제할 수는 없지만 자체 암호화 키를 사용하도록 선택할 수 있습니다. Aurora Serverless v2를 사용하면 클러스터 볼륨을 암호화할지 여부를 선택할 수 있습니다.

AWS CLI 또는 RDS API를 사용하여 Aurora Serverless v1 DB 클러스터를 만들 수 있습니다.

> **Note**
>
> 클러스터를 만들려고 할 때 다음과 같은 오류 메시지가 나타나면 계정에 추가 권한이 필요합니다.
>
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
>
> 자세한 내용은 Amazon Aurora에 서비스 연결 역할 사용를 참조하십시오.

DB 클러스터의 Aurora Serverless v1 DB 인스턴스에 직접 연결할 수는 없습니다. Aurora Serverless v1 DB 클러스터에 연결하려면 데이터베이스 엔드포인트를 사용합니다. Aurora Serverless v1 DB 클러스터의 엔드포인트는 AWS Management Console에서 클러스터의 [연결 및 보안(Connectivity & security)] 탭에서 확인할 수 있습니다. 자세한 내용은 Amazon Aurora DB 클러스터에 연결 섹션을 참조하세요.

AWS CLI

Aurora Serverless v1를 사용하여 새 AWS CLI DB 클러스터를 생성하려면 [create-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/create-db-cluster.html) 명령을 실행하고 `serverless` 옵션에 `--engine-mode`를 지정합니다.

최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `--scaling-configuration` 옵션을 선택적으로 지정할 수 있습니다.

다음 명령 예제에서는 `--engine-mode` 옵션을 `serverless`로 설정하여 새로운 Serverless DB 클러스터를 생성합니다. 또한 `--scaling-configuration` 옵션 값도 지정합니다.

Aurora MySQL 예

다음 명령을 실행하면 새로운 Aurora MySQL 호환 서버리스 DB 클러스터가 생성됩니다. Aurora MySQL에서 유효한 용량 값은 `1`, `2`, `4`, `8`, `16`, `32`, `64`, `128` 및 `256`입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Aurora PostgreSQL 예

다음 명령을 실행하면 새로운 PostgreSQL 13.9 호환 서버리스 DB 클러스터가 생성됩니다. Aurora PostgreSQL에 유효한 용량 값은 `2`, `4`, `8`, `16`, `32`, `64`, `192` 및 `384`입니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

RDS API

RDS API를 사용하여 새 Aurora Serverless v1 DB 클러스터를 생성하려면 CreateDBCluster 작업을 실행하고 `serverless` 파라미터에 `EngineMode`를 지정합니다.

최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `ScalingConfiguration` 파라미터를 선택적으로 지정할 수 있습니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

Aurora Serverless v1 DB 클러스터 복원

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Aurora Serverless v1 또는 RDS API를 사용하여 프로비저닝된 DB 클러스터 스냅샷을 복원하는 경우 AWS CLI DB 클러스터를 구성할 수 있습니다.

Aurora Serverless v1 DB 클러스터로 스냅샷을 복원할 때 다음과 같은 특정 값을 설정할 수 있습니다.

- 최소 Aurora 용량 단위 - Aurora Serverless v1은 이 용량 단위까지 용량을 줄일 수 있습니다.
- 최대 Aurora 용량 단위 - Aurora Serverless v1은 이 용량 단위까지 용량을 늘릴 수 있습니다.
- 제한 시간 조치 – 조정점을 찾을 수 없기 때문에 용량 수정 시간이 초과될 때 수행할 작업입니다. Aurora Serverless v1 [용량을 지정된 값으로 강제 조정 옵션을 설정하면(Force scaling the capacity to the specified values)] 옵션을 설정하면 DB 클러스터가 해당 DB 클러스터를 새 용량으로 강제 설정할 수 있습니다. 또는 옵션을 선택하지 않은 경우 용량 변경을 롤백하여 취소할 수 있습니다. 자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.
- Pause after inactivity(비활성 후 일시 중지) – 처리 용량이 0이 될 때까지 조정하기 위해 데이터베이스 트래픽이 없는 시간입니다. 데이터베이스 트래픽이 다시 시작되면 Aurora가 처리 용량을 자동으로 재개하고 조정하여 트래픽을 처리합니다.

스냅샷에서 DB 클러스터를 복원하는 일반적인 방법은 DB 클러스터 스냅샷에서 복원 단원을 참조하십시오.

AWS CLI

Aurora Serverless, AWS Management Console 또는 RDS API를 사용하여 프로비저닝된 DB 클러스터 스냅샷을 복원하는 경우 AWS CLI DB 클러스터를 구성할 수 있습니다.

Aurora Serverless DB 클러스터로 스냅샷을 복원할 때 다음과 같은 특정 값을 설정할 수 있습니다.

- 최소 Aurora 용량 단위 - Aurora Serverless은 이 용량 단위까지 용량을 줄일 수 있습니다.
- 최대 Aurora 용량 단위 - Aurora Serverless은 이 용량 단위까지 용량을 늘릴 수 있습니다.
- 제한 시간 조치 – 조정점을 찾을 수 없기 때문에 용량 수정 시간이 초과될 때 수행할 작업입니다. Aurora Serverless v1 [용량을 지정된 값으로 강제 조정 옵션을 설정하면(Force scaling the capacity to the specified values)] 옵션을 설정하면 DB 클러스터가 해당 DB 클러스터를 새 용량으로 강제 설정할 수 있습니다. 또는 옵션을 선택하지 않은 경우 용량 변경을 롤백하여 취소할 수 있습니다. 자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.
- Pause after inactivity(비활성 후 일시 중지) – 처리 용량이 0이 될 때까지 조정하기 위해 데이터베이스 트래픽이 없는 시간입니다. 데이터베이스 트래픽이 다시 시작되면 Aurora가 처리 용량을 자동으로 재개하고 조정하여 트래픽을 처리합니다.

Note

DB 클러스터 스냅샷의 버전은 Aurora Serverless v1과 호환되어야 합니다. 지원되는 버전의 목록은 Aurora Serverless v1 섹션을 참조하세요.

MySQL 5.7 호환성이 있는 Aurora Serverless v1 클러스터로 스냅샷을 복원하려면 다음과 같은 추가 파라미터를 포함합니다.

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

`--engine` 및 `--engine-version` 파라미터를 사용하면 MySQL 5.7 호환 Aurora 또는 Aurora Serverless v1 스냅샷에서 MySQL 5.6 호환 Aurora Serverless v1 클러스터를 생성할 수 있습니다. 다음 예제에서는 *`mydbclustersnapshot`*이라는 MySQL 5.6 호환 클러스터에서 *`mynewdbcluster`*라는 MySQL 5.7 호환 Aurora Serverless v1 클러스터로 스냅샷을 복원합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `--scaling-configuration` 옵션을 선택적으로 지정할 수 있습니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

다음 예제에서는 *mydbclustersnapshot*이라는 이전에 생성된 DB 클러스터 스냅샷에서 *mynewdbcluster*라는 새로운 DB 클러스터로 복원합니다. 워크로드를 처리하기 위해 필요에 따라 새 `--scaling-configuration` DB 클러스터를 8개 ACU에서 64개 ACU(Aurora 용량 단위)로 확장할 수 있도록 Aurora Serverless v1을 설정합니다. 처리가 완료되고 지원할 연결 없이 1000초가 지나면 연결 요청 시 다시 시작하라는 메시지가 표시될 때까지 클러스터가 종료됩니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

RDS API

RDS API를 사용하여 DB 클러스터에서 복원할 때 Aurora Serverless v1 DB 클러스터를 구성하려면 RestoreDBClusterFromSnapshot 작업을 실행하고 `serverless` 파라미터에 `EngineMode`를 지정합니다.

최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `ScalingConfiguration` 파라미터를 선택적으로 지정할 수 있습니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

Aurora Serverless v1 DB 클러스터 수정

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터를 구성한 후 AWS Management Console, AWS CLI 또는 RDS API를 사용하여 특정 속성을 수정할 수 있습니다. 수정할 수 있는 대부분의 속성은 다른 종류의 Aurora 클러스터와 동일합니다.

Aurora Serverless v1와 가장 연관성 있는 변경 사항은 다음과 같습니다.

- 조정 구성 수정
- 메이저 버전 업그레이드
- Aurora Serverless v1에서 프로비저닝으로 변환

Aurora Serverless v1 DB 클러스터의 조정 구성 수정

DB 클러스터에 대해 최소 및 최대 용량을 설정할 수 있습니다. 각 용량 단위는 특정 컴퓨팅 및 메모리 구성과 동일합니다. Aurora Serverless가 CPU 사용률, 연결 및 가용 메모리 임계값에 대한 조정 규칙을 자동으로 생성합니다. 또한 활동이 없을 때 Aurora Serverless가 데이터베이스를 일시 중지하고 활동이 다시 시작되면 다시 시작하도록 할지 여부를 설정할 수도 있습니다.

크기 조정 구성에 대해 다음과 같은 특정 값을 설정할 수 있습니다.

- 최소 Aurora 용량 단위 - Aurora Serverless는 이 용량 단위까지 용량을 줄일 수 있습니다.
- 최대 Aurora 용량 단위 - Aurora Serverless는 이 용량 단위까지 용량을 늘릴 수 있습니다.
- 자동 크기 조정 시간 초과 및 작업 - 이 섹션에서는 Aurora Serverless(이)가 제한 시간 이전에 크기 조정점을 찾기 위해 대기하는 시간을 지정합니다. 또한 크기 조정점을 찾을 수 없기 때문에 용량 수정 시간이 초과될 때 수행할 작업을 지정합니다. Aurora는 용량 변경을 통해 용량을 지정된 값으로

최대한 빨리 설정할 수 있습니다. 또는 용량 변경을 롤백하여 변경을 취소할 수 있습니다. 자세한 내용은 용량 변경을 위한 제한 시간 조치 섹션을 참조하세요.

- 비활성 후 일시 중지 - 선택 사항인 클러스터가 유휴 상태일 때 용량을 0ACU로 조정 설정을 사용하여 데이터베이스가 비활성 상태일 때 처리 용량을 0으로 조정합니다. 데이터베이스 트래픽이 다시 시작되면 Aurora가 처리 용량을 자동으로 재개하고 조정하여 트래픽을 처리합니다.

> **Note**
>
> Aurora Serverless DB 클러스터의 용량 범위를 수정하면 변경 사항을 즉시 적용하는 옵션과 예정된 다음 유지 관리 기간에 적용하는 옵션 중 무엇을 선택했든지 변경 사항이 즉시 적용됩니다.

콘솔

AWS Management Console을 사용하여 Aurora DB 클러스터의 크기 조정 구성을 수정할 수 있습니다.

Aurora Serverless v1 DB 클러스터를 수정하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 수정할 Aurora Serverless v1 DB 클러스터를 선택합니다.
4. 작업에서 클러스터 수정을 선택합니다.
5. 용량 설정 섹션에서 조정 구성을 수정합니다.
6. Continue(계속)를 선택합니다.
7. DB 클러스터 수정 페이지에서 수정 사항을 검토한 후 다음 중 하나를 선택하여 적용합니다.
8. 클러스터 수정을 선택합니다.

AWS CLI

AWS CLI를 사용하여 Aurora Serverless v1 DB 클러스터의 조정 구성을 수정하려면 modify-db-cluster AWS CLI 명령을 실행합니다. 최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `--scaling-configuration` 옵션을 지정합니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

이 예에서는 *sample-cluster*라는 Aurora Serverless v1 DB 클러스터의 조정 구성을 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

RDS API

ModifyDBCluster API 작업으로 Aurora DB 클러스터의 조정 구성을 수정할 수 있습니다. 최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 중지를 구성하도록 `ScalingConfiguration` 파라미터를 지정합니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

Aurora Serverless v1 DB 클러스터의 메이저 버전 업그레이드

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

PostgreSQL 11과 호환되는 Aurora Serverless v1 DB 클러스터를 해당 PostgreSQL 13 호환 버전으로 메이저 버전 업그레이드할 수 있습니다.

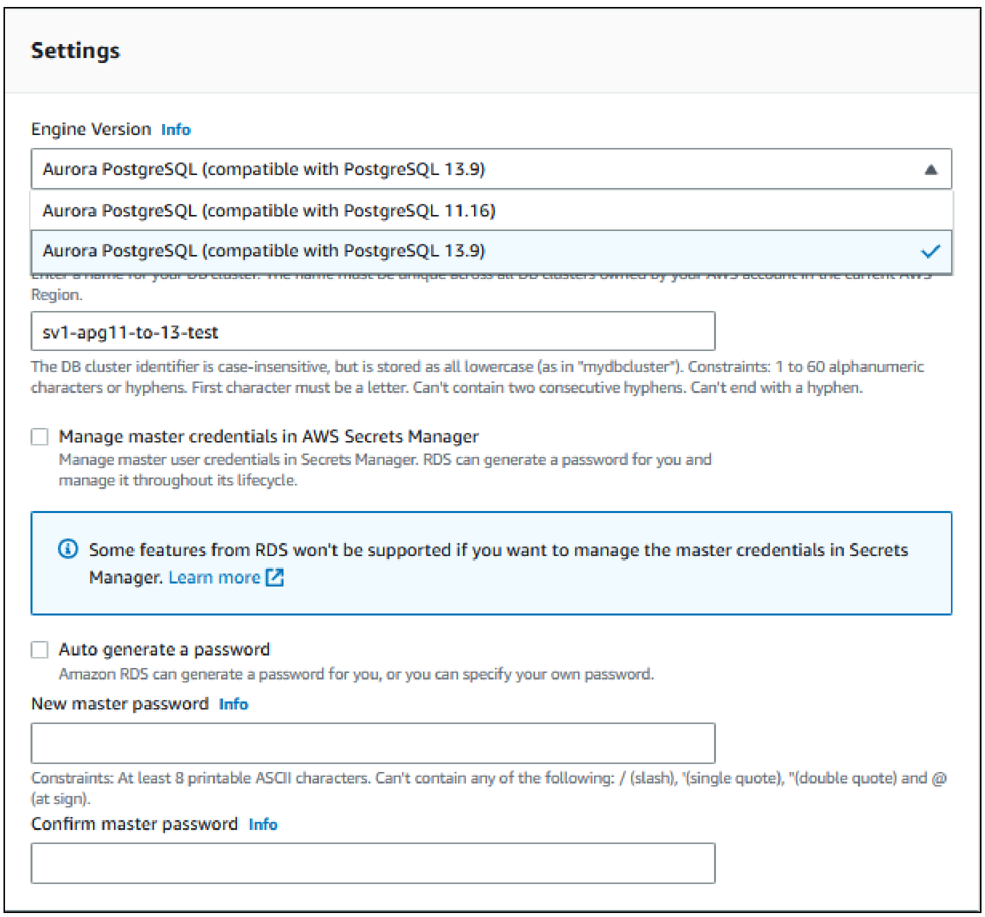
콘솔

AWS Management Console을 사용하여 Aurora Serverless v1 DB 클러스터에 대해 현재 위치 업그레이드를 수행할 수 있습니다.

Aurora Serverless v1 DB 클러스터를 업그레이드하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 업그레이드할 Aurora Serverless v1 DB 클러스터를 선택합니다.
4. 작업에서 클러스터 수정을 선택합니다.
5. 버전에서 Aurora PostgreSQL 버전 13 버전 번호를 선택합니다.

   다음 예는 Aurora MySQL 11.16에서 13.9로의 현재 위치 업그레이드를 보여줍니다.

   메이저 버전 업그레이드를 수행하는 경우 다른 모든 속성을 동일하게 유지합니다. 다른 속성을 변경하려면 업그레이드를 수행한 후에 다른 수정 작업을 진행합니다.
6. 계속을 선택합니다.

7. DB 클러스터 수정 페이지에서 수정 사항을 검토한 후 다음 중 하나를 선택하여 적용합니다.
8. 클러스터 수정을 선택합니다.

AWS CLI

PostgreSQL 11 호환 Aurora Serverless v1 DB 클러스터에서 PostgreSQL 13 호환 클러스터로 현재 위치 업그레이드를 수행하려면 Aurora Serverless v1와 호환되는 Aurora PostgreSQL 버전 13 버전 번호와 함께 `--engine-version` 파라미터를 지정합니다. 또한 `--allow-major-version-upgrade` 파라미터도 포함합니다.

이 예에서는 `sample-cluster`라는 이름의 PostgreSQL 11 호환 Aurora Serverless v1 DB 클러스터의 메이저 버전을 수정합니다. 이렇게 하려면 PostgreSQL 13 호환 Aurora Serverless v1 DB 클러스터로 현재 위치 업그레이드를 수행합니다.

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.serverless_12 \
    --allow-major-version-upgrade
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.serverless_12 ^
    --allow-major-version-upgrade
```

RDS API

PostgreSQL 11 호환 Aurora Serverless v1 DB 클러스터에서 PostgreSQL 13 호환 클러스터로 현재 위치 업그레이드를 수행하려면 Aurora Serverless v1와 호환되는 Aurora PostgreSQL 버전 13 버전 번호와 함께 `EngineVersion` 파라미터를 지정합니다. 또한 `AllowMajorVersionUpgrade` 파라미터도 포함합니다.

Aurora Serverless v1 DB 클러스터를 프로비저닝으로 변환

Aurora Serverless v1 DB 클러스터를 프로비저닝된 DB 클러스터로 변환할 수 있습니다. 변환을 수행하려면 AWS CLI 또는 Amazon RDS API를 사용하여 DB 인스턴스 클래스를 프로비저닝됨으로 변경합니다. 아래 단계에 따라 DB 인스턴스 클래스를 수정합니다.

다음 예제에서는 AWS CLI를 사용하여 Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환하는 방법을 보여줍니다.

AWS CLI

Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환하려면 modify-db-cluster AWS CLI 명령을 실행합니다.

다음 파라미터는 필수 파라미터입니다.

- `--db-cluster-identifier` - 프로비저닝된 상태로 변환할 Aurora Serverless v1 DB 클러스터입니다.
- `--engine-mode` – `provisioned` 값을 사용합니다.
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` - Aurora Serverless v1 DB 클러스터의 용량을 기준으로 프로비저닝된 DB 클러스터의 DB 인스턴스 클래스를 선택합니다.

이 예에서는 이름이 `sample-cluster`인 Aurora Serverless v1 DB 클러스터를 변환하고 `db.r5.xlarge` DB 인스턴스 클래스를 사용합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine-mode provisioned \
--allow-engine-mode-change \
--db-cluster-instance-class db.r5.xlarge
```

Windows의 경우:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-mode provisioned ^
--allow-engine-mode-change ^
--db-cluster-instance-class db.r5.xlarge
```

다음 예제에서는 Amazon RDS API를 사용하여 Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환하는 방법을 보여줍니다.

RDS API

Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환하려면 ModifyDBCluster API 작업을 사용합니다.

다음 파라미터는 필수 파라미터입니다.

- `DBClusterIdentifier` - 프로비저닝된 상태로 변환할 Aurora Serverless v1 DB 클러스터입니다.
- `EngineMode` – `provisioned` 값을 사용합니다.
- `AllowEngineModeChange`
- `DBClusterInstanceClass` - Aurora Serverless v1 DB 클러스터의 용량을 기준으로 프로비저닝된 DB 클러스터의 DB 인스턴스 클래스를 선택합니다.

Aurora Serverless v1 DB 클러스터에서 프로비저닝된 클러스터로 변환할 때 고려 사항

Aurora Serverless v1 DB 클러스터가 프로비저닝된 클러스터로 변환되는 경우 다음 고려 사항이 적용됩니다.

- 이 변환을 DB 클러스터를 Aurora Serverless v1에서 Aurora Serverless v2로 업그레이드하는 과정의 일부로 사용할 수 있습니다. 자세한 내용은 Aurora Serverless v1 클러스터에서 Aurora Serverless v2로 업그레이드 섹션을 참조하세요.
- 변환 프로세스는 DB 클러스터에 리더 DB 인스턴스를 생성하고 리더 인스턴스를 라이터 인스턴스로 승격한 다음, 원본 Aurora Serverless v1 인스턴스를 삭제합니다. DB 클러스터를 변환할 경우 DB 엔진 버전 또는 DB 클러스터 파라미터 그룹 변경과 같은 다른 수정 작업을 동시에 수행할 수 없습니다. 변환 작업은 즉시 적용되며 취소할 수 없습니다.
- 변환 중에 오류가 발생할 경우에 대비하여 DB 클러스터의 백업 DB 클러스터 스냅샷이 생성됩니다. DB 클러스터 스냅샷의 식별자는 `pre-modify-engine-mode-`*`DB_cluster_identifier`*`-`*`timestamp`*의 형태로 되어 있습니다.
- Aurora에서는 프로비저닝된 DB 클러스터에 현재의 기본 DB 마이너 엔진 버전을 사용합니다.
- 변환된 DB 클러스터에 DB 인스턴스 클래스를 사용자가 제공하지 않을 경우, Aurora에서 원본 Aurora Serverless v1 DB 클러스터의 최대 용량을 기준으로 클래스를 권장합니다. 인스턴스 클래스 매핑에 대한 권장 용량은 다음 표에 나와 있습니다.

Serverless 최대 용량(ACU)	프로비저닝된 DB 인스턴스 클래스
1	db.t3.small
2	db.t3.medium
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

Note

선택한 DB 인스턴스 클래스와 데이터베이스 사용량에 따라, 프로비저닝된 DB 클러스터의 비용은 Aurora Serverless v1에 비해 다르게 나타날 수 있습니다.

Aurora Serverless v1 DB 클러스터를 버스트 가능한(db.t*) DB 인스턴스 클래스로 변환할 경우 DB 클러스터 사용에 대한 추가 비용이 발생할 수 있습니다. 자세한 내용은 [DB 인스턴스 클래스 유형](#) 섹션을 참조하세요.

수동으로 Aurora Serverless v1 DB 클러스터 용량 확장

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

일반적으로 Aurora Serverless v1 DB 클러스터는 워크로드에 따라 원활하게 확장됩니다. 하지만 트랜잭션의 기하급수적 증가와 같은 갑작스러운 급증 상황을 충족할 수 있을 만큼 용량이 항상 빠르게 확장되지는 않을 수 있습니다. 이러한 경우 새 용량 값을 설정하여 조정 작업을 수동으로 시작할 수 있습니다. 용량을 명시적으로 설정하면 Aurora Serverless v1가 DB 클러스터를 자동으로 조정합니다. 이 작업은 축소를 위한 휴지 기간에 근거하여 이루어집니다.

Aurora Serverless v1, AWS Management Console 또는 RDS API를 사용하여 AWS CLI DB 클러스터의 용량을 특정 값으로 명시적으로 설정할 수 있습니다.

콘솔

AWS Management Console을 사용하여 Aurora DB 클러스터의 용량을 설정할 수 있습니다.

Aurora Serverless v1 DB 클러스터를 수정하려면

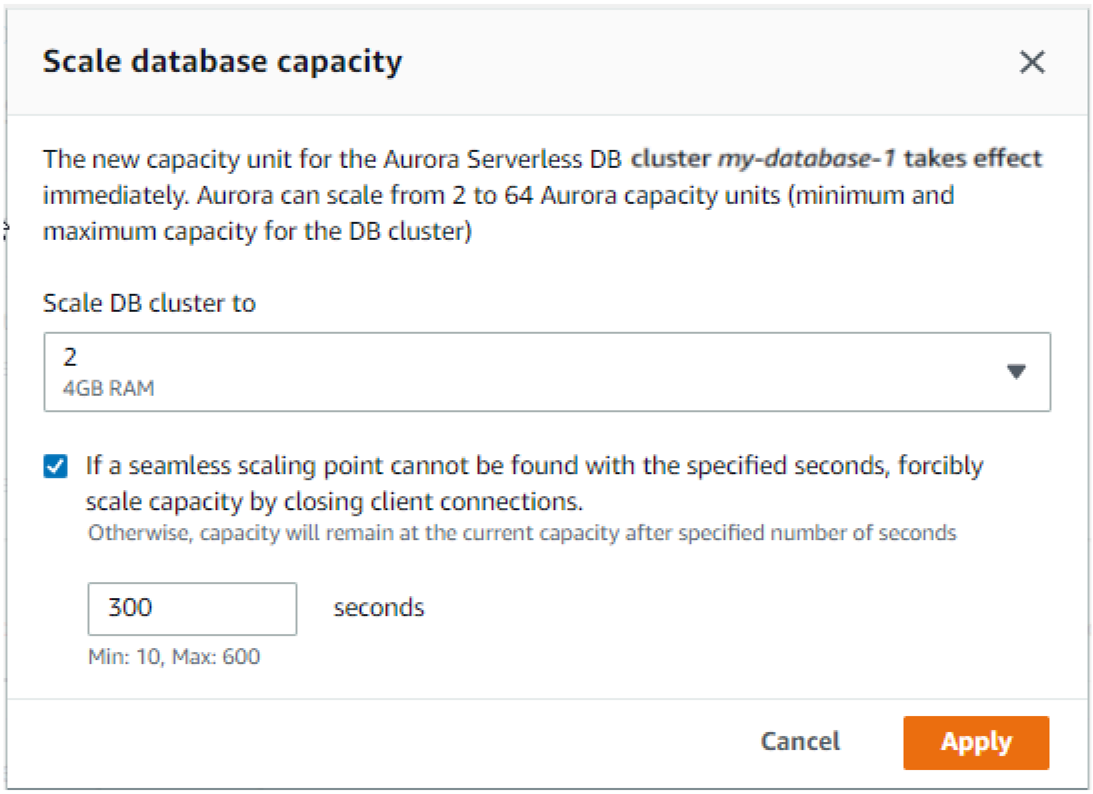
1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 수정할 Aurora Serverless v1 DB 클러스터를 선택합니다.
4. 작업에서 Set capacity(용량 설정)를 선택합니다.
5. [데이터베이스 용량 조정(Scale database capacity)] 창에서 다음을 선택합니다.

    a. [다음으로 DB 클러스터 조정(Scale DB cluster to)] 드롭다운 선택기에서 DB 클러스터에 사용할 새 용량을 선택합니다.

    b. [원활한 조정점을 찾을 수 없는 경우(If a seamless scaling point cannot be found)] 확인란에서 다음과 같이 Aurora Serverless v1 DB 클러스터 `TimeoutAction` 설정에 대해 원하는 동작을 선택합니다.

- Aurora Serverless v1가 시간 초과되기 전에 크기 조정 지점을 찾을 수 없는 경우 용량을 변경하지 않으려면 이 옵션을 선택 취소합니다.
- 시간 초과되기 전에 크기 조정 지점을 찾을 수 없더라도 Aurora Serverless v1 DB 클러스터의 용량을 강제로 변경하려면 이 옵션을 선택합니다. 이 옵션을 사용하면 Aurora Serverless v1가 연결을 삭제하여 조정점을 찾을 수 없게 됩니다.

c. 시간이 초과되기 전에 Aurora Serverless v1 DB 클러스터가 조정 지점을 찾도록 허용할 시간을 초 동안 입력합니다. 10초에서 600초(10분)까지 지정할 수 있습니다. 기본값은 5분(300초)입니다. 다음 예에서는 5분 이내에 조정점을 찾을 수 없더라도 Aurora Serverless v1 DB 클러스터가 2개의 ACU로 축소되도록 강제합니다.



6. 적용을 선택합니다.

조정점, `TimeoutAction` 및 휴지 기간에 대한 자세한 내용은 Aurora Serverless v1에서의 Auto Scaling 섹션을 참조하세요.

AWS CLI

Aurora Serverless v1를 사용하여 AWS CLI DB 클러스터의 용량을 설정하려면 modify-current-db-cluster-capacity AWS CLI 명령을 실행하고 `--capacity` 옵션을 지정합니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

이 예에서는 *sample-cluster*라는 Aurora Serverless v1 DB 클러스터의 용량을 *64*로 설정합니다.

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

RDS API

ModifyCurrentDBClusterCapacity API 작업을 사용하여 Aurora DB 클러스터의 용량을 설정할 수 있습니다. `Capacity` 파라미터를 지정합니다. 유효한 용량 값은 다음과 같습니다.

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, 256
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 및 384.

Aurora Serverless v1 DB 클러스터 보기

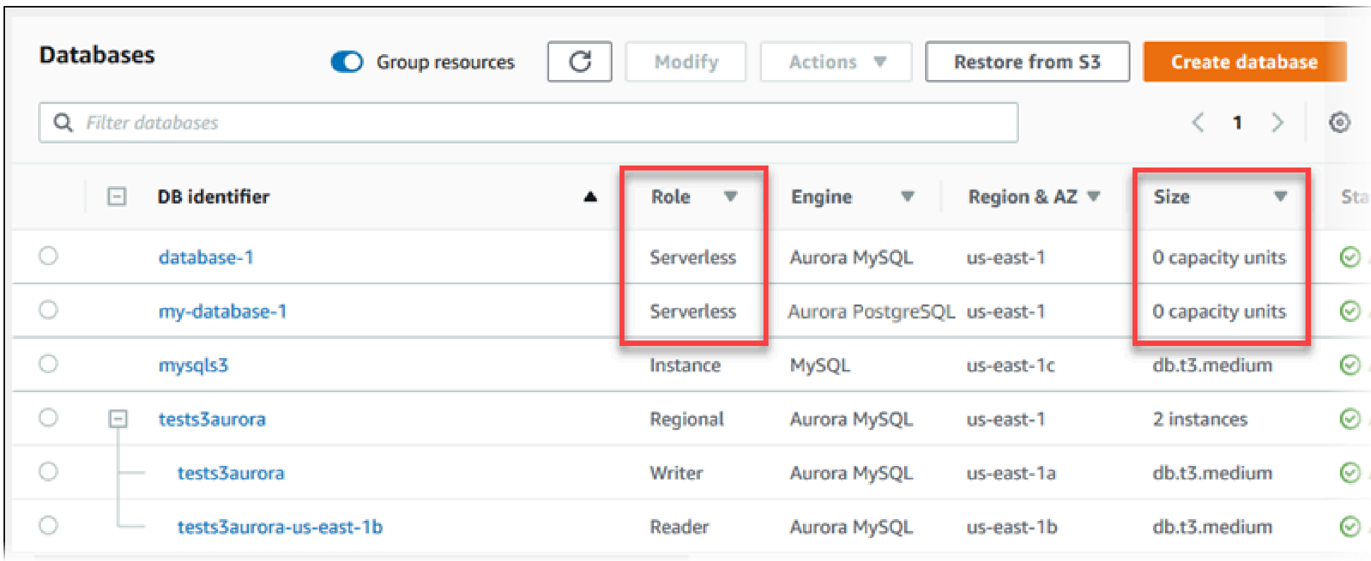
> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터를 하나 이상 생성하면 유형이 서버리스인 DB 클러스터와 인스턴스인 DB 클러스터를 확인할 수 있습니다. 각 Aurora Serverless v1 DB 클러스터에서 사용 중인 Aurora 용량 단위(ACU)의 현재 개수도 확인할 수 있습니다. 각 ACU는 처리(CPU) 용량과 메모리(RAM) 용량의 조합입니다.

Aurora Serverless v1 DB 클러스터를 보려면

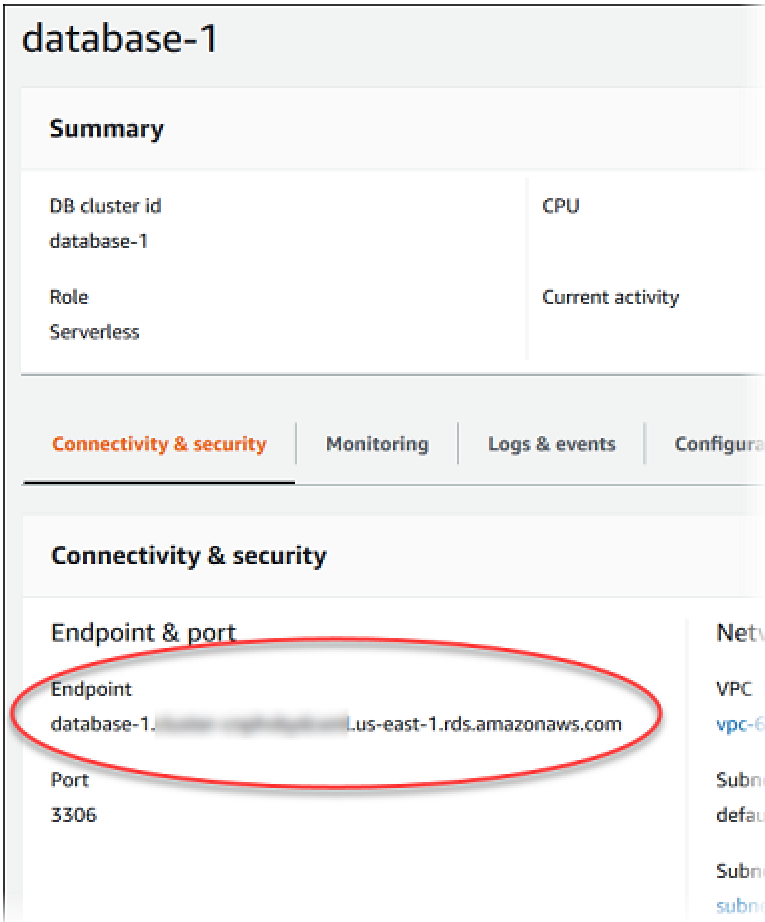
1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 AWS 리전 DB 클러스터를 생성한 Aurora Serverless v1을 선택합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.

   각 DB 클러스터의 DB 클러스터 유형은 역할 아래에 표시됩니다. Aurora Serverless v1 DB 클러스터는 유형에 대해 서버리스라고 표시합니다. Aurora Serverless v1 DB 클러스터의 현재 용량은 크기에서 확인할 수 있습니다.

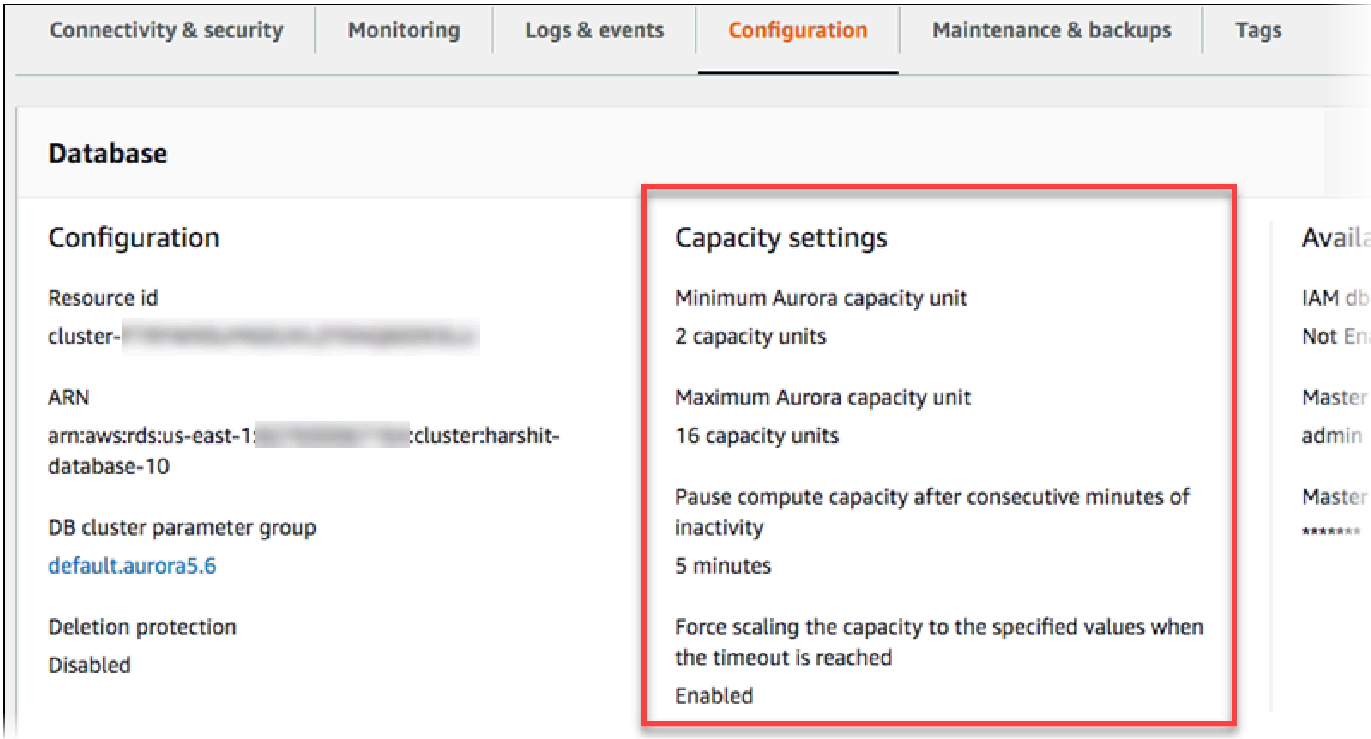


4. Aurora Serverless v1 DB 클러스터의 이름을 선택하면 세부 정보가 표시됩니다.

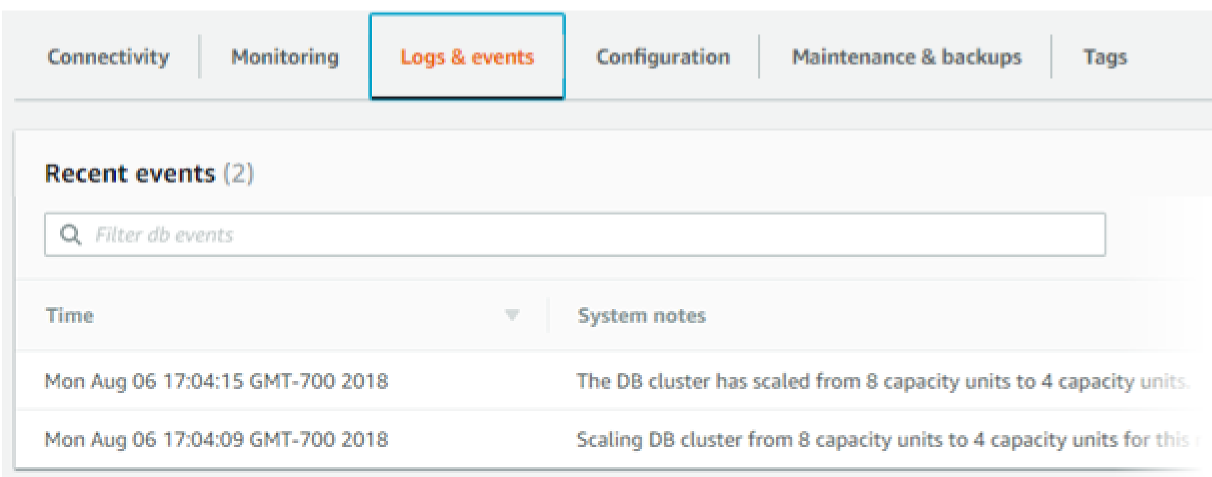
   [연결 및 보안(Connectivity & security)] 탭에서 데이터베이스 엔드포인트를 확인합니다. 이 엔드포인트를 사용하여 Aurora Serverless v1 DB 클러스터에 연결합니다.



[구성(Configuration)] 탭을 선택하여 용량 설정을 확인합니다.



조정 이벤트는 DB 클러스터 확장, 축소, 일시 중지 또는 다시 시작 시 생성됩니다. Logs & events(로그 및 이벤트) 탭을 선택하여 최근 이벤트를 확인합니다. 다음 이미지는 이러한 이벤트의 예를 보여줍니다.



Aurora Serverless v1 DB 클러스터의 용량 및 조정 이벤트 모니터링

CloudWatch에서 Aurora Serverless v1 DB 클러스터를 보고 `ServerlessDatabaseCapacity` 지표를 사용하여 DB 클러스터에 할당된 용량을 모니터링할 수 있습니다. 또한 `CPUUtilization`, `DatabaseConnections`, `Queries` 등의 표준 Aurora CloudWatch 지표를 모두 모니터링할 수 있습니다.

Aurora가 데이터베이스 로그 중 일부 또는 전체를 CloudWatch에 게시하게 할 수 있습니다. `general_log` 클러스터에 연결된 DB 클러스터 파라미터 그룹에 있는 `slow_query_log`, Aurora Serverless v1 등의 구성 파라미터를 활성화하여 게시할 로그를 선택합니다. 프로비저닝된 클러스터와 달리 Aurora Serverless v1 클러스터의 경우 DB 클러스터 설정에서 CloudWatch에 업로드할 로그 유형을 지정할 필요가 없습니다. Aurora Serverless v1 클러스터가 사용 가능한 모든 로그를 자동으로 업로드합니다. 로그 구성 파라미터를 비활성화하면 CloudWatch에 대한 로그 게시가 중지됩니다. 더 이상 필요하지 않은 경우 CloudWatch에서 로그를 삭제할 수도 있습니다.

Aurora Serverless v1 DB 클러스터에 대해 Amazon CloudWatch를 시작하려면 Amazon CloudWatch를 사용하여 Aurora Serverless v1 로그 보기 섹션을 참조하세요. CloudWatch를 통해 Aurora DB 클러스터를 모니터링하는 방법에 대한 자세한 내용은 Amazon CloudWatch에서 로그 이벤트 모니터링 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터에 연결하려면 데이터베이스 엔드포인트를 사용합니다. 자세한 내용은 Amazon Aurora DB 클러스터에 연결 섹션을 참조하세요.

> **Note**
>
> Aurora Serverless v1 DB 클러스터의 특정 DB 인스턴스에는 직접 연결할 수 없습니다.

Aurora Serverless v1 DB 클러스터 삭제

> **Important**
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터를 만드는 방법에 따라 삭제 방지 기능이 기본적으로 켜져 있을 수 있습니다. 삭제 방지 기능이 활성화된 Aurora Serverless v1 DB 클러스터는 즉시 삭제할 수 없습니다.

삭제 방지된 Aurora Serverless v1 DB 클러스터를 AWS Management Console을 사용하여 삭제하려면 먼저 클러스터를 수정하여 이 보호를 제거합니다. 이 태스크에 AWS CLI를 사용하는 방법에 대한 자세한 내용은 AWS CLI 섹션을 참조하세요.

AWS Management Console을 사용하여 삭제 방지 기능을 비활성화하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [DB 클러스터(DB clusters)]를 선택합니다.
3. 목록에서 Aurora Serverless v1 DB 클러스터를 선택합니다.
4. [수정(Modify)]을 선택하여 DB 클러스터의 구성을 엽니다. [DB 클러스터 수정(Modify DB cluster)] 페이지에 Aurora Serverless v1 DB 클러스터에 대한 설정, 용량 설정 및 기타 구성 세부 정보가 표시됩니다. 삭제 방지 기능은 [추가 구성(Additional configuration)] 섹션에 있습니다.
5. 추가 구성(Additional configuration) 속성 카드에서 삭제 방지 활성화(Enable deletion protection) 확인란을 선택 취소합니다.
6. [Continue]를 선택합니다. [수정 사항 요약(Summary of modifications)]이 나타납니다.
7. [클러스터 수정(Modify cluster)]을 선택하여 수정 사항 요약을 적용합니다. [뒤로(Back)]를 선택하여 변경 사항을 수정하거나 [취소(Cancel)]를 선택하여 변경 사항을 취소할 수도 있습니다.

삭제 방지 기능이 비활성화되면 Aurora Serverless v1을 사용하여 AWS Management Console DB 클러스터를 삭제할 수 있습니다.

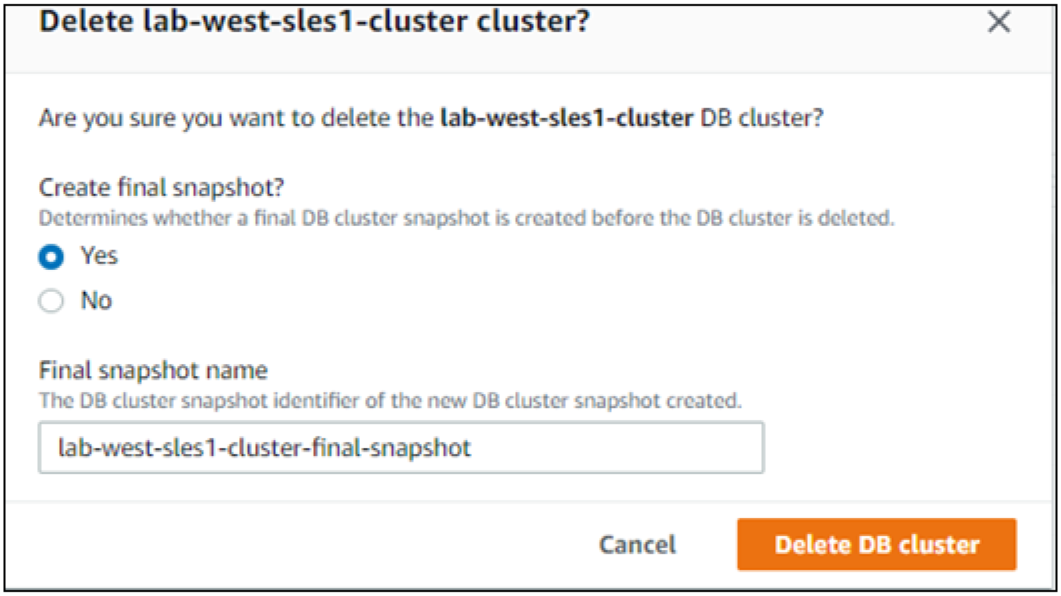
콘솔

Aurora Serverless v1 DB 클러스터를 삭제하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 리소스 섹션에서 DB 클러스터를선택합니다.
3. 삭제할 Aurora Serverless v1 DB 클러스터를 선택합니다.
4. [Actions]에 대해 [Delete]를 선택합니다. Aurora Serverless v1 DB 클러스터를 삭제할지 여부를 확인하는 메시지가 표시됩니다.
5. 미리 선택된 옵션을 유지하는 것이 좋습니다.

   • [최종 스냅샷을 생성하시겠습니까?(Create final snapshot?)]에 대해 [예(Yes)]를 선택합니다.

- [최종 스냅샷 이름(Final snapshot name)]에는 Aurora Serverless v1 DB 클러스터에 `-final-snapshot`을 붙인 값을 입력합니다. 그러나 이 필드에서 최종 스냅샷의 이름을 변경할 수 있습니다.



[최종 스냅샷을 생성하시겠습니까?(Create final snapshot?)]에 대해 [아니요(No)]를 선택한 경우 스냅샷이나 특정 시점으로 복구 기능을 사용하여 DB 클러스터를 복원할 수 없습니다.

6. [DB 클러스터 삭제(Delete DB cluster)]를 선택합니다.

Aurora Serverless v1가 DB 클러스터를 삭제합니다. 최종 스냅샷을 생성하도록 선택한 경우, 삭제되기 전에 Aurora Serverless v1 DB 클러스터의 상태가 “백업 중”으로 변경되어 목록에 더 이상 나타나지 않습니다.

AWS CLI

시작하기 전에 AWS CLI 액세스 키 ID, AWS 보안 액세스 키 및 AWS DB 클러스터가 있는 AWS 리전을 사용하여 Aurora Serverless v1를 구성합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 구성 기초를 참조하세요.

이 옵션으로 구성된 클러스터에 대해 삭제 방지를 먼저 비활성화해야 Aurora Serverless v1 DB 클러스터를 삭제할 수 있습니다. 이 보호 옵션이 활성화되어 있는 클러스터를 삭제하려고 하면 다음 오류 메시지가 나타납니다.

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
```

```
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

다음과 같이 modify-db-cluster Aurora Serverless v1 명령을 사용하여 AWS CLI DB 클러스터의 삭제 방지 설정을 변경할 수 있습니다.

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

이 명령은 지정된 DB 클러스터의 수정된 속성을 반환합니다. 이제 Aurora Serverless v1 DB 클러스터를 삭제할 수 있습니다.

Aurora Serverless v1 DB 클러스터를 삭제할 때마다 항상 최종 스냅샷을 생성하는 것이 좋습니다. 다음의 AWS CLI delete-db-cluster 사용 예는 그 방법을 보여줍니다. DB 클러스터의 이름과 스냅샷의 이름을 제공합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

Windows의 경우:

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 및 Aurora 데이터베이스 엔진 버전

> Important
>
> AWS는 Aurora Serverless v1의 수명 종료일을 2025년 3월 31일로 발표했습니다. 2025년 3월 31일까지 마이그레이션되지 않은 모든 Aurora Serverless v1 클러스터는 유지 관리 기간 동안 Aurora Serverless v2로 마이그레이션됩니다. 업그레이드에 실패하면 Amazon Aurora는 유지 관리 기간 동안 Serverless v1 클러스터를 동등한 엔진 버전을 가진 프로비저닝된 클러스터로 변환합니다. 해당하는 경우 Amazon Aurora는 Amazon RDS 추가 지원에 변환된 프로비저닝된 클러스터를 등록합니다. 자세한 내용은 RDS 추가 지원 섹션을 참조하세요.

Aurora Serverless v1은 특정 AWS 리전과 특정 Aurora MySQL 및 Aurora PostgreSQL 버전에서만 사용할 수 있습니다. AWS 리전을 지원하는 Aurora Serverless v1의 최신 목록과 각 리전에서 사용할 수 있는 특정 Aurora MySQL 및 Aurora PostgreSQL 버전은 Aurora Serverless v1을 참조하세요.

Aurora Serverless v1은 연결된 Aurora 데이터베이스 엔진을 사용하여 지원되는 각 데이터베이스 엔진별로 지원되는 특정 릴리스를 식별합니다.

- Aurora MySQL Serverless
- Aurora PostgreSQL Serverless

Aurora Serverless v1용으로 데이터베이스 엔진의 마이너 릴리스가 제공되면 AWS 리전를 사용할 수 있는 다양한 Aurora Serverless v1에서 자동으로 적용됩니다. 즉, Aurora Serverless v1용으로 제공되는 클러스터 DB 엔진에 대한 새 마이너 릴리스를 얻기 위해 Aurora Serverless v1 DB 클러스터를 업그레이드할 필요가 없습니다.

Aurora MySQL Serverless

Aurora Serverless v1은 특정 AWS 리전와 특정 Aurora MySQL 버전에서만 사용할 수 있습니다. Aurora Serverless v1을 지원하는 AWS 리전의 최신 목록과 각 리전에서 사용할 수 있는 특정 Aurora MySQL 버전은 Aurora MySQL을 사용하는 Aurora Serverless v1 섹션을 참조하세요.

Aurora MySQL 버전 2의 향상된 기능 및 버그 수정에 대해 알아보려면 Aurora MySQL 릴리스 정보의 Amazon Aurora MySQL 버전 2에 대한 데이터베이스 엔진 업데이트를 참조하세요.

Aurora Serverless v2를 사용하여 최신 버전의 Aurora MySQL을 사용할 수 있습니다. Aurora Serverless v2와 함께 사용할 수 있는 AWS 리전 및 Aurora MySQL 버전에 대한 자세한 내용은 Aurora MySQL을 사용하는 Aurora Serverless v2 섹션을 참조하세요. Aurora Serverless v2 사용 정보는 Aurora Serverless v2 사용하기 섹션을 참조하세요.

Aurora PostgreSQL Serverless

Aurora Serverless v1은 특정 AWS 리전과 특정 Aurora PostgreSQL 버전에서만 사용할 수 있습니다. Aurora Serverless v1을 지원하는 AWS 리전의 최신 목록과 각 리전에서 사용할 수 있는 특정 Aurora PostgreSQL 버전은 Aurora PostgreSQL을 사용하는 Aurora Serverless v1 섹션을 참조하세요.

Aurora Serverless v1 DB 클러스터에 Aurora PostgreSQL을 사용하려는 경우 Aurora PostgreSQL 13 호환 버전을 사용하면 됩니다. Aurora PostgreSQL 호환 버전의 마이너 릴리스에는 이전 버전과 호환되는 변경 사항만 포함됩니다. Aurora Serverless v1 DB 클러스터는 사용 중인 AWS 리전에서 Aurora

Serverless v1에 Aurora PostgreSQL 마이너 릴리스를 사용할 수 있게 되면 자동으로 업그레이드됩니다.

Aurora Serverless v2를 사용하여 최신 버전의 Aurora PostgreSQL을 사용할 수 있습니다. Aurora Serverless v2와 함께 사용할 수 있는 AWS 리전 및 Aurora PostgreSQL 버전에 대한 자세한 내용은 Aurora PostgreSQL을 사용하는 Aurora Serverless v2 섹션을 참조하세요. Aurora Serverless v2 사용 정보는 Aurora Serverless v2 사용하기 섹션을 참조하세요.

Aurora Serverless v1 마이너 버전 자동 업그레이드

Aurora Serverless v1용으로 데이터베이스 엔진의 마이너 릴리스가 제공되면 AWS 리전를 사용할 수 있는 다양한 Aurora Serverless v1에서 자동으로 적용됩니다. 즉, Aurora Serverless v1용으로 제공되는 클러스터 DB 엔진에 대한 새 마이너 릴리스를 얻기 위해 Aurora Serverless v1 DB 클러스터를 업그레이드할 필요가 없습니다.

Amazon RDS Data API 사용

RDS 데이터 API(데이터 API)를 사용하면 Aurora DB 클러스터에 대한 웹 서비스 인터페이스로 작업할 수 있습니다. 데이터 API에서 DB 클러스터에 지속적으로 연결하지 않아도 됩니다. 대신에 AWS SDK와의 통합과 보안 HTTP 엔드포인트를 제공합니다. 연결을 관리하지 않고 엔드포인트를 사용하여 SQL 문을 실행할 수 있습니다.

데이터 API는 AWS Secrets Manager에 저장된 데이터베이스 보안 인증 정보를 사용하기 때문에 사용자는 데이터 API 호출과 함께 보안 인증 정보를 전달할 필요가 없습니다. Secrets Manager에 보안 인증 정보를 저장하려면 사용자에게 Secrets Manager 및 데이터 API를 사용할 수 있는 적절한 권한을 부여해야 합니다. 사용자 인증에 대한 자세한 내용은 Amazon RDS Data API에 대한 액세스 권한 부여 단원을 참조하세요.

또한 데이터 API를 사용하여 Amazon Aurora를 다른 AWS 애플리케이션(예: AWS Lambda, AWS AppSync, AWS Cloud9)과 통합할 수 있습니다 데이터 API는 AWS Lambda를 사용하는 더 안전한 방법도 제공합니다. Virtual Private Cloud(VPC)의 리소스에 액세스하는 Lambda 함수를 구성할 필요 없이 DB 클러스터에 액세스할 수 있습니다. 자세한 내용은 AWS Lambda, AWS AppSync, AWS Cloud9 단원을 참조하십시오.

Aurora DB 클러스터를 생성할 때 데이터 API를 활성화할 수 있습니다. 나중에 구성을 수정할 수도 있습니다. 자세한 내용은 Amazon RDS Data API 활성화 섹션을 참조하세요.

데이터 API를 활성화하면 VPC에서 Aurora에 액세스하도록 쿼리 도구를 구성하지 않고도 쿼리 편집기를 사용하여 임시 쿼리를 실행할 수 있습니다. 자세한 내용은 Aurora 쿼리 편집기 사용하기 섹션을 참조하세요.

주제

- RDS 데이터 API용 Java 클라이언트 라이브러리 사용
- JSON 형식의 Amazon RDS Data API 쿼리 결과 처리
- Amazon RDS Data API 문제 해결
- AWS CloudTrail을 사용하여 Amazon RDS Data API 직접 호출 로깅
- 성능 개선 도우미를 사용한 RDS 데이터 API 쿼리 모니터링

Amazon RDS Data API의 리전 및 버전 가용성

데이터 API에 사용할 수 있는 리전 및 엔진 버전에 대한 자세한 내용은 다음 섹션을 참조하세요.

클러스터 유형	리전 및 버전 사용 가능 여부
Aurora PostgreSQL 프로비저닝 및 Serverless v2	Aurora PostgreSQL Serverless v2를 사용하고 프로비저닝된 Data API
Aurora MySQL 프로비저닝 및 Serverless v2	???
Aurora PostgreSQL Serverless v1	Aurora PostgreSQL Serverless v1을 사용하는 Data API
Aurora MySQL Serverless v1	Aurora MySQL Serverless v1을 사용하는 Data API

명령줄 인터페이스 또는 API를 통해 데이터 API에 액세스할 때 FIPS 140-2에서 유효성을 검사한 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2를 참조하세요.

Amazon RDS Data API에 대한 제한 사항

RDS 데이터 API에는 다음과 같은 제한 사항이 적용됩니다.

- DB 클러스터의 라이터 인스턴스에서만 데이터 API 쿼리를 실행할 수 있습니다. 하지만 라이터 인스턴스는 쓰기 쿼리와 읽기 쿼리를 모두 수락할 수 있습니다.
- Aurora Global Database를 사용하면 기본 및 보조 DB 클러스터 모두에서 데이터 API를 활성화할 수 있습니다. 하지만 보조 클러스터는 기본 클러스터로 승격되기 전까지는 라이터 인스턴스를 갖지 않

습니다. 데이터 API를 사용하려면 읽기 전용 쿼리를 포함한 쿼리 처리를 위해 라이터 인스턴스에 액세스해야 합니다. 따라서 라이터 인스턴스가 없는 동안 보조 클러스터로 전송된 읽기 및 쓰기 쿼리가 실패합니다. 보조 클러스터가 승격되고 사용 가능한 라이터 인스턴스를 갖게 되면 해당 DB 인스턴스에 대한 데이터 API 쿼리가 성공합니다.

- T DB 인스턴스 클래스에서는 데이터 API가 지원되지 않습니다.
- Aurora Serverless v2 및 프로비저닝된 DB 클러스터의 경우 RDS 데이터 API는 일부 데이터 유형을 지원하지 않습니다. 지원되는 유형 목록은 [the section called “Aurora Serverless v1 및 Aurora Serverless v2의 Data API 비교”](#) 섹션을 참조하세요.
- Aurora PostgreSQL 버전 14 이상의 데이터베이스에서는 데이터 API가 암호 암호화에 `scram-sha-256`만 지원합니다.
- 응답 크기 제한은 1MiB입니다. 호출이 1MiB를 초과하는 응답 데이터를 반환하면 호출이 종료됩니다.
- Aurora Serverless v1의 경우 초당 최대 요청 수는 1,000개입니다. 지원되는 다른 모든 데이터베이스의 경우 제한이 없습니다.
- 데이터 API는 데이터베이스에서 반환되는 결과 집합에서 각 행의 크기를 64KB로 제한합니다. 결과 집합의 각 행 크기를 64KB 이하로 유지해야 합니다.

Aurora Serverless v2 및 프로비저닝된 클러스터와 Aurora Serverless v1 클러스터에 대한 Amazon RDS Data API 동작 비교

Amazon RDS Data API의 최신 개선 사항을 통해 최신 버전의 PostgreSQL 또는 MySQL 엔진을 사용하는 클러스터에서도 Data API를 사용할 수 있습니다. 이러한 클러스터는 `db.r6g` 또는 `db.r6i` 등의 프로비저닝된 인스턴스 클래스나 Aurora Serverless v2를 사용하도록 구성할 수 있습니다.

다음 섹션에는 Aurora Serverless v2 및 프로비저닝된 DB 클러스터와 Aurora Serverless v1 DB 클러스터 간의 Amazon RDS Data API에 대한 차이점이 나와 있습니다. Aurora Serverless v1 DB 클러스터는 `serverless` 엔진 모드를 사용합니다. 프로비저닝 DB 클러스터는 `provisioned` 엔진 모드를 사용합니다. 또한 Aurora Serverless v2 DB 클러스터는 `provisioned` 엔진 모드를 사용하며, `db.serverless` 인스턴스 클래스와 함께 하나 이상의 Aurora Serverless v2 DB 인스턴스를 포함합니다.

초당 최대 요청 수

Aurora Serverless v1

Data API는 초당 최대 1,000개의 요청을 생성할 수 있습니다.

Aurora Serverless v2

Data API는 초당 무제한의 요청을 생성할 수 있습니다.

기존 데이터베이스에서 Amazon RDS Data API 활성화 또는 비활성화

Aurora Serverless v1

- Amazon RDS API의 경우 – `ModifyCluster` 작업을 사용하고 `EnableHttpEndpoint` 파라미터에 대해 `True` 또는 `False`(해당하는 경우)를 지정합니다.
- AWS CLI의 경우 – `--enable-http-endpoint` 또는 `--no-enable-http-endpoint`(해당하는 경우) 옵션과 함께 `modify-db-cluster` 작업을 사용합니다.

Aurora Serverless v2

- Amazon RDS API 사용 - `EnableHttpEndpoint` 및 `DisableHttpEndpoint` 작업을 사용합니다.
- AWS CLI의 경우: `enable-http-endpoint` 및 `disable-http-endpoint` 작업을 사용합니다.

CloudTrail 이벤트

Aurora Serverless v1

데이터 API 호출의 이벤트는 관리 이벤트입니다. 이러한 이벤트는 기본적으로 추적에 자동으로 포함됩니다. 자세한 내용은 the section called "CloudTrail 추적에서 데이터 API 이벤트 제외(Aurora Serverless v1 전용)" 섹션을 참조하세요.

Aurora Serverless v2

데이터 API 호출의 이벤트는 데이터 이벤트입니다. 이러한 이벤트는 기본적으로 추적에서 자동으로 제외됩니다. 자세한 내용은 the section called "CloudTrail 추적에서 데이터 API 이벤트 포함" 섹션을 참조하세요.

다중 문 지원

Aurora Serverless v1

- Aurora MySQL의 경우 다중 문이 지원되지 않습니다.
- Aurora PostgreSQL의 경우 다중 문은 첫 번째 쿼리 응답만 반환합니다.

Aurora Serverless v2

다중 문은 지원되지 않습니다. 단일 API 직접 호출에서 다중 문을 실행하려고 하면 “An error occurred (ValidationException) when calling the ExecuteStatement operation: Multistatements aren't supported.”가 발생합니다. 다중 문을 실행하려면 별도의 ExecuteStatement API 직접 호출을 사용하거나 배치 처리를 위해 BatchExecuteStatement를 사용합니다.

다음 예제에서는 다중 문 실행을 시도하는 API 직접 호출의 결과로 표시되는 오류 메시지를 보여 줍니다.

```
aws rds-data execute-statement \
   --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
   --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
   --database "your_database" \
   --sql "SELECT * FROM your_table; Select * FROM next_table;

                                "An error occurred (ValidationException) when calling
the ExecuteStatement operation: Multistatements aren't supported.
```

다음 예제에서는 별도의 ExecuteStatement API 직접 호출을 사용하여 여러 문을 실행합니다.

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table;"

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM next_table;"
```

동일한 트랜잭션 ID에 대한 동시 요청

Aurora Serverless v1

후속 요청은 현재 요청이 완료될 때까지 대기합니다. 대기 기간이 너무 길면 애플리케이션에서 제한 시간 초과 오류를 처리해야 합니다.

Aurora Serverless v2

Data API가 동일한 트랜잭션 ID로 여러 요청을 수신하면 즉시 다음 오류를 반환합니다.

```
DatabaseErrorException: Transaction is still running a query
```

이 오류는 다음의 두 가지 상황에서 발생합니다.

- 애플리케이션은 동일한 트랜잭션 ID를 사용하여 비동기식 요청(예: JavaScript promise)을 수행합니다.
- 해당 트랜잭션 ID의 이전 요청은 여전히 처리 중입니다.

다음 예제는 `promise.all()`과 병렬로 실행되는 모든 요청을 보여 줍니다.

```
const api_calls = [];
for (let i = 0; i < 10; i++) {
api_calls.push(
    client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    )
);
}
await Promise.all(api_calls);
```

이 오류를 해결하려면 동일한 트랜잭션 ID로 다른 요청을 보내기 전에 현재 요청이 완료될 때까지 기다리거나 병렬 요청을 허용하도록 트랜잭션 ID를 제거합니다.

다음 예제에서는 동일한 트랜잭션 ID로 순차적 실행을 사용하는 API 직접 호출을 보여 줍니다.

```
 for (let i = 0; i < 10; i++) {
    await client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    ).promise()
```

```
);
}
```

BatchExecuteStatement 동작

`BatchExecuteStatement`에 대한 자세한 내용은 [BatchExecuteStatement](https://)를 참조하세요.

Aurora Serverless v1

업데이트 결과에서 생성된 필드 객체에는 삽입된 값이 포함됩니다.

Aurora Serverless v2

- Aurora MySQL의 경우 업데이트 결과에 생성된 필드 객체에는 삽입된 값이 포함됩니다.
- Aurora PostgreSQL에서 생성된 필드 객체는 비어 있습니다.

ExecuteSQL 동작

`ExecuteSQL`에 대한 자세한 내용은 ExecuteSQL을 참조하세요.

Aurora Serverless v1

`ExecuteSQL` 작업은 더 이상 사용되지 않습니다.

Aurora Serverless v2

`ExecuteSQL` 작업은 지원되지 않습니다.

ExecuteStatement 동작

`ExecuteStatement`에 대한 자세한 내용은 ExecuteStatement를 참조하세요.

Aurora Serverless v1

`ExecuteStatement` 파라미터는 다차원 배열 열 및 모든 고급 데이터 유형 검색을 지원합니다.

Aurora Serverless v2

`ExecuteStatement` 파라미터는 다차원 배열 열을 지원하지 않습니다. 또한 지오메트리 및 통화 유형을 포함한 특정 PostgreSQL 데이터 형식을 지원하지 않습니다. Data API가 지원되지 않는 데이터 유

형을 발견하면 UnsupportedResultException: The result contains the unsupported data type data_type 오류를 반환합니다.

이 문제를 해결하려면 지원되지 않는 데이터 유형을 TEXT로 캐스팅합니다. 다음 예제에서는 지원되지 않는 데이터 형식을 TEXT로 캐스팅합니다.

```
SELECT custom_type::TEXT FROM my_table;--
ORSELECT CAST(custom_type AS TEXT) FROM my_table;
```

각 Aurora 데이터베이스 엔진에 지원되는 데이터 유형 목록은 Data API 작업 참조를 참조하세요.

스키마 파라미터 동작

Aurora Serverless v1

Schema 파라미터는 지원되지 않습니다. API 직접 호출에 Schema 파라미터를 포함하면 Data API는 파라미터를 무시합니다.

Aurora Serverless v2

Schema 파라미터는 이제 사용되지 않습니다. API 직접 호출에 Schema 파라미터를 포함하면 Data API는 ValidationException: The schema parameter isn't supported 오류를 반환합니다. 다음 예제에서는 ValidationException 오류를 반환하는 Data API 직접 호출을 보여 줍니다.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--schema "your_schema" \
--sql "SELECT * FROM your_table LIMIT 10"
```

이 문제를 해결하려면 API 직접 호출에서 Schema 파라미터를 제거합니다.

다음 예제에서는 Schema 파라미터가 제거된 Data API 직접 호출을 보여 줍니다.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
```

```
--sql "SELECT * FROM your_table LIMIT 10"
```

Amazon RDS Data API에 대한 액세스 권한 부여

사용자는 권한이 있는 경우에만 RDS Data API(Data API) 작업을 간접적으로 호출할 수 있습니다. 권한을 정의하는 AWS Identity and Access Management(IAM) 정책을 연결하여 데이터 API를 사용할 수 있는 권한을 사용자에게 부여할 수 있습니다. IAM 역할을 사용하는 경우 정책을 역할에 연결할 수도 있습니다. AWS 관리형 정책인 `AmazonRDSDataFullAccess`에는 데이터 API에 대한 권한이 포함되어 있습니다.

`AmazonRDSDataFullAccess` 정책에는 사용자가 AWS Secrets Manager에서 보안 암호 값을 가져올 수 있는 권한도 포함되어 있습니다. 사용자는 데이터 API 호출에 사용할 수 있는 보안 암호를 저장하는 데 Secrets Manager를 사용해야 합니다. 보안 암호를 사용하면 사용자가 데이터 API 호출의 대상이 되는 리소스에 대한 데이터베이스 보안 인증 정보를 포함할 필요가 없습니다. 데이터 API는 투명하게 Secrets Manager를 호출하여 사용자의 보안 암호에 대한 요청을 허용(또는 거부)합니다. 데이터 API와 함께 사용할 보안 암호 설정에 대한 자세한 내용은 AWS Secrets Manager에 데이터베이스 자격 증명 저장 섹션을 참조하세요.

`AmazonRDSDataFullAccess` 정책은 데이터 API를 통해 리소스에 대한 완전한 액세스를 제공합니다. 리소스의 Amazon 리소스 이름(ARN)을 지정하는 사용자 고유의 정책을 정의하여 범위를 좁힐 수 있습니다.

예를 들어, 다음 정책은 사용자가 ARN으로 식별되는 DB 클러스터의 데이터 API에 액세스하는 데 필요한 최소 권한을 보여줍니다. 정책에는 Secrets Manager에 액세스하고 사용자의 DB 인스턴스에 대한 권한 부여를 얻는 데 필요한 권한이 포함되어 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
```

```
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

예제와 같이 정책 문에서 와일드카드(*) 대신 “Resources” 요소에 대한 특정 ARN을 사용하는 것이 좋습니다.

태그 기반 권한 부여 작업

RDS 데이터 API(데이터 API) 및 Secrets Manager 모두 태그 기반 권한 부여를 지원합니다. 태그는 RDS 클러스터와 같은 리소스에 추가 문자열 값을 사용하여 레이블을 지정하는 키-값 쌍입니다. 예를 들면 다음과 같습니다.

- `environment:production`
- `environment:development`

비용 할당, 운영 지원, 액세스 제어 및 기타 여러 이유로 리소스에 태그를 적용할 수 있습니다. 리소스에 아직 태그가 없는 상태에서 태그를 적용하려는 경우 Amazon RDS 리소스 태그 지정에서 자세한 내용을 확인할 수 있습니다. 정책 문의 태그를 사용하여 이러한 태그로 레이블이 지정된 RDS 클러스터에 대한 액세스를 제한할 수 있습니다. 예를 들어 Aurora DB 클러스터에는 환경을 프로덕션 또는 개발로 식별하는 태그가 있을 수 있습니다.

다음 예제에서는 정책 문에서 태그를 사용하는 방법을 보여줍니다. 이 문을 사용하려면 클러스터와 데이터 API 요청에 전달된 보안 암호에 `environment:production` 태그가 있어야 합니다.

정책이 적용되는 방법은 다음과 같습니다. 사용자가 데이터 API를 사용하여 호출하면 요청이 서비스로 전송됩니다. 데이터 API는 먼저 요청에서 전달된 클러스터 ARN에 `environment:production`으로

태그가 지정되어 있는지 확인합니다. 그런 다음 Secrets Manager를 호출하여 요청에서 사용자의 비밀 값을 검색합니다. Secrets Manager는 또한 `environment:production`이 사용자의 비밀에 태깅되었는지 확인됩니다. 그렇다면 데이터 API는 사용자의 DB 암호에 대해 검색된 값을 사용합니다. 마지막으로, 올바른 경우 사용자에 대해 데이터 API 요청이 성공적으로 호출됩니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
              ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                         "production"
                                        ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
              ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                         "production"
                                        ]
                     }
             }
         }
     ]
}
```

이 예제에서는 데이터 API 및 Secrets Manager의 `rds-data` 및 `secretsmanager`에 대한 별도의 작업을 보여줍니다. 그러나 여러 가지 방법으로 작업을 결합하고 태그 조건을 정의하여 특정 사용 사례를 지원할 수 있습니다. 자세한 내용은 Secrets Manager에 대한 자격 증명 기반 정책(IAM 정책) 사용을 참조하십시오.

정책의 “Condition” 요소에 대해 다음 중에서 태그 키를 선택할 수 있습니다.

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

리소스 태그 및 `aws:TagKeys` 사용 방법에 대한 자세한 내용은 리소스 태그를 사용하여 AWS 리소스에 대한 액세스 제어를 참조하십시오.

> **Note**
>
> 데이터 API 및 AWS Secrets Manager 모두 사용자에게 권한을 부여합니다. 정책에 정의된 모든 작업에 대한 권한이 없는 경우 `AccessDeniedException` 오류가 발생합니다.

AWS Secrets Manager에 데이터베이스 자격 증명 저장

Amazon RDS Data API(Data API)를 직접적으로 호출할 때 Secrets Manager에서 보안 암호를 사용하여 Aurora DB 클러스터에 대한 자격 증명을 전달합니다. 이 방식으로 자격 증명을 전달하려면 보안 암호의 이름 또는 보안 암호의 Amazon 리소스 이름(ARN)을 지정합니다.

보안 암호에 DB 클러스터 자격 증명을 저장하려면

1. Secrets Manager을 사용하여 Aurora DB 클러스터의 자격 증명을 포함하는 보안 암호를 생성합니다.

   이에 관한 지침은 AWS Secrets Manager 사용 설명서의 데이터베이스 암호 생성에서 확인하세요.
2. Secrets Manager 콘솔을 사용하여 생성한 비밀에 대한 세부 정보를 보거나 `aws secretsmanager describe-secret` AWS CLI 명령을 실행합니다.

   보안 암호의 이름 및 ARN을 적어둡니다. 이러한 이름이나 ARN은 데이터 API 호출에서 사용할 수 있습니다.

Secrets Manager 사용에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서를 참조하세요.

Amazon Aurora이 자격 증명 및 액세스를 관리하는 방법을 이해하려면 IAM에서 Amazon Aurora을 사용하는 방식을 참조하십시오.

IAM 정책 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오. 사용자에게 IAM 정책 추가에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하십시오.

Amazon RDS Data API 활성화

Amazon RDS Data API(Data API)를 사용하려면 Aurora DB 클러스터에 대해 해당 API를 활성화하세요. DB 클러스터를 생성하거나 수정할 때 데이터 API를 활성화할 수 있습니다.

> **Note**
>
> 클러스터에 데이터 API를 사용할 수 있는지 여부는 Aurora 버전, 데이터베이스 엔진 및 AWS 리전에 따라 달라집니다. 이전 Aurora 버전의 경우 데이터 API는 Aurora Serverless v1 클러스터에서만 작동합니다. 최신 Aurora 버전의 경우 데이터 API는 프로비저닝된 인스턴스와 Aurora Serverless v2 인스턴스를 모두 사용하는 클러스터에서 작동합니다. 클러스터가 데이터 API를 사용할 수 있는지 확인하려면 RDS Data API를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

주제

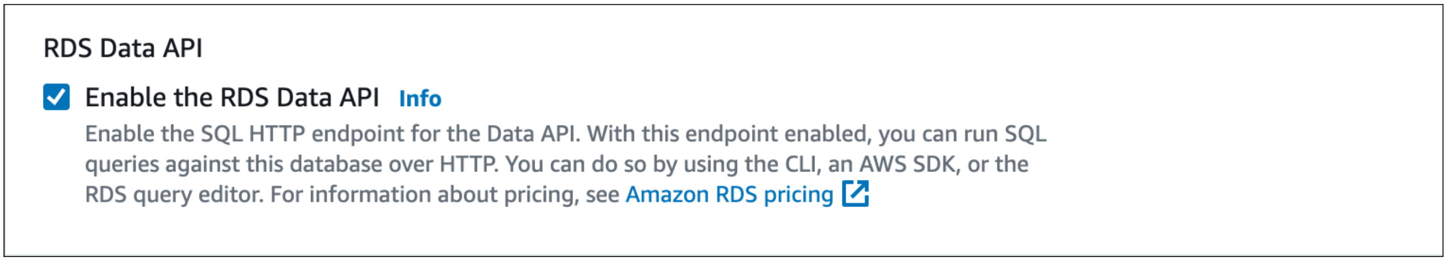
- 데이터베이스를 생성할 때 RDS 데이터 API 활성화
- 기존 데이터베이스에서 RDS 데이터 API 활성화

데이터베이스를 생성할 때 RDS 데이터 API 활성화

RDS 데이터 API(데이터 API)를 지원하는 데이터베이스를 생성하는 동안 이 기능을 활성화할 수 있습니다. 다음 절차에서는 AWS Management Console, AWS CLI 또는 RDS API를 사용할 때 해당 작업을 수행하는 방법을 설명합니다.

콘솔

DB 클러스터를 생성할 때 데이터 API를 활성화하려면 다음 스크린샷과 같이 데이터베이스 생성 페이지의 연결 섹션에서 RDS 데이터 API 활성화 확인란을 선택합니다.



RDS 데이터 API를 사용할 수 있는 Aurora DB 클러스터를 생성하는 방법에 대한 지침은 다음을 참조하세요.

- Aurora Serverless v2 및 프로비저닝된 클러스터의 경우 - Amazon Aurora DB 클러스터 생성
- Aurora Serverless v1의 경우 – Aurora Serverless v1 DB 클러스터 생성

AWS CLI

Aurora DB 클러스터를 생성하는 동안 데이터 API를 활성화하려면 `--enable-http-endpoint` 옵션과 함께 create-db-cluster AWS CLI 명령을 실행합니다.

다음 예제에서는 데이터 API가 활성화된 Aurora PostgreSQL DB 클러스터를 생성합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

RDS API

Aurora DB 클러스터를 생성하는 동안 데이터 API를 활성화하려면 `EnableHttpEndpoint` 파라미터 값을 `true`로 설정한 상태에서 CreateDBCluster 작업을 사용하세요.

기존 데이터베이스에서 RDS 데이터 API 활성화

RDS 데이터 API(데이터 API)를 지원하는 DB 클러스터를 수정하여 이 기능을 활성화하거나 비활성화할 수 있습니다.

주제

- 데이터 API 활성화 또는 비활성화(Aurora Serverless v2 및 프로비저닝)
- 데이터 API 활성화 또는 비활성화(Aurora Serverless v1 전용)

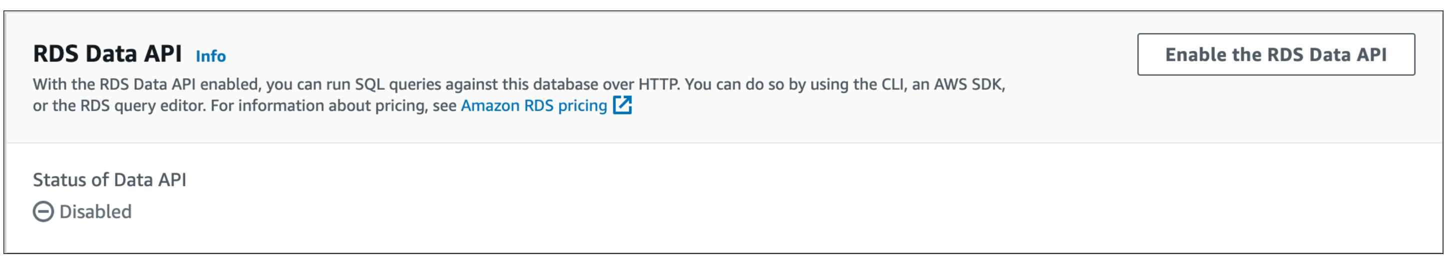
데이터 API 활성화 또는 비활성화(Aurora Serverless v2 및 프로비저닝)

다음 절차를 사용하여 Aurora Serverless v2 및 프로비저닝된 데이터베이스에서 데이터 API를 활성화 또는 비활성화합니다. Aurora Serverless v1 데이터베이스에서 데이터 API를 활성화하거나 비활성화하려면 the section called “데이터 API 활성화 또는 비활성화(Aurora Serverless v1 전용)”의 절차를 사용하세요.

콘솔

이 기능을 지원하는 DB 클러스터의 RDS 콘솔을 사용하여 데이터 API를 활성화 또는 비활성화할 수 있습니다. 이렇게 하려면 데이터 API를 활성화하거나 비활성화하려는 데이터베이스의 클러스터 세부 정보 페이지를 열고 연결 및 보안 탭에서 RDS 데이터 API 섹션으로 이동합니다. 이 섹션에는 데이터 API의 상태가 표시되며 이를 활성화하거나 비활성화할 수 있습니다.

다음 스크린샷은 활성화되지 않은 RDS 데이터 API를 보여줍니다.



AWS CLI

기존 데이터베이스에서 데이터 API를 활성화하거나 비활성화하려면 enable-http-endpoint 또는 disable-http-endpoint AWS CLI 명령을 실행하고 DB 클러스터의 ARN을 지정합니다.

다음 예제에서는 데이터 API를 활성화합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

Windows의 경우:

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

RDS API

기존 데이터베이스에서 데이터 API를 활성화하거나 비활성화하려면 EnableHttpEndpoint 및 DisableHttpEndpoint 작업을 사용합니다.

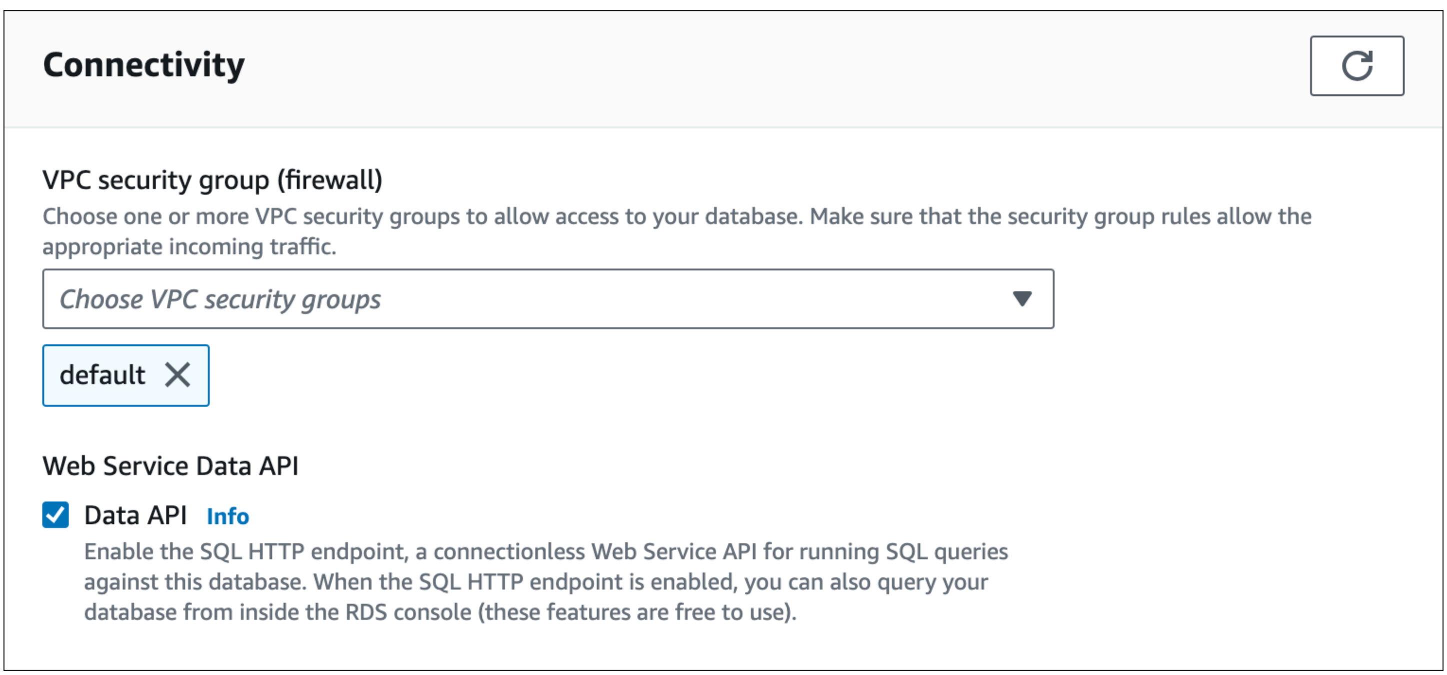
데이터 API 활성화 또는 비활성화(Aurora Serverless v1 전용)

다음 절차를 사용하여 기존 Aurora Serverless v1 데이터베이스에서 데이터 API를 활성화 또는 비활성화합니다. Aurora Serverless v2 및 프로비저닝된 데이터베이스에서 데이터 API를 활성화하거나 비활성화하려면 the section called “데이터 API 활성화 또는 비활성화”의 절차를 사용하세요.

콘솔

Aurora Serverless v1 DB 클러스터를 수정할 때 RDS 콘솔의 연결 섹션에서 데이터 API를 활성화합니다.

다음 스크린샷은 Aurora DB 클러스터를 수정할 때 활성화된 데이터 API를 보여줍니다.



Aurora Serverless v1 DB 클러스터를 수정하는 방법에 대한 지침은 Aurora Serverless v1 DB 클러스터 수정 섹션을 참조하세요.

AWS CLI

데이터 API를 활성화하거나 비활성화하려면 해당하는 경우 `--enable-http-endpoint` 또는 `--no-enable-http-endpoint`와 함께 modify-db-cluster AWS CLI 명령을 실행합니다.

다음 예제에서는 `sample-cluster`에서 데이터 API를 활성화합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --enable-http-endpoint
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

RDS API

데이터 API를 활성화하려면 ModifyDBCluster 작업을 사용하고 해당하는 경우 `EnableHttpEndpoint` 의 값을 `true` 또는 `false`로 설정합니다.

Amazon RDS Data API에 대한 Amazon VPC 엔드포인트(AWS PrivateLink) 만들기

Amazon VPC를 사용하면 Aurora DB 클러스터 및 애플리케이션과 같은 AWS 리소스를 Virtual Private Cloud(VPC)에서 시작할 수 있습니다. AWS PrivateLink는 Amazon 네트워크에서 VPC 및 AWS 서비스 간의 프라이빗 연결을 안전하게 제공합니다. AWS PrivateLink를 사용하면 Amazon VPC 엔드포인트를 생성하여 Amazon VPC 기반의 다른 계정 및 VPC에서 서비스에 연결할 수 있습니다. AWS PrivateLink에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC 엔드포인트 서비스(AWS PrivateLink)를 참조하세요.

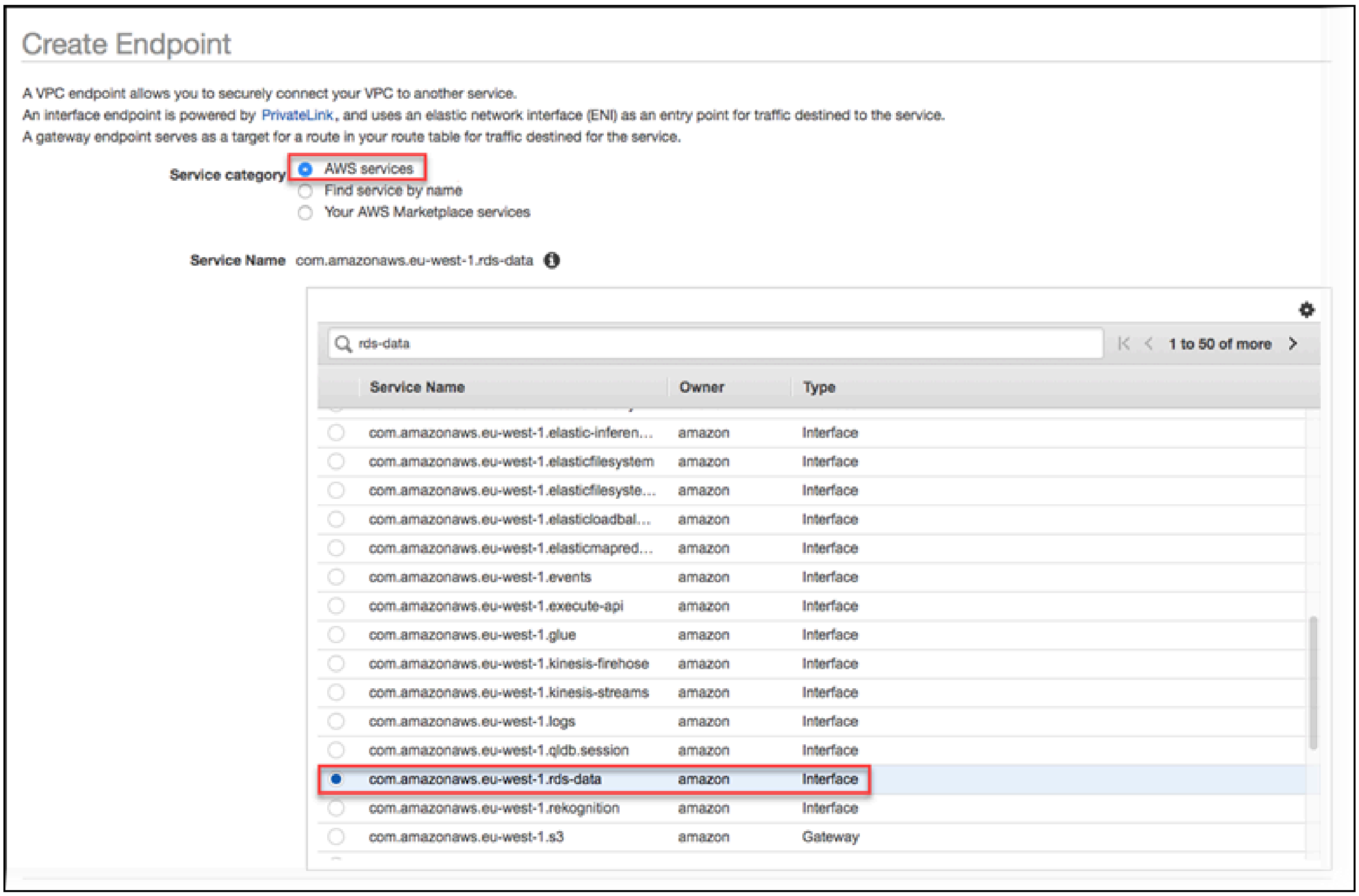
Amazon VPC 엔드포인트를 사용하여 RDS 데이터 API(데이터 API)를 호출할 수 있습니다. Amazon VPC 엔드포인트를 사용하면 퍼블릭 IP 주소를 사용하지 않고 Amazon VPC의 애플리케이션과 AWS

네트워크의 데이터 API 간에 트래픽을 유지합니다. Amazon VPC 엔드포인트를 사용하면 퍼블릭 인터넷 연결 제한과 관련된 규정 준수 및 규정 요구 사항을 충족할 수 있습니다. 예를 들어, Amazon VPC 엔드포인트를 사용하는 경우 Amazon EC2 인스턴스에서 실행되는 애플리케이션과 해당 애플리케이션이 포함된 VPC의 데이터 API 간 트래픽을 유지할 수 있습니다.

Amazon VPC 엔드포인트를 생성한 후에는 애플리케이션에서 코드나 구성을 변경하지 않고 엔드포인트를 사용할 수 있습니다.

데이터 API에 대한 Amazon VPC 엔드포인트를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. 엔드포인트를 선택한 다음 엔드포인트 생성을 선택합니다.
3. 엔드포인트 생성 페이지에서 서비스 범주에 대해 AWS 서비스를 선택합니다. 서비스 이름에 대해 rds-data를 선택합니다.



4. VPC의 경우 엔드포인트를 생성할 VPC를 선택합니다.

   데이터 API를 호출하는 애플리케이션이 포함된 VPC를 선택합니다.

5. 서브넷의 경우 애플리케이션을 실행 중인 AWS 서비스에서 사용하는 각 가용 영역(AZ)의 서브넷을 선택합니다.

Amazon VPC 엔드포인트를 생성하려면 엔드포인트에 액세스할 수 있는 프라이빗 IP 주소 범위를 지정합니다. 이렇게 하려면 각 가용 영역에 대한 서브넷을 선택합니다. 이렇게 하면 VPC 엔드포인트가 각 가용 영역별 프라이빗 IP 주소 범위로 제한되고 각 가용 영역에 Amazon VPC 엔드포인트가 생성됩니다.

6. Enable DNS name(DNS 이름 활성화)에서 이 엔드포인트에 대해 활성화를 선택합니다.

프라이빗 DNS는 표준 데이터 API DNS 호스트 이름(https://rds-data.*region*.amazonaws.com)을 Amazon VPC 엔드포인트에 특정한 DNS 호스트 이름과 연결된 프라이빗 IP 주소로 확인합니다. 따라서 데이터 API 엔드포인트 URL을 업데이트하기 위해 코드나 구성을 변경하지 않고도 AWS CLI 또는 AWS SDK를 사용하여 데이터 API VPC 엔드포인트에 액세스할 수 있습니다.

7. 보안 그룹의 경우 Amazon VPC 엔드포인트와 연결할 보안 그룹을 선택합니다.

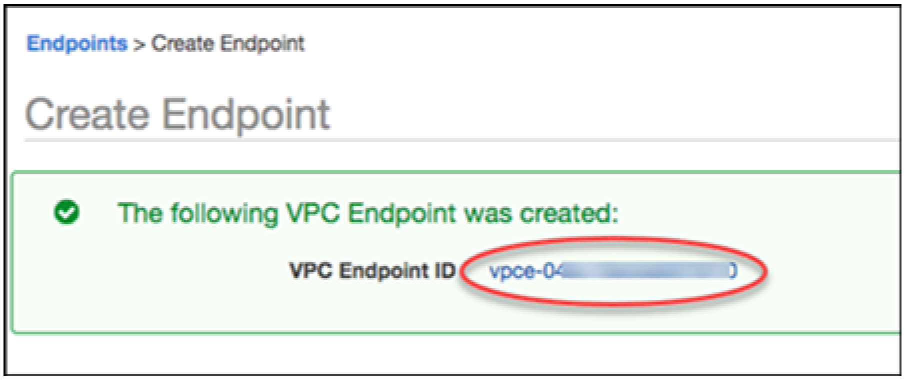
애플리케이션을 실행 중인 AWS 서비스에 대한 액세스를 허용하는 보안 그룹을 선택합니다. 예를 들어 Amazon EC2 인스턴스가 애플리케이션을 실행 중인 경우 Amazon EC2 인스턴스에 대한 액세스를 허용하는 보안 그룹을 선택합니다. 보안 그룹을 사용하면 VPC의 리소스에서 Amazon VPC 엔드포인트로 가는 트래픽을 제어할 수 있습니다.

8. 정책의 경우 모든 액세스를 선택하여 Amazon VPC에 있는 모든 사용자가 이 엔드포인트를 통해 데이터 API에 액세스할 수 있도록 허용합니다. 또는 사용자 지정을 선택하여 액세스를 제한하는 정책을 지정합니다.

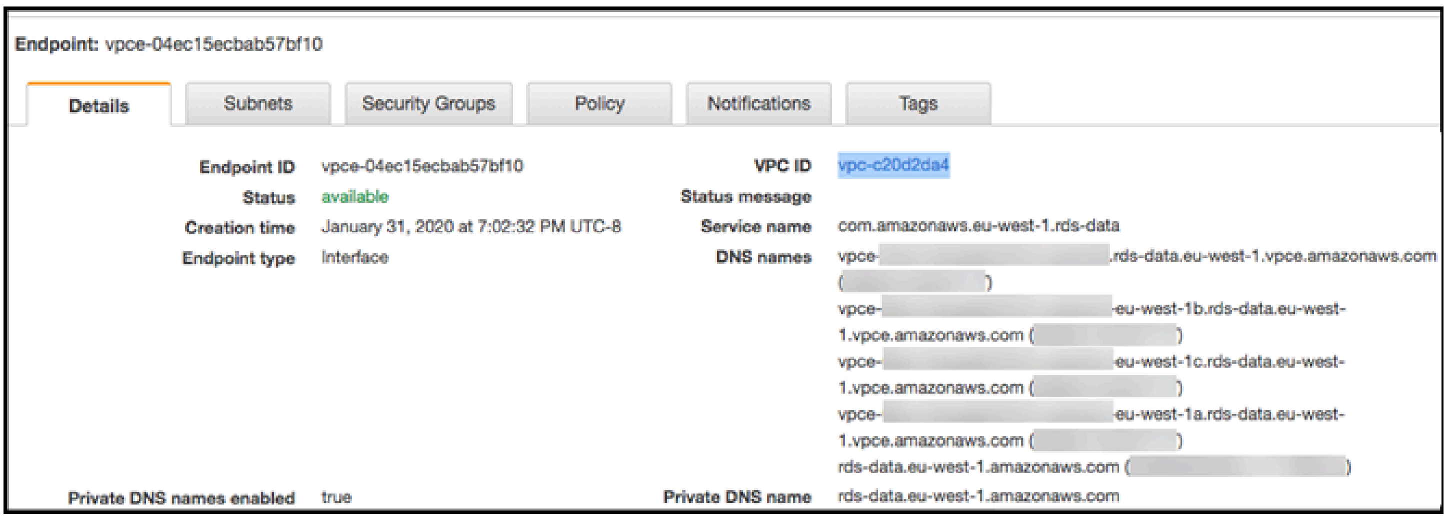
사용자 지정을 선택한 경우 정책 생성 도구에 정책을 입력합니다.

9. [Create endpoint]를 선택합니다.

엔드포인트를 생성한 후 AWS Management Console에서 링크를 선택하여 엔드포인트 세부 정보를 봅니다.



엔드포인트 세부 정보 탭에는 Amazon VPC 엔드포인트를 만드는 동안 생성된 DNS 호스트 이름이 표시됩니다.



표준 엔드포인트(`rds-data.`*`region`*`.amazonaws.com`) 또는 VPC 관련 엔드포인트 중 하나를 사용하여 Amazon VPC에서 데이터 API를 호출할 수 있습니다. 표준 데이터 API 엔드포인트는 자동으로 Amazon VPC 엔드포인트로 라우팅됩니다. 이 라우팅은 Amazon VPC 엔드포인트를 생성할 때 프라이빗 DNS 호스트 이름을 활성화했기 때문에 발생합니다.

데이터 API 호출에서 Amazon VPC 엔드포인트를 사용하는 경우 애플리케이션과 데이터 API 간의 모든 트래픽은 해당 트래픽이 포함된 Amazon VPC에 남아 있습니다. 모든 유형의 데이터 API 호출에 Amazon VPC 엔드포인트를 사용할 수 있습니다. 데이터 API 호출에 대한 자세한 내용은 Amazon RDS Data API 직접 호출 섹션을 참조하세요.

Amazon RDS Data API 직접 호출

Aurora DB 클러스터에서 Amazon RDS Data API(Data API)를 활성화하면 Data API 또는 AWS CLI를 사용하여 Aurora DB 클러스터에서 SQL 문을 실행할 수 있습니다. 데이터 API는 AWS SDK에서 지원되는 프로그래밍 언어를 지원합니다. 자세한 내용은 AWS 기반 구축 도구를 참조하세요.

주제

Amazon RDS Data API 작업 참조

Amazon RDS Data API는 SQL 문을 수행하는 다음 작업을 제공합니다.

데이터 API 작업	AWS CLI 명령	설명
`ExecuteStatement`	`aws rds-data execute-statement`	데이터베이스에서 SQL 문을 실행합니다.
`BatchExecuteStatement`	`aws rds-data batch-execute-statement`	대량 업데이트 및 삽입 작업을 위해 데이터 배열을 통해 일괄 SQL 문을 실행합니다. 파라미터 집합 배열을 사용하여 데이터 조작 언어(DML) 문을 실행할 수 있습니다. 일괄 SQL 문은 개별 삽입 및 업데이트 문을 통해 중요한 성능 개선을 제공합니다.

두 작업 중 하나를 사용하여 개별 SQL 문을 실행하거나 트랜잭션을 실행할 수 있습니다. 트랜잭션의 경우 데이터 API는 다음과 같은 작업을 제공합니다.

데이터 API 작업	AWS CLI 명령	설명
[BeginTransaction](#)	[aws rds-data begin-transaction](#)	SQL 트랜잭션을 시작합니다.
[CommitTransaction](#)	[aws rds-data commit-transaction](#)	SQL 트랜잭션을 종료하고 변경을 수행합니다.
[RollbackTransaction](#)	[aws rds-data rollback-transaction](#)	트랜잭션의 롤백을 수행합니다.

SQL 문을 수행하고 트랜잭션을 지원하기 위한 작업에는 다음과 같은 일반적인 데이터 API 파라미터와 AWS CLI 옵션이 있습니다. 일부 작업은 다른 파라미터 또는 옵션을 지원합니다.

데이터 API 작업 파라미터	AWS CLI 명령 옵션	필수	설명
`resourceArn`	`--resource-arn`	예	Aurora DB 클러스터의 Amazon 리소스 이름(ARN)입니다. 클러스터는 데이터 API를 간접적으로 호출하는 IAM 역할 또는 사용자와 동일한 AWS 계정에 있어야 합니다. 다른 계정의 클러스터에 액세스하려면 해당 계정에서 역할을 수임합니다.
`secretArn`	`--secret-arn`	예	DB 클러스터에 대한 액세스를 활성화하는 보안 암호의 이름 또는 ARN.

RDS 데이터 API는 Aurora MySQL에 대해 다음 데이터 유형을 지원합니다.

- `TINYINT(1)`, `BOOLEAN`, `BOOL`
- `TINYINT`
- `SMALLINT` [`SIGNED` | `UNSIGNED`]

- MEDIUMINT [SIGNED | UNSIGNED]
- INT [SIGNED | UNSIGNED]
- BIGINT [SIGNED | UNSIGNED]
- FLOAT
- DOUBLE
- VARCHAR, CHAR, TEXT, ENUM
- VARBINARY, BINARY, BLOB
- DATE, TIME, DATETIME, TIMESTAMP
- DECIMAL
- JSON
- BIT, BIT(N)

RDS 데이터 API는 다음 Aurora PostgreSQL 스칼라 유형을 지원합니다.

- BOOL
- BYTEA
- DATE
- CIDR
- DECIMAL, NUMERIC
- ENUM
- FLOAT8, DOUBLE PRECISION
- INET
- INT, INT4, SERIAL
- INT2, SMALLINT, SMALLSERIAL
- INT8, BIGINT, BIGSERIAL
- JSONB, JSON
- REAL, FLOAT
- TEXT, CHAR(N), VARCHAR, NAME
- TIME
- TIMESTAMP

- UUID
- VECTOR

RDS 데이터 API는 다음 Aurora PostgreSQL 배열 유형을 지원합니다.

- BOOL[], BIT[]
- DATE[]
- DECIMAL[], NUMERIC[]
- FLOAT8[], DOUBLE PRECISION[]
- INT[], INT4[]
- INT2[]
- INT8[], BIGINT[]
- JSON[]
- REAL[], FLOAT[]
- TEXT[], CHAR(N)[], VARCHAR[], NAME[]
- TIME[]
- TIMESTAMP[]
- UUID[]

ExecuteStatement 및 BatchExecuteStatement에 대한 데이터 API 호출에서, 그리고 AWS CLI 명령 execute-statement 및 batch-execute-statement 실행 시 파라미터를 사용할 수 있습니다. 파라미터를 사용하려면 SqlParameter 데이터 형식에 이름-값 페어를 지정합니다. Field 데이터 형식으로 값을 지정하십시오. 다음 표는 JDBC(Java Database Connectivity) 데이터 형식을 Data API 호출에서 지정하는 데이터 형식에 매핑합니다.

JDBC 데이터 형식	데이터 API 데이터 형식
INTEGER, TINYINT, SMALLINT, BIGINT	LONG – , 또는 STRING
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING
BOOLEAN, BIT	BOOLEAN

JDBC 데이터 형식	데이터 API 데이터 형식
`BLOB`, `BINARY`, `LONGVARBINARY`, `VARBINARY`	`BLOB`
`CLOB`	`STRING`
다른 형식(날짜 및 시간과 관련된 형식 포함)	`STRING`

> **Note**
>
> 데이터베이스에서 반환된 `LONG` 값에 대한 Data API 호출에서 `STRING` 또는 `LONG` 데이터 형식을 지정할 수 있습니다. JavaScript로 작업할 때 발생할 수 있는 매우 큰 수의 정밀도를 잃지 않도록 하는 것이 좋습니다.

`DECIMAL` 및 `TIME` 등 특정 유형에는 데이터 API가 `String` 값을 데이터베이스에 올바른 유형으로 전달할 수 있도록 힌트가 필요합니다. 힌트를 사용하려면 `typeHint`에 대한 값을 `SqlParameter` 데이터 형식에 포함합니다. `typeHint`에 가능한 값은 다음과 같습니다.

- `DATE` – 해당되는 `String` 파라미터 값은 `DATE` 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 `YYYY-MM-DD`입니다.
- `DECIMAL` – 해당되는 `String` 파라미터 값은 `DECIMAL` 유형의 객체로 데이터베이스에 전송됩니다.
- `JSON` – 해당되는 `String` 파라미터 값은 `JSON` 유형의 객체로 데이터베이스에 전송됩니다.
- `TIME` – 해당되는 `String` 파라미터 값은 `TIME` 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 `HH:MM:SS[.FFF]`입니다.
- `TIMESTAMP` – 해당되는 `String` 파라미터 값은 `TIMESTAMP` 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 `YYYY-MM-DD HH:MM:SS[.FFF]`입니다.
- `UUID` – 해당되는 `String` 파라미터 값은 `UUID` 유형의 객체로 데이터베이스에 전송됩니다.

> **Note**
>
> 현재, 데이터 API는 범용 고유 식별자(UUID) 배열을 지원하지 않습니다.

> **Note**
>
> Amazon Aurora PostgreSQL의 경우 데이터 API는 항상 UTC 시간대로 Aurora PostgreSQL 데이터 형식 `TIMESTAMPTZ`를 반환합니다.

AWS CLI를 사용하여 Amazon RDS Data API 직접 호출

AWS CLI를 사용하여 RDS 데이터 API(데이터 API)를 호출할 수 있습니다.

다음 예제에서는 데이터 API용 AWS CLI를 사용합니다. 자세한 내용은 AWS CLI Data API 참조를 참조하세요.

각 예에서 DB 클러스터의 Amazon 리소스 이름(ARN)을 Aurora DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 Secrets Manager의 보안 암호의 ARN으로 보안 암호 ARN을 바꿉니다.

> **Note**
>
> AWS CLI는 응답을 JSON 형식으로 지정할 수 있습니다.

주제

- SQL 트랜잭션 시작
- SQL 문 실행
- 데이터 배열에서 일괄 SQL 문 실행
- SQL 트랜잭션 커밋
- SQL 트랜잭션 롤백

SQL 트랜잭션 시작

`aws rds-data begin-transaction` CLI 명령을 사용하여 SQL 트랜잭션을 시작할 수 있습니다. 이 호출은 트랜잭션 식별자를 반환합니다.

> **Important**
>
> 데이터 API 내에서 3분 안에 트랜잭션 ID를 사용하는 호출이 없는 경우 트랜잭션 시간이 초과됩니다. 트랜잭션이 커밋되기 전에 시간이 초과되면 데이터 API는 트랜잭션을 자동으로 롤백합니다.
>
> 트랜잭션 내의 MySQL 데이터 정의 언어(DDL) 문은 암시적 커밋을 발생시킵니다. `execute-statement` 옵션과 함께 별도의 `--continue-after-timeout` 명령으로 각 MySQL DDL 문을 실행하는 것이 좋습니다.

일반 옵션 외에도, 데이터베이스 이름을 제공하는 `--database` 옵션을 지정합니다.

예를 들어, 다음 CLI 명령은 SQL 트랜잭션을 시작합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret"
```

Windows의 경우:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret"
```

다음은 이 응답의 예입니다.

```
{
    "transactionId": "ABC1234567890xyz"
}
```

SQL 문 실행

`aws rds-data execute-statement` CLI 명령을 사용하여 SQL 문을 실행할 수 있습니다.

`--transaction-id` 옵션으로 트랜잭션 식별자를 지정하여 트랜잭션에서 SQL 문을 실행할 수 있습니다. `aws rds-data begin-transaction` CLI 명령을 사용하여 트랜잭션을 시작할 수 있습니다.

`aws rds-data commit-transaction` CLI 명령을 사용하여 트랜잭션을 끝내고 커밋할 수 있습니다.

> **Important**
>
> `--transaction-id` 옵션을 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

일반 옵션 외에도 다음 옵션을 지정하십시오.

- `--sql` (필수) – DB 클러스터에서 실행할 SQL 문.
- `--transaction-id` (선택 사항) – `begin-transaction` CLI 명령을 사용하여 시작된 트랜잭션의 식별자. SQL 문을 포함할 트랜잭션의 트랜잭션 ID를 지정하십시오.
- `--parameters` (선택 사항) – SQL 문의 파라미터.
- `--include-result-metadata | --no-include-result-metadata` (선택 사항) – 메타데이터를 결과에 포함할지 나타내는 값. 기본값은 `--no-include-result-metadata`입니다.
- `--database` (선택 사항) – 데이터베이스 이름.

  이전 요청에서 `--sql "use` *`database_name`*`;"`를 실행한 후 SQL 문을 실행하면 `--database` 옵션이 작동하지 않을 수 있습니다. `--sql "use` *`database_name`*`;"` 문을 실행하는 대신 `--database` 옵션을 사용하는 것이 좋습니다.
- `--continue-after-timeout | --no-continue-after-timeout`(선택 사항) - 호출이 데이터 API 시간 초과 간격인 45초를 초과한 후에도 문을 계속 실행할지 여부를 나타내는 값입니다. 기본값은 `--no-continue-after-timeout`입니다.

  데이터 정의 언어(DDL) 문에 대해서는 오류 및 데이터 구조 손상 가능성을 피하기 위해 호출 시간이 초과된 후 문을 계속 실행하는 것이 좋습니다.
- `--format-records-as "JSON"|"NONE"` - 결과 집합의 형식을 JSON 문자열로 지정할지를 결정하는 선택적 값입니다. 기본값은 `"NONE"`입니다. JSON 결과 집합 처리에 대한 사용 정보는 [JSON 형식의 Amazon RDS Data API 쿼리 결과 처리](#) 섹션을 참조하세요.

DB 클러스터는 각 호출에 대한 응답을 반환합니다.

> **Note**
>
> 응답 크기 제한은 1MiB입니다. 호출이 1MiB를 초과하는 응답 데이터를 반환하면 호출이 종료됩니다.

Aurora Serverless v1의 경우 초당 최대 요청 수는 1,000개입니다. 지원되는 다른 모든 데이터베이스의 경우 제한이 없습니다.

예를 들어, 다음 CLI 명령은 단일 SQL 문을 실행하고 결과에서 메타데이터를 생략합니다(기본값).

대상 LinuxmacOS, 또는Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

Windows의 경우:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

다음은 이 응답의 예입니다.

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
```

```
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
        ]
    ]
}
```

다음 CLI 명령은 `--transaction-id` 옵션을 지정하여 트랜잭션에서 단일 SQL 문을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Windows의 경우:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

다음은 이 응답의 예입니다.

```
{
    "numberOfRecordsUpdated": 1
}
```

다음 CLI 명령은 파라미터가 있는 단일 SQL 문을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Windows의 경우:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

다음은 이 응답의 예입니다.

```
{
    "numberOfRecordsUpdated": 1
}
```

다음 CLI 명령은 데이터 정의 언어(DDL) SQL 문을 실행합니다. DDL 문은 `job` 열을 `role` 열로 이름을 바꿉니다.

> **Important**
>
> DDL 문에 대해서는 호출 시간이 초과된 후에도 문을 계속 실행하는 것이 좋습니다. DDL 문 실행이 끝나기 전에 종료되면 오류가 발생하고 데이터 구조가 손상될 수 있습니다. 호출이 RDS 데이터 API 시간 초과 간격인 45초를 초과한 후에도 문을 계속 실행하려면 `--continue-after-timeout` 옵션을 지정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
```

```
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Windows의 경우:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

다음은 이 응답의 예입니다.

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

Note

generatedFields 데이터는 Aurora PostgreSQL에서 지원하지 않습니다. 생성된 필드의 값을 가져오려면 RETURNING 절을 사용하십시오. 자세한 내용은 PostgreSQL 설명서의 수정된 행에서 데이터 반환 단원을 참조하십시오.

데이터 배열에서 일괄 SQL 문 실행

aws rds-data batch-execute-statement CLI 명령을 사용하여 데이터 배열에 대해 일괄 SQL 문을 실행할 수 있습니다. 이 명령을 사용하여 일괄 가져오기 또는 업데이트 작업을 수행할 수 있습니다.

--transaction-id 옵션으로 트랜잭션 식별자를 지정하여 트랜잭션에서 SQL 문을 실행할 수 있습니다. aws rds-data begin-transaction CLI 명령을 사용하여 트랜잭션을 시작할 수 있습니다. aws rds-data commit-transaction CLI 명령을 사용하여 트랜잭션을 종료하고 커밋할 수 있습니다.

> **Important**
>
> `--transaction-id` 옵션을 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

일반 옵션 외에도 다음 옵션을 지정하십시오.

- `--sql` (필수) – DB 클러스터에서 실행할 SQL 문.

  > **Tip**
  >
  > MySQL 호환 문의 경우 `--sql` 파라미터 끝에 세미콜론을 포함하지 마세요. 후행 세미콜론을 사용하면 구문 오류가 발생할 수 있습니다.

- `--transaction-id` (선택 사항) – `begin-transaction` CLI 명령을 사용하여 시작된 트랜잭션의 식별자. SQL 문을 포함할 트랜잭션의 트랜잭션 ID를 지정하십시오.
- `--parameter-set` (선택 사항) – 일괄 처리를 위한 파라미터 집합.
- `--database` (선택 사항) – 데이터베이스 이름.

DB 클러스터는 호출에 대한 응답을 반환합니다.

> **Note**
>
> 파라미터 세트 수는 상한이 정해져 있지 않습니다. 그러나 데이터 API를 통해 전송된 HTTP 요청의 최대 크기는 4MiB입니다. 요청이 이 제한을 초과하면 데이터 API가 오류를 반환하고 요청을 처리하지 않습니다. 이 4MiB 제한에는 요청의 HTTP 헤더와 JSON 표기법의 크기가 포함됩니다. 따라서 포함할 수 있는 파라미터 세트의 수는 SQL 문의 크기 및 각 파라미터 세트의 크기와 같은 요인의 조합에 따라 달라집니다.
>
> 응답 크기 제한은 1MiB입니다. 호출이 1MiB를 초과하는 응답 데이터를 반환하면 호출이 종료됩니다.
>
> Aurora Serverless v1의 경우 초당 최대 요청 수는 1,000개입니다. 지원되는 다른 모든 데이터베이스의 경우 제한이 없습니다.

예를 들어, 다음 CLI 명령은 파라미터 세트를 이용해 데이터 배열에 대해 배치 SQL 문을 실행합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

Windows의 경우:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> `--parameter-sets` 옵션에 줄바꿈을 포함하지 마십시오.

SQL 트랜잭션 커밋

`aws rds-data commit-transaction` CLI 명령을 사용하여 `aws rds-data begin-transaction`으로 시작한 SQL 트랜잭션을 종료하고 변경 사항을 커밋할 수 있습니다.

일반 옵션 외에 다음 옵션을 지정하십시오.

- `--transaction-id` (필수) – `begin-transaction` CLI 명령을 사용하여 시작된 트랜잭션의 식별자. 종료하고 커밋할 트랜잭션의 트랜잭션 ID를 지정하십시오.

예를 들어 다음 CLI 명령은 SQL 트랜잭션을 끝내고 변경 내용을 커밋합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Windows의 경우:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

다음은 이 응답의 예입니다.

```
{
    "transactionStatus": "Transaction Committed"
}
```

SQL 트랜잭션 롤백

`aws rds-data rollback-transaction` CLI 명령을 사용하여 `aws rds-data begin-transaction`으로 시작한 SQL 트랜잭션을 롤백할 수 있습니다. 트랜잭션을 롤백하면 변경 내용이 취소됩니다.

> **Important**
> 트랜잭션 ID가 만기되었다면 트랜잭션이 자동으로 롤백된 것입니다. 이 경우 만기된 트랜잭션 ID를 지정하는 `aws rds-data rollback-transaction` 명령이 오류를 반환합니다.

일반 옵션 외에 다음 옵션을 지정하십시오.

- `--transaction-id` (필수) – `begin-transaction` CLI 명령을 사용하여 시작된 트랜잭션의 식별자. 롤백하려는 트랜잭션의 트랜잭션 ID를 지정하십시오.

예를 들어 다음 AWS CLI 명령은 SQL 트랜잭션을 롤백합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Windows의 경우:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

다음은 이 응답의 예입니다.

```
{
    "transactionStatus": "Rollback Complete"
    }
```

Python 애플리케이션에서 Amazon RDS Data API 직접 호출

Python 애플리케이션에서 Amazon RDS Data API(Data API)를 직접적으로 호출할 수 있습니다.

다음 예제에서는 AWS SDK for Python(Boto)을 사용합니다. Boto에 대한 자세한 내용은 AWS SDK for Python(Boto 3) 설명서를 참조하세요.

각 예에서 DB 클러스터의 Amazon 리소스 이름(ARN)을 Aurora DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 Secrets Manager의 보안 암호의 ARN으로 보안 암호 ARN을 바꿉니다.

주제

- SQL 쿼리 실행
- DML SQL 문 실행
- SQL 트랜잭션 실행

SQL 쿼리 실행

SELECT 문을 실행하고 Python 애플리케이션으로 결과를 가져올 수 있습니다.

다음 예는 SQL 쿼리를 실행합니다.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
```

```
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

DML SQL 문 실행

데이터 조작 언어(DML) 문을 실행하여 데이터베이스의 데이터를 삽입, 업데이트 또는 삭제할 수 있습니다. DML 문에 파라미터를 사용할 수도 있습니다.

> **Important**
>
> 호출이 `transactionID` 파라미터를 포함하지 않아서 트랜잭션의 일부가 아닌 경우 호출 결과는 자동으로 커밋됩니다.

다음 예제는 insert SQL 문을 실행하고 파라미터를 사용합니다.

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]
```

```
response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

SQL 트랜잭션 실행

SQL 트랜잭션을 시작하고 하나 이상의 SQL 문을 실행한 다음, 변경 사항을 Python 애플리케이션으로 커밋할 수 있습니다.

> **Important**
> 3분 안에 트랜잭션 ID를 사용하는 호출이 없는 경우 트랜잭션 시간이 초과됩니다. 커밋되기 전에 트랜잭션 시간이 초과되면 자동으로 롤백됩니다.
> 트랜잭션 ID를 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

다음 예에서는 테이블에 행을 삽입하는 SQL 트랜잭션을 실행합니다.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])
```

```
cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> 데이터 정의 언어(DDL) 문을 실행하는 경우 호출 시간이 초과된 후에도 문을 계속 실행하는 것이 좋습니다. DDL 문 실행이 끝나기 전에 종료되면 오류가 발생하고 데이터 구조가 손상될 수 있습니다. 호출이 RDS 데이터 API 시간 초과 간격인 45초를 초과한 후에도 문을 계속 실행하려면 `continueAfterTimeout` 파라미터를 `true`로 설정합니다.

Java 애플리케이션에서 Amazon RDS Data API 직접 호출

Java 애플리케이션에서 Amazon RDS Data API(Data API)를 직접적으로 호출할 수 있습니다.

다음 예제는 AWS SDK for Java를 사용합니다. 자세한 내용은 AWS SDK for Java 개발자 안내서를 참조하십시오.

각 예에서 DB 클러스터의 Amazon 리소스 이름(ARN)을 Aurora DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 Secrets Manager의 보안 암호의 ARN으로 보안 암호 ARN을 바꿉니다.

주제

- SQL 쿼리 실행
- SQL 트랜잭션 실행
- 일괄 SQL 작업 실행

SQL 쿼리 실행

`SELECT` 문을 실행하고 Java 애플리케이션으로 결과를 가져올 수 있습니다.

다음 예는 SQL 쿼리를 실행합니다.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
}
```

SQL 트랜잭션 실행

SQL 트랜잭션을 시작하고 하나 이상의 SQL 문을 실행한 다음, 변경 사항을 Java 애플리케이션으로 커밋할 수 있습니다.

⚠ Important

3분 안에 트랜잭션 ID를 사용하는 호출이 없는 경우 트랜잭션 시간이 초과됩니다. 커밋되기 전에 트랜잭션 시간이 초과되면 자동으로 롤백됩니다.
트랜잭션 ID를 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

다음 예는 SQL 트랜잭션을 실행합니다.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);
```

```
    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> 데이터 정의 언어(DDL) 문을 실행하는 경우 호출 시간이 초과된 후에도 문을 계속 실행하는 것이 좋습니다. DDL 문 실행이 끝나기 전에 종료되면 오류가 발생하고 데이터 구조가 손상될 수 있습니다. 호출이 RDS 데이터 API 시간 초과 간격인 45초를 초과한 후에도 문을 계속 실행하려면 `continueAfterTimeout` 파라미터를 `true`로 설정합니다.

일괄 SQL 작업 실행

Java 애플리케이션을 사용하여 데이터 배열에 대해 대량 삽입 및 업데이트 작업을 실행할 수 있습니다. 파라미터 세트의 배열을 사용하여 DML 문을 실행할 수 있습니다.

> **Important**
>
> 트랜잭션 ID를 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

다음 예제에서는 대량 삽입 작업을 실행합니다.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
```

```
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

데이터 API 시간 초과 동작 제어

데이터 API에 대한 모든 호출은 동기식입니다. `INSERT` 또는 `CREATE TABLE`과 같은 SQL 문을 실행하는 데이터 API 작업을 수행한다고 가정해 보겠습니다. 데이터 API 호출이 성공적으로 반환되면 호출이 반환되는 시점에 SQL 처리가 완료됩니다.

기본적으로 Data API는 45초 이내에 처리가 완료되지 않으면 작업을 취소하고 시간 초과 오류를 반환합니다. 이 경우 데이터가 삽입되지 않고 테이블이 생성되지 않는 등의 문제가 발생합니다.

Data API를 사용하여 45초 이내에 완료할 수 없는 장기 실행 작업을 수행할 수 있습니다. 대용량 테이블에 대한 대량 `INSERT` 또는 DDL 작업과 같은 작업이 45초보다 오래 걸릴 것으로 예상되는 경우, `ExecuteStatement` 작업에 `continueAfterTimeout` 파라미터를 지정할 수 있습니다. 애플리케이

션에 여전히 시간 초과 오류가 발생합니다. 하지만 작업은 계속 실행되며 취소되지 않습니다. 문제 해결 예는 [SQL 트랜잭션 실행]을(를) 참조하세요.

사용 중인 프로그래밍 언어용 AWS SDK에 API 호출 또는 HTTP 소켓 연결에 대한 자체 시간 초과 기간이 있는 경우 이러한 시간 초과 기간이 모두 45초 이상인지 확인하세요. 일부 SDK의 경우 기본적으로 시간 초과 기간이 45초 미만입니다. SDK별 또는 클라이언트별 시간 초과 기간을 최소 1분으로 설정하는 것이 좋습니다. 이렇게 하면 데이터 API 작업이 성공적으로 완료되는 동안 애플리케이션에 시간 초과 오류가 발생할 가능성을 방지할 수 있습니다. 이렇게 하면 작업을 다시 시도할지 여부를 확인할 수 있습니다.

예를 들어 SDK가 애플리케이션에 시간 초과 오류를 반환하지만 데이터 API 작업이 여전히 데이터 API 시간 초과 간격 내에 완료되었다고 가정해 보겠습니다. 이 경우 작업을 다시 시도하면 중복 데이터가 삽입되거나 잘못된 결과가 발생할 수 있습니다. SDK가 자동으로 작업을 다시 시도하여 애플리케이션에서 아무런 조치를 취하지 않아도 잘못된 데이터가 발생할 수 있습니다.

시간 초과 간격은 Java 2 SDK에서 특히 중요합니다. 해당 SDK에서 API 호출 시간 초과와 HTTP 소켓 시간 초과 모두 기본적으로 30초입니다. 다음은 이러한 시간 제한을 더 높은 값으로 설정하는 예제입니다.

```
public RdsDataClient createRdsDataClient() {
    return RdsDataClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .httpClientBuilder(createHttpClientBuilder())
        .credentialsProvider(defaultCredentialsProvider()) // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return ApacheHttpClient.builder() // Change this to your desired HttpClient
        .socketTimeout(Duration.ofSeconds(60));
}
```

다음은 비동기 데이터 클라이언트를 사용한 동일한 예제입니다.

```
public static RdsDataAsyncClient createRdsDataAsyncClient() {
    return RdsDataAsyncClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .credentialsProvider(defaultCredentialsProvider())  // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallAttemptTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return NettyNioAsyncHttpClient.builder() // Change this to your desired
 AsyncHttpClient
        .readTimeout(Duration.ofSeconds(60));
}
```

RDS 데이터 API용 Java 클라이언트 라이브러리 사용

RDS 데이터 API(데이터 API)용 Java 클라이언트 라이브러리를 다운로드하여 사용할 수 있습니다. 이 Java 클라이언트 라이브러리는 데이터 API를 사용할 수 있는 다른 방법을 제공합니다. 이 라이브러리를 사용하면 클라이언트 측 클래스를 데이터 API 요청 및 응답에 매핑할 수 있습니다. 이러한 매핑 지원을 통해 `Date`, `Time`, `BigDecimal` 등 일부 특정 Java 유형과 쉽게 통합할 수 있습니다.

데이터 API용 Java 클라이언트 라이브러리 다운로드

Data API Java 클라이언트 라이브러리는 다음 위치의 GitHub에서 오픈 소스로 제공됩니다.

https://github.com/awslabs/rds-data-api-client-library-java

소스 파일에서 라이브러리를 수동으로 빌드할 수 있지만 모범 사례는 Apache Maven 종속성 관리를 사용해 라이브러리를 사용하는 것입니다. Maven POM 파일에 다음 종속성을 추가하세요.

AWS SDK 2.x와 호환되는 버전 2.x의 경우 다음을 사용하세요.

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
```

```
    <version>2.0.0</version>
</dependency>
```

AWS SDK 1.x와 호환되는 버전 1.x의 경우 다음을 사용하세요.

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java 클라이언트 라이브러리 예시

아래에는 데이터 API Java 클라이언트 라이브러리 사용에 관한 몇 가지 일반적인 예시가 나와 있습니다. 이 예시에서는 accounts와 accountId이라는 두 개의 열이 있는 name라는 테이블이 있다고 가정합니다. 또한 다음과 같은 데이터 전송 객체(DTO)도 있습니다.

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

클라이언트 라이브러리를 통해 DTO를 입력 파라미터로 전달할 수 있습니다. 다음 예시에서는 고객 DTO가 입력 파라미터 세트로 매핑되는 방식을 보여줍니다.

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
         .execute();
```

어떤 경우에는 입력 파라미터인 간단한 값으로 작업하는 것이 더 쉽습니다. 다음 구문을 사용하면 이러한 작업이 가능합니다.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParameter("accountId", 3)
         .withParameter("name", "Zhang")
         .execute();
```

다음은 입력 파라미터인 간단한 값으로 작업하는 또 다른 예시입니다.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
         .execute();
```

클라이언트 라이브러리에서는 결과가 반환될 때 DTO에 대한 자동 매핑을 제공합니다. 다음 예시에서는 결과가 DTO에 매핑되는 방식을 보여줍니다.

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

대부분의 경우 데이터베이스 결과 집합에는 단일 값만 포함됩니다. 이러한 결과 검색을 단순화하기 위해 클라이언트 라이브러리는 다음 API를 제공합니다.

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

Note

`mapToList` 함수는 SQL 결과 집합을 사용자 정의 객체 목록으로 변환합니다. Java 클라이언트 라이브러리에 대한 `ExecuteStatement` 호출에 이 `.withFormatRecordsAs(RecordsFormatType.JSON)` 문을 사용할 수 없습니다. 동일한 용도로 사용되기 때문입니다. 자세한 내용은 [JSON 형식의 Amazon RDS Data API 쿼리 결과 처리](#) 섹션을 참조하세요.

JSON 형식의 Amazon RDS Data API 쿼리 결과 처리

`ExecuteStatement` 작업을 호출할 때 쿼리 결과를 JSON 형식의 문자열로 반환하도록 선택할 수 있습니다. 이렇게 하면 프로그래밍 언어의 JSON 구문 분석 기능을 사용하여 결과 집합을 해석하고 형식을 다시 지정할 수 있습니다. 그러면 결과 집합을 반복하고 각 열 값을 해석하는 추가 코드를 작성하지 않아도 됩니다.

결과 집합을 JSON 형식으로 요청하려면 선택적으로 `formatRecordsAs` 파라미터를 `JSON` 값과 함께 전달합니다. JSON 형식의 결과 집합이 `ExecuteStatementResponse` 구조의 `formattedRecords` 필드에 반환됩니다.

`BatchExecuteStatement` 작업은 결과 집합을 반환하지 않습니다. 따라서 JSON 옵션은 해당 작업에 적용되지 않습니다.

JSON 해시 구조에서 키를 사용자 정의하려면 결과 집합에서 열 별칭을 정의합니다. SQL 쿼리의 열 목록에 있는 `AS` 절을 사용하면 됩니다.

JSON 기능을 사용하여 결과 집합을 더 쉽게 읽도록 하고 내용을 언어별 프레임워크에 매핑할 수 있습니다. ASCII로 인코딩된 결과 집합의 볼륨이 기본 표현보다 크기 때문에 많은 수의 행이나 큰 열 값을 반환하여 애플리케이션에서 사용할 수 있는 것보다 많은 메모리를 사용하는 쿼리에 기본 표현을 선택할 수 있습니다.

주제

JSON 형식의 쿼리 결과 검색

결과 집합을 JSON 문자열로 수신하려면 `ExecuteStatement` 호출에 `.withFormatRecordsAs(RecordsFormatType.JSON)`을 포함하세요. 반환 값은 `formattedRecords` 필드에 JSON 문자열로 반환됩니다. 이 경우 `columnMetadata`는 `null`입니다. 열 레이블은 각 행을 나타내는 객체의 키입니다. 이 열 이름은 결과 집합의 각 행에서 반복됩니다. 열 값은 따옴표로 묶인 문자열, 숫자 값 또는 `true`, `false` 또는 `null`을 나타내는 특수 값입니다. 길이 제약 조건 및 숫자 및 문자열의 정확한 유형과 같은 열 메타데이터는 JSON 응답에 보존되지 않습니다.

`.withFormatRecordsAs()` 호출을 생략하거나 `NONE`의 파라미터를 지정하는 경우 결과 집합은 `Records` 및 `columnMetadata` 필드를 사용하여 이진 형식으로 반환됩니다.

데이터 유형 매핑

결과 집합의 SQL 값은 더 작은 JSON 유형 집합에 매핑됩니다. 값은 JSON에서 문자열, 숫자 및 `true`, `false` 및 `null`과 같은 특수 상수로 표시됩니다. 프로그래밍 언어에 적합하도록 강한 타이핑 또는 약한 타이핑을 사용하여 이 값을 애플리케이션에서 변수로 변환할 수 있습니다.

JDBC 데이터 유형	JSON 데이터 유형
INTEGER, TINYINT, SMALLINT, BIGINT	기본적으로 숫자입니다. LongReturnType 옵션이 STRING으로 설정되어 있으면 문자열입니다.
FLOAT, REAL, DOUBLE	숫자
DECIMAL	기본적으로 문자열입니다. DecimalReturnType 옵션이 DOUBLE_OR_LONG 으로 설정되어 있으면 숫자입니다.
STRING	문자열
BOOLEAN, BIT	부울
BLOB, BINARY, VARBINARY , LONGVARBINARY	base64 인코딩의 문자열입니다.
CLOB	String
ARRAY	Array
NULL	null
다른 형식(날짜 및 시간과 관련된 형식 포함)	String

문제 해결

JSON 응답은 10MB로 제한됩니다. 응답이 이 한도보다 크면 프로그램에서 BadRequestException 오류를 받습니다. 이런 경우 다음 기법 중 하나를 사용하여 오류를 해결할 수 있습니다.

- 결과 집합에서 행의 수를 줄입니다. LIMIT 절을 추가하면 됩니다. LIMIT 및 OFFSET 절이 있는 여러 쿼리를 제출하여 큰 결과 집합을 여러 개의 작은 결과 집합으로 나눌 수 있습니다.

  결과 집합에 애플리케이션 로직별로 필터링된 행이 포함되어 있는 경우 WHERE 절에 조건을 더 추가하여 결과 집합에서 해당 행을 제거할 수 있습니다.
- 결과 집합에서 열의 수를 줄입니다. 쿼리의 선택 목록에서 항목을 제거하면 됩니다.

- 쿼리에서 열 별칭을 사용하여 열 레이블을 줄입니다. 각 열 이름은 결과 집합의 각 행에 대한 JSON 문자열에서 반복됩니다. 따라서 열 이름이 길고 행이 많은 쿼리 결과는 크기 제한을 초과할 수 있습니다. 특히 복잡한 표현식에 열 별칭을 사용하면 JSON 문자열에서 전체 표현식이 반복되는 것을 방지할 수 있습니다.
- SQL을 사용하면 열 별칭을 사용하여 동일한 이름의 열이 두 개 이상 있는 결과 집합을 생성할 수 있지만 JSON에서는 중복 키 이름이 허용되지 않습니다. 결과 집합을 JSON 형식으로 요청하고 둘 이상의 열의 이름이 같은 경우 RDS Data API에서 오류를 반환합니다. 따라서 모든 열 레이블의 이름이 고유하도록 해야 합니다.

예시

다음 Java 예제는 응답이 JSON 형식의 문자열인 `ExecuteStatement`를 호출한 후 결과 집합을 해석하는 방법을 보여줍니다. *databaseName*, *secretStoreArn* 및 *clusterArn* 파라미터를 적절한 값으로 대체합니다.

다음 Java 예제에서는 결과 집합에서 소수 값을 반환하는 쿼리를 보여줍니다. `assertThat` 호출은 응답 필드에 JSON 결과 집합에 대한 규칙을 기반으로 예상되는 속성이 있는지 테스트합니다.

이 예제는 다음 스키마 및 샘플 데이터에서 작동합니다.

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

이전 프로그램의 `formattedRecords` 필드 값은 다음과 같습니다.

```
[{"a":10.0}]
```

JSON 결과 집합이 있기 때문에 응답의 `Records` 및 `ColumnMetadata` 필드는 모두 Null입니다.

다음 Java 예제에서는 결과 집합에서 정수 값을 반환하는 쿼리를 보여줍니다. 예제는 `getFormattedRecords`를 호출하여 JSON 형식의 문자열만 반환하고 비어 있거나 Null인 다른 응답 필드는 무시합니다. 이 예제에서는 결과를 레코드 목록을 나타내는 구조로 역직렬화합니다. 각 레코드에는 이름이 결과 집합의 열 별칭에 해당하는 필드가 있습니다. 이 기법은 결과 집합을 구문 분석하는 코드를 단순화합니다. 애플리케이션에서 결과 집합의 행과 열을 반복하여 각 값을 적절한 유형으로 변환할 필요가 없습니다.

이 예제는 다음 스키마 및 샘플 데이터에서 작동합니다.

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

이전 프로그램의 `formattedRecords` 필드 값은 다음과 같습니다.

```
[{"a":17}]
```

결과 행 0의 a 열을 검색하려면 애플리케이션은 `recordsList.get(0).a`를 참조합니다.

반대로 다음 Java 예제에서는 JSON 형식을 사용하지 않을 때 결과 집합을 포함하는 데이터 구조를 구성하는 데 필요한 코드 종류를 보여줍니다. 이 경우 결과 집합의 각 행에는 단일 사용자에 대한 정보가 포함된 필드가 포함됩니다. 결과 집합을 나타내는 데이터 구조를 구축하려면 행을 반복해야 합니다. 각

행에 대해 코드는 각 필드의 값을 검색하고 적절한 형식 변환을 수행하며 행을 나타내는 객체의 해당 필드에 결과를 할당합니다. 그런 다음 코드는 각 사용자를 나타내는 객체를 전체 결과 집합을 나타내는 데이터 구조에 추가합니다. 결과 집합에서 필드를 재정렬, 추가 또는 제거하도록 쿼리가 변경된 경우 애플리케이션 코드도 변경해야 합니다.

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

다음 샘플 값은 열, 열 별칭 및 열 데이터 유형이 서로 다른 결과 집합에 대한 `formattedRecords` 필드의 값을 보여줍니다.

결과 집합에 여러 행이 포함된 경우 각 행은 배열 요소인 객체로 표시됩니다. 결과 집합의 각 열은 객체 내에서 키가 됩니다. 키는 결과 집합의 각 행에서 반복됩니다. 따라서 여러 행과 열로 구성된 결과 집합의 경우 전체 응답의 길이 제한을 초과하지 않도록 짧은 열 별칭을 정의해야 할 수 있습니다.

이 예제는 다음 스키마 및 샘플 데이터에서 작동합니다.

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

결과 집합의 열이 표현식으로 정의된 경우 표현식의 텍스트가 JSON 키가 됩니다. 따라서 일반적으로 쿼리의 선택 목록에서 각 표현식에 대한 설명 열 별칭을 정의하는 것이 편리합니다. 예를 들어, 다음 쿼리는 선택 목록에 함수 호출 및 산술 연산과 같은 표현식을 포함합니다.

```
select count(*), max(id), 4+7 from sample_names;
```

이러한 표현식은 키로 JSON 결과 집합을 통해 전달됩니다.

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

설명 레이블이 있는 AS 열을 추가하면 JSON 결과 집합에서 키를 보다 간단하게 해석할 수 있습니다.

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

수정된 SQL 쿼리의 경우, AS 절에 의해 정의된 열 레이블이 키 이름으로 사용됩니다.

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

JSON 문자열의 각 키 값 쌍의 값은 따옴표로 묶인 문자열일 수 있습니다. 문자열에는 유니코드 문자가 포함될 수 있습니다. 문자열에 이스케이프 시퀀스가 포함되어 있거나 " 또는 \문자가 포함되어 있는 경우 이러한 문자 앞에는 백슬래시 이스케이프 문자가 있습니다. 다음 JSON 문자열의 예제는 이러한 가능성을 보여줍니다. 예를 들어 `string_with_escape_sequences` 결과에는 특수 문자 백스페이스, 줄 바꿈, 캐리지 리턴, 탭, 양식 피드 및 \가 포함됩니다.

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

JSON 문자열의 각 키 값 쌍의 값은 숫자를 나타낼 수도 있습니다. 숫자는 정수, 부동 소수점 값, 음수 값 또는 지수 표기법으로 표현되는 값일 수 있습니다. 다음 JSON 문자열의 예제는 이러한 가능성을 보여줍니다.

```
[{"integer_value":17}]
[{"float_value":10.0}]
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

부울 값과 Null 값은 따옴표 없는 특수 키워드 `true`, `false` 및 `null`로 표현됩니다. 다음 JSON 문자열의 예제는 이러한 가능성을 보여줍니다.

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

BLOB 유형의 값을 선택하면 결과가 JSON 문자열에서 base64로 인코딩된 값으로 표시됩니다. 값을 원래 표현으로 다시 변환하려면 애플리케이션의 언어로 적절한 디코딩 기능을 사용할 수 있습니다.

예를 들어 Java에서는 `Base64.getDecoder().decode()` 함수를 호출합니다. 다음 샘플 출력은 `hello world`의 BLOB 값을 선택하고 결과 집합을 JSON 문자열로 반환한 것을 보여줍니다.

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

다음 Python 예제는 Python `execute_statement` 함수에 대한 호출의 결과에서 값에 액세스하는 방법을 보여줍니다. 결과 집합은 필드 `response['formattedRecords']`의 문자열 값입니다. 이 코드는 `json.loads` 함수를 호출하여 JSON 문자열을 데이터 구조로 변환합니다. 그러면 결과 집합의 각 행은 데이터 구조 내의 목록 요소이며 각 행 내에서 이름으로 결과 집합의 각 필드를 참조할 수 있습니다.

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

다음 JavaScript 예제는 JavaScript `executeStatement` 함수에 대한 호출의 결과에서 값에 액세스하는 방법을 보여줍니다. 결과 집합은 필드 `response.formattedRecords`의 문자열 값입니다. 이 코드는 `JSON.parse` 함수를 호출하여 JSON 문자열을 데이터 구조로 변환합니다. 그러면 결과 집합의 각 행은 데이터 구조 내의 배열 요소이며 각 행 내에서 이름으로 결과 집합의 각 필드를 참조할 수 있습니다.

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

Amazon RDS Data API 문제 해결

공통 오류 메시지로 제목이 지정된 다음 섹션은 Amazon RDS Data API(Data API)를 사용하면서 발생하는 문제를 해결하는 데 유용합니다.

주제

- [DatabaseErrorException: 트랜잭션이 여전히 쿼리 실행 중]
- [지원되지 않는 결과 예외]
- [다중 문은 지원되지 않음]
- [스키마 파라미터는 지원되지 않음]

트랜잭션 <transaction_ID>을 찾을 수 없음

위 오류 메시지는 데이터 API 호출 시 지정한 트랜잭션 ID를 찾을 수 없다는 것을 의미합니다. 이러한 문제가 발생하는 원인이 오류 메시지에 추가되며, 다음 중 한 가지로 표시됩니다.

- 트랜잭션이 만료될 수 있습니다.

  각 트랜잭션 호출은 지난 호출 이후 3분 이내에 실행되어야 합니다.

  지정된 트랜잭션 ID가 [BeginTransaction] 호출에 의해 생성되지 않았을 수도 있습니다. 호출 시 유효한 트랜잭션 ID를 지정해야 합니다.
- 하나의 이전 호출로 인해 트랜잭션이 종료되었습니다.

  `CommitTransaction` 또는 `RollbackTransaction` 호출에 의해 트랜잭션이 이미 종료되었습니다.
- 이전 호출의 오류로 인해 트랜잭션이 중단되었습니다.

  이전 호출에서 예외가 발생했는지 확인합니다.

트랜잭션 실행에 대한 자세한 내용은 [Amazon RDS Data API 직접 호출] 단원을 참조하세요.

쿼리 패킷이 너무 큽니다

위 오류 메시지는 행마다 반환되는 결과 집합이 너무 크다는 것을 의미합니다. 데이터 API는 데이터베이스에서 반환되는 결과 집합에서 각 행의 크기를 64KB로 제한합니다.

이 문제를 해결하려면 결과 집합의 각 행마다 크기를 64KB 이하로 유지해야 합니다.

데이터베이스 응답이 크기 제한을 초과했습니다

위 오류 메시지는 데이터베이스에서 반환되는 결과 집합의 크기가 너무 크다는 것을 의미합니다. Data API는 데이터베이스에서 반환되는 결과 집합의 크기를 1MiB로 제한합니다.

이러한 문제를 해결하려면 데이터 API 호출 시 반환되는 데이터 크기를 1MiB 이하로 유지해야 합니다. 반환해야 하는 결과 집합의 크기가 1MiB보다 크면 쿼리에서 다수의 `ExecuteStatement` 호출을 `LIMIT` 절과 함께 사용할 수 있습니다.

`LIMIT` 절에 대한 자세한 내용은 MySQL 설명서에서 SELECT 구문을 참조하세요.

HttpEndpoint가 클러스터 <cluster_ID>에서 활성화되지 않음

이 문제의 잠재적인 원인은 다음과 같습니다.

- Aurora DB 클러스터는 데이터 API를 지원하지 않습니다. RDS 데이터 API가 지원하는 DB 클러스터 유형에 대한 자세한 내용은 the section called “Amazon RDS Data API의 리전 및 버전 가용성” 섹션을 참조하세요.
- Aurora DB 클러스터에서 데이터 API가 활성화되어 있지 않습니다. Aurora DB 클러스터에서 데이터 API를 사용하려면 DB 클러스터에서 데이터 API를 먼저 활성화해야 합니다. 데이터 API 활성화에 대한 자세한 내용은 Amazon RDS Data API 활성화 섹션을 참조하세요.
- 데이터 API가 활성화된 후 DB 클러스터의 이름이 바뀌었습니다. 이 경우 해당 클러스터에 대한 데이터 API를 끄고 다시 활성화하세요.
- 지정한 ARN이 클러스터의 ARN과 정확하게 일치하지 않습니다. 다른 소스에서 반환되었거나 프로그램 논리로 구성된 ARN이 클러스터의 ARN과 정확히 일치하는지 확인합니다. 예를 들어 사용하는 ARN에서 모든 영문자의 대소문자가 올바른지 확인하세요.

DatabaseErrorException: 트랜잭션이 여전히 쿼리 실행 중

애플리케이션이 트랜잭션 ID로 요청을 보내고 해당 트랜잭션이 현재 다른 요청을 처리 중인 경우 데이터 API는 이 오류를 애플리케이션에 즉시 반환합니다. 이 조건은 애플리케이션이 Javascript의 'promises'와 같은 메커니즘을 사용하여 비동기식 요청을 하는 경우 발생할 수 있습니다.

이 문제를 해결하려면 이전 요청이 완료될 때까지 기다린 다음 요청을 다시 시도하세요. 오류가 더 이상 발생하지 않거나 애플리케이션에 몇 가지 다른 오류가 수신될 때까지 계속 재시도를 할 수 있습니다.

이 조건은 Aurora Serverless v2용 데이터 API 및 프로비저닝된 인스턴스에서 발생할 수 있습니다. Aurora Serverless v1용 데이터 API에서 동일한 트랜잭션 ID에 대한 후속 요청은 이전 요청이 완료될 때까지 자동으로 대기합니다. 그러나 이전 요청이 너무 오래 걸려서 이전 동작에 제한 시간 초과가 발생할 수 있습니다. 동시 요청을 하는 이전 데이터 API 애플리케이션을 이식하는 경우 예외 처리 로직을 수정하여 이 새로운 종류의 오류를 처리하세요.

지원되지 않는 결과 예외

데이터 API가 모든 데이터 유형을 지원하는 것은 아닙니다. 이 오류는 지원되지 않는 데이터 유형을 반환하는 쿼리를 실행할 때 발생합니다.

이 문제를 해결하려면 지원되지 않는 데이터 유형을 TEXT로 캐스팅합니다. 예시:

```
SELECT custom_type::TEXT FROM my_table;
-- OR
SELECT CAST(custom_type AS TEXT) FROM my_table;
```

다중 문은 지원되지 않음

Aurora Serverless v2 및 프로비저닝된 클러스터용 데이터 API에서는 다중 문이 지원되지 않습니다. 단일 API 직접 호출에서 다중 문을 실행하려고 하면 이 오류가 발생합니다.

다중 문을 실행하려면 별도의 `ExecuteStatement` API 직접 호출을 사용하거나 배치 처리를 위해 `BatchExecuteStatement` API를 사용합니다.

스키마 파라미터는 지원되지 않음

Aurora Serverless v1은 스키마 파라미터를 자동으로 무시합니다. 그러나 Aurora Serverless v2 및 프로비저닝된 클러스터는 스키마 파라미터를 포함하는 API 직접 호출을 명시적으로 거부합니다.

이 문제를 해결하려면 Aurora Serverless v2 또는 프로비저닝된 클러스터를 사용할 때 데이터 API에 대한 모든 직접 호출에서 스키마 파라미터를 제거합니다.

AWS CloudTrail을 사용하여 Amazon RDS Data API 직접 호출 로깅

RDS 데이터 API(데이터 API)는 사용자, 역할, AWS 서비스가 데이터 API에서 수행한 작업의 레코드를 제공하는 서비스, AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon RDS 콘솔의 호출과 데이터 API 작업에 대한 코드 호출의 호출을 포함하여 데이터 API에 대한 모든 API 호출을 이벤트로 캡처합니다. 추적을 생성하면 데이터 API 이벤트를 비롯하여 CloudTrail 이벤트를 Amazon S3 버킷으로 지속적으로 배포하도록 할 수 있습니다. CloudTrail에서 수집한 데이터를 사용하여 많은 정보를 확인할 수 있습니다. 이 정보에는 Data API로 한 요청, 요청한 IP 주소, 요청 주체, 요청 시점 및 추가 세부 정보가 포함됩니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서]를 참조하세요.

CloudTrail의 데이터 API 정보 작업

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 데이터 API에서 지원되는 활동(관리 이벤트)이 발생하면 해당 활동은 이벤트 기록에 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 관리 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 AWS CloudTrail 사용 설명서의 CloudTrail 이벤트 기록 작업을 참조하세요.

데이터 API의 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려는 경우 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 AWS 리전의 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 작업하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 다음 주제를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 수신 및 여러 계정에서 CloudTrail 로그 파일 수신

모든 데이터 API 작업은 CloudTrail이 기록하고 Amazon RDS데이터 서비스 API 참조에 문서화됩니다. 예를 들어 `BatchExecuteStatement`, `BeginTransaction`, `CommitTransaction`, `ExecuteStatement` 작업에 대한 호출은 CloudTrail 로그 파일의 항목을 생성합니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 사용자 자격 증명으로 했는지 여부
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

AWS CloudTrail 추적에서 데이터 API 이벤트 포함 및 제외

대부분의 데이터 API 사용자는 AWS CloudTrail 추적의 이벤트에 의존하여 데이터 API 작업에 대한 기록을 제공합니다 이벤트 데이터는 데이터 API에 대한 요청의 데이터베이스 이름, 스키마 이름 또는

SQL 문을 나타내지 않습니다. 하지만 구체적인 시간에 어떤 사용자가 특정 DB 클러스터에 대해 호출을 했는지 알면 비정상적인 액세스 패턴을 감지하는 데 도움이 될 수 있습니다.

AWS CloudTrail 추적에서 데이터 API 이벤트 포함

Aurora PostgreSQL Serverless v2 및 프로비저닝된 데이터베이스의 경우 다음과 같은 데이터 API 작업이 AWS CloudTrail에 데이터 이벤트로 로깅됩니다. 데이터 이벤트는 CloudTrail이 기본적으로 로깅하지 않는 대용량 데이터 영역 API 작업입니다. 데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

CloudTrail 콘솔, AWS CLI 또는 CloudTrail API 작업을 사용하여 이러한 데이터 API 작업을 로깅할 수 있습니다. CloudTrail 콘솔에서 데이터 이벤트 유형으로 RDS 데이터 API - DB 클러스터를 선택합니다. 자세한 내용을 알아보려면 AWS CloudTrail 사용 설명서의 AWS Management Console을 사용한 데이터 이벤트 로깅을 참조하세요.

AWS CLI를 통해 `aws cloudtrail put-event-selectors` 명령을 실행하여 추적에 대한 데이터 API 작업을 로깅합니다. DB 클러스터에 모든 Data API 이벤트를 로깅하려면 리소스 유형으로 `AWS::RDS::DBCluster`를 지정하세요. 다음 예시는 모든 데이터 API 이벤트를 DB 클러스터에 로깅합니다. 자세한 내용을 알아보려면 AWS CloudTrail 사용 설명서의 AWS Command Line Interface을 사용한 데이터 이벤트 로깅을 참조하세요.

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
```

```
            "Equals": [
                "AWS::RDS::DBCluster"
            ]
        }
    ]
}'
```

readOnly, eventName, 및 resources.ARN 필드를 추가로 필터링하도록 고급 이벤트 선택기를 구성할 수 있습니다. 이러한 필드에 대한 자세한 내용은 AdvancedFieldSelector를 참조하세요.

AWS CloudTrail 추적에서 데이터 API 이벤트 제외(Aurora Serverless v1 전용)

Aurora Serverless v1의 경우, 데이터 API 이벤트는 관리 이벤트입니다. 기본적으로 모든 데이터 API 이벤트는 AWS CloudTrail 추적에 포함됩니다. 그러나 데이터 API는 수많은 이벤트를 생성할 수 있으므로, CloudTrail 추적에서 이러한 이벤트를 제외할 수 있습니다. Amazon RDS 데이터 API 이벤트 제외 설정은 모든 데이터 API 이벤트를 추적에서 제외합니다. 특정 데이터 API 이벤트는 제외할 수 없습니다.

추적에서 데이터 API 이벤트를 제외하려면 다음을 수행합니다.

- CloudTrail 콘솔에서 추적을 생성하거나 추적을 업데이트할 때 [Amazon RDS 데이터 API 이벤트 제외(Exclude Amazon RDS Data API events)] 설정을 선택합니다.
- CloudTrail API에서 PutEventSelectors 작업을 사용합니다. 고급 이벤트 선택기를 사용하는 경우 eventSource 필드를 rdsdata.amazonaws.com과 같지 않게 설정하여 데이터 API 이벤트를 제외할 수 있습니다. 기본 이벤트 선택기를 사용하는 경우 ExcludeManagementEventSources 속성 값을 rdsdata.amazonaws.com으로 설정하여 데이터 API 이벤트를 제외할 수 있습니다. 자세한 내용을 알아보려면 AWS CloudTrail 사용 설명서의 AWS Command Line Interface을 사용한 이벤트 로깅을 참조하세요.

> **Warning**
>
> CloudTrail 로그에서 데이터 API 이벤트를 제외하면 데이터 API 작업이 모호해질 수 있습니다. 보안 주체에게 이 작업을 수행하는 데 필요한 cloudtrail:PutEventSelectors 권한을 부여할 때는 주의해야 합니다.

콘솔 설정 또는 추적용 이벤트 선택기를 변경하여 언제든지 이 제외를 비활성화할 수 있습니다. 그러면 추적이 데이터 API 이벤트 기록을 시작합니다. 그러나 제외가 유효한 동안 발생한 데이터 API 이벤트는 복구할 수 없습니다.

콘솔 또는 API를 사용하여 데이터 API 이벤트를 제외하면 결과로 나타나는 CloudTrail `PutEventSelectors` API 작업도 CloudTrail Logs에 기록됩니다. 데이터 API 이벤트가 CloudTrail Logs에 나타나지 않으면 `ExcludeManagementEventSources` 속성이 `rdsdata.amazonaws.com`으로 설정된 `PutEventSelectors` 이벤트를 찾습니다.

자세한 내용은 AWS CloudTrail 사용 설명서의 추적 관리 이벤트 로깅을 참조하세요.

데이터 API 로그 파일 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

Aurora PostgreSQL Serverless v2 및 프로비저닝

다음은 Aurora PostgreSQL Serverless v2와 프로비저닝된 데이터베이스의 `ExecuteStatement` 작업을 보여주는 CloudTrail 로그 항목을 나타낸 예제입니다. 이러한 데이터베이스의 경우, 모든 데이터 API 이벤트는 이벤트 소스가 rdsdataapi.amazonaws.com이고 이벤트 유형이 Rds 데이터 서비스인 데이터 이벤트입니다.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
```

```
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```


Aurora Serverless v1

다음 예는 위의 예제 CloudTrail 로그 항목이 Aurora Serverless v1에 대해 어떻게 나타나는지 보여줍니다. Aurora Serverless v1의 경우, 모든 이벤트는 이벤트 소스가 rdsdata.amazonaws.com이고 이벤트 유형이 AwsApiCall인 관리 이벤트입니다.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
```

```
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

성능 개선 도우미를 사용한 RDS 데이터 API 쿼리 모니터링

Aurora 클러스터가 Aurora Serverless v2 또는 프로비저닝된 인스턴스를 실행 중인 경우 RDS 데이터 API와 함께 성능 개선 도우미를 사용할 수 있습니다.

Aurora와 함께 성능 개선 도우미를 사용하는 방법에 대한 자세한 내용은 성능 개선 도우미를 통한 Amazon Aurora 모니터링 섹션을 참조하세요.

성능 개선 도우미에서 RDS 데이터 API 쿼리를 나타내는 방법

데이터 API를 사용하면 Aurora 클러스터는 애플리케이션에서 제출한 데이터 API 호출을 기반으로 쿼리를 처리합니다. 또한 데이터 API는 제한 시간 임계값을 초과하는 쿼리 취소와 같은 내부 작업의 일부로 일부 SQL 문을 수행합니다. 두 종류의 SQL 작업은모두 성능 개선 도우미 통계 및 차트에 표시됩니다.

- Aurora 클러스터에 제출하는 데이터 API 쿼리의 경우 PI 대시보드의 호스트 필드가 RDS 데이터 API로 표시됩니다. Aurora PostgreSQL의 경우 application_name 필드에 `rds-data-api` 값이 있습니다. 상위 호스트 또는 상위 애플리케이션을 차원으로 사용하여 데이터베이스 로드를 분석할 때 이러한 레이블을 찾습니다.
- 연결 풀 및 쿼리 제한 시간과 같은 데이터베이스 측면을 관리하기 위해 데이터 API가 실행하는 모든 내부 쿼리는 접두사 RDS 데이터 API로 주석이 지정됩니다. 예: `/* RDS Data API */ select * from my_table;`은 상위 SQL을 차원으로 데이터베이스 로드를 분석할 때 이러한 접두사를 찾습니다. 문에는 `/* RDS Data API */`의 SQL 주석이 달립니다.

Aurora 쿼리 편집기 사용하기

Aurora 쿼리 편집기를 사용하면 AWS Management Console을 통해 Aurora DB 클러스터에서 SQL 문을 실행할 수 있습니다. SQL 쿼리, 데이터 조작(DML) 문, 데이터 정의(DDL) 문을 실행할 수 있습니다. 콘솔 인터페이스를 사용하면 데이터베이스 유지 관리, 보고서 생성, SQL 실험을 수행할 수 있습니다. EC2 인스턴스나 노트북 컴퓨터와 같은 별도의 클라이언트 시스템에서 DB 클러스터에 연결하기 위해 네트워크 구성을 설정하지 않아도 됩니다.

쿼리 편집기에 RDS 데이터 API(데이터 API)가 활성화된 Aurora DB 클러스터가 필요합니다. 데이터 API를 지원하는 DB 클러스터와 이를 활성화하는 방법에 대한 자세한 내용은 Amazon RDS Data API 사용 섹션을 참조하세요. 실행할 수 있는 SQL에는 데이터 API 제한이 적용됩니다. 자세한 내용은 the section called “제한 사항” 섹션을 참조하세요.

쿼리 편집기의 가용성

쿼리 편집기는 데이터 API를 사용할 수 있는 AWS 리전에서 데이터 API를 지원하는 Aurora MySQL 및 Aurora PostgreSQL 엔진 버전을 사용하는 Aurora DB 클러스터에 사용할 수 있습니다. 자세한 내용은 RDS Data API를 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

쿼리 편집기에 대한 액세스 권한 부여

쿼리 편집기에서 쿼리를 실행하려면 사용자에게 권한을 부여해야 합니다. 미리 정의된 AWS Identity and Access Management(IAM) 정책인 `AmazonRDSDataFullAccess` 정책을 사용자에게 추가하여 쿼리 편집기에서 쿼리를 실행할 수 있는 권한을 사용자에게 부여할 수 있습니다.

> **Note**
>
> IAM 사용자를 생성할 때 관리 사용자 이름 및 암호와 같이 데이터베이스 사용자에 대해 설정한 것과 동일한 사용자 이름과 암호를 사용해야 합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 AWS 계정에서 IAM 사용자 생성을 참조하세요.

또한 쿼리 편집기에 대한 액세스를 허가하는 IAM 정책을 생성할 수도 있습니다. 정책을 생성한 후에는 해당 정책을 쿼리 편집기에 액세스해야 하는 각 사용자에게 추가합니다.

다음 정책은 사용자가 쿼리 편집기에 액세스하는 데 필요한 최소 권한을 제공합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

IAM 정책 생성에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 IAM 정책 생성을 참조하세요.

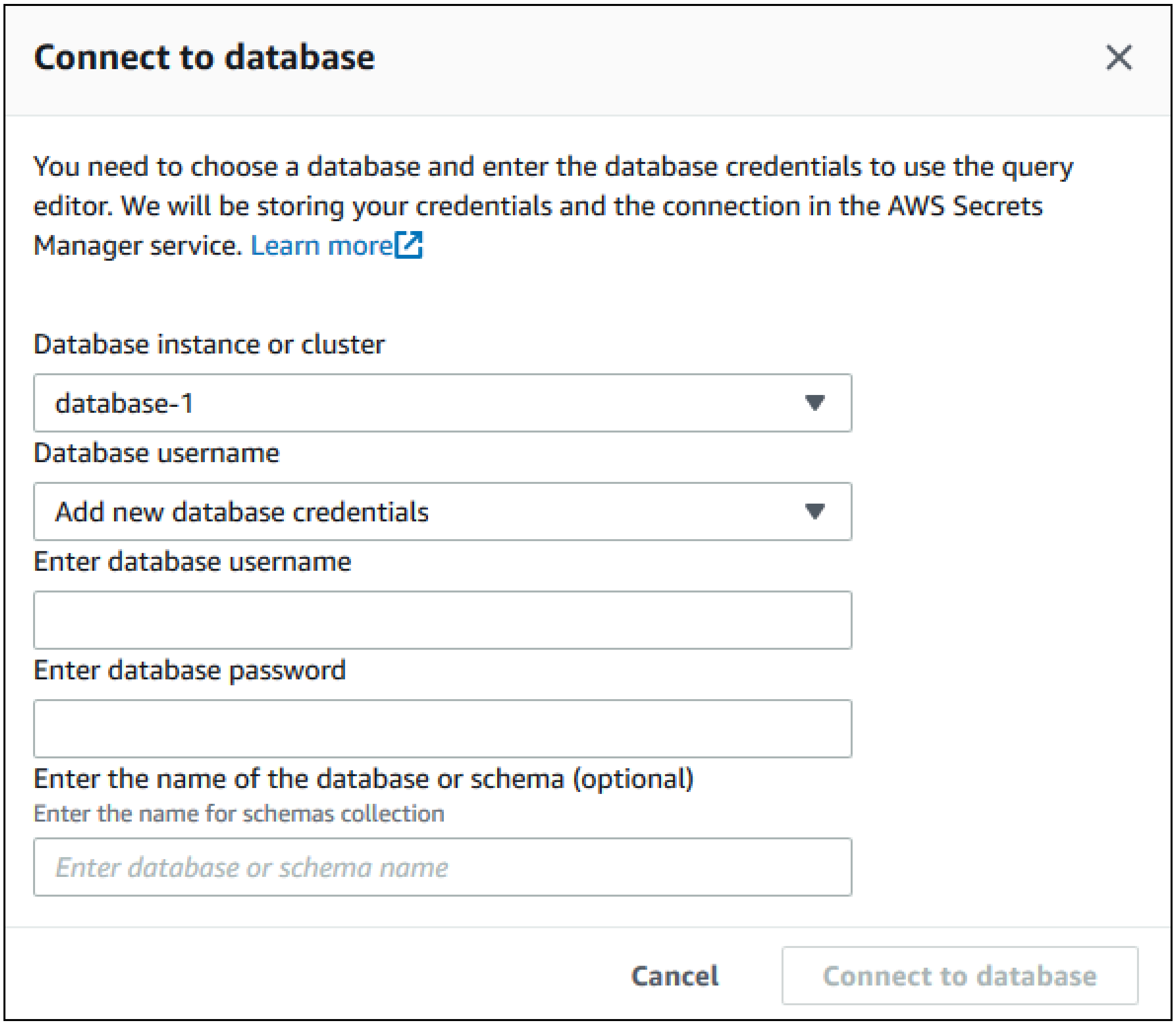
사용자에게 IAM 정책 추가에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.

쿼리 편집기에서 쿼리 실행

쿼리 편집기에서 Aurora DB 클러스터에 대해 SQL 문을 실행할 수 있습니다. 실행할 수 있는 SQL에는 데이터 API 제한이 적용됩니다. 자세한 내용은 the section called “제한 사항” 섹션을 참조하세요.

쿼리 편집기에서 쿼리를 실행하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 쿼리할 Aurora DB 클러스터를 생성한 AWS 리전을 선택합니다.
3. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
4. SQL 쿼리를 실행할 Aurora DB 클러스터를 선택합니다.
5. 작업에서 쿼리를 선택합니다. 이전에 데이터베이스에 연결하지 않은 경우 데이터베이스에 연결 페이지가 열립니다.



6. 다음 정보를 입력합니다.

   a. 데이터베이스 인스턴스 또는 클러스터에서 SQL 쿼리를 실행할 Aurora DB 클러스터를 선택합니다.

   b. Database username(데이터베이스 사용자 이름)에서 연결할 데이터베이스 사용자의 사용자 이름을 선택하거나 Add new database credentials(새 데이터베이스 자격 증명 추가)를 선택합니다. Add new database credentials(새 데이터베이스 자격 증명 추가)를 선택한 경우 Enter database username(데이터베이스 사용자 이름 입력)에 새 데이터베이스 자격 증명의 사용자 이름을 입력합니다.

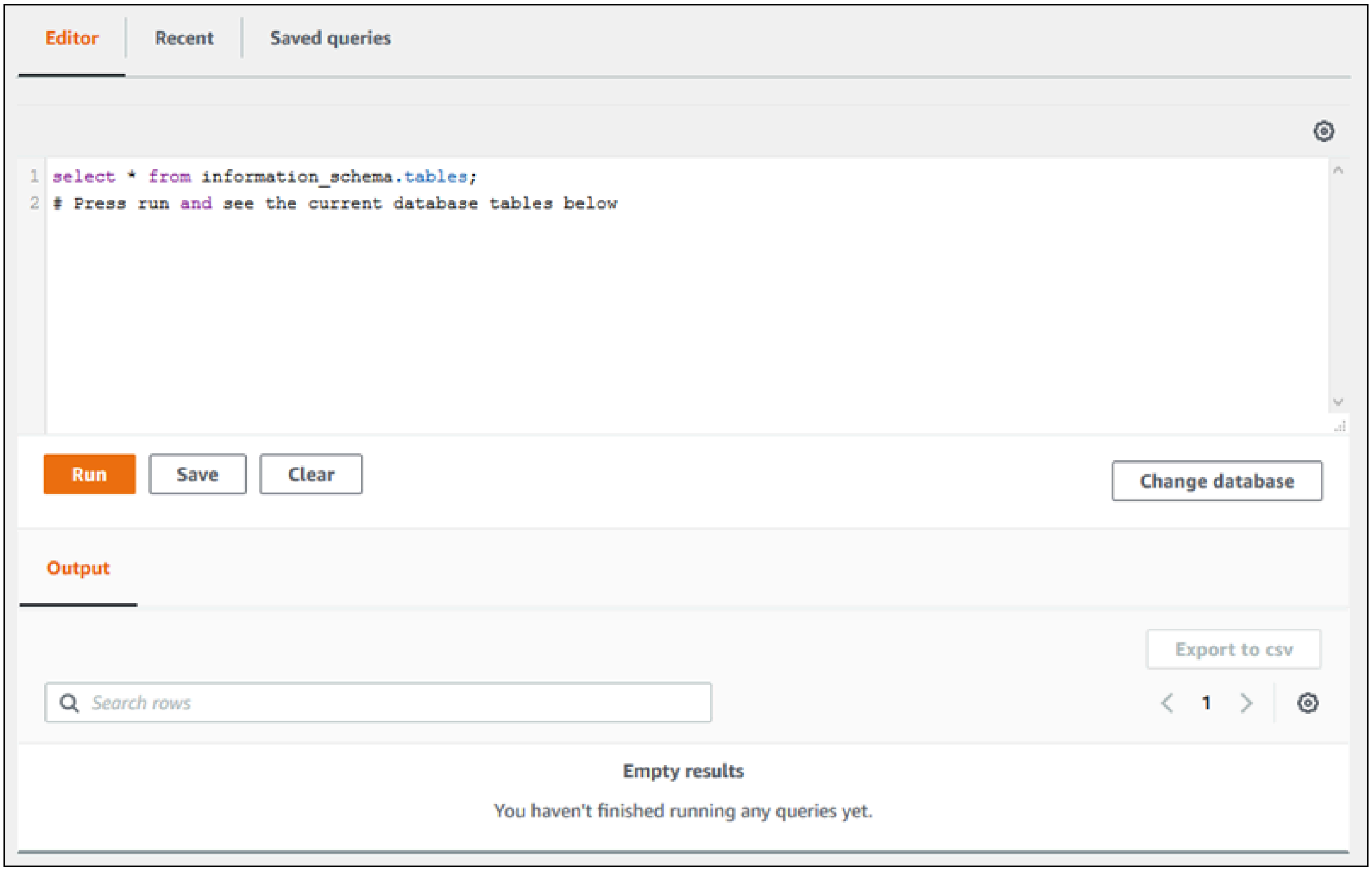
   c. Enter database username(데이터베이스 사용자 이름 입력)에 선택한 데이터베이스 사용자의 암호를 입력합니다.

   d. 마지막 상자에는 Aurora DB 클러스터에 사용할 데이터베이스 또는 스키마의 이름을 입력합니다.

   e. [데이터베이스에 연결(Connect to database)]을 선택합니다.

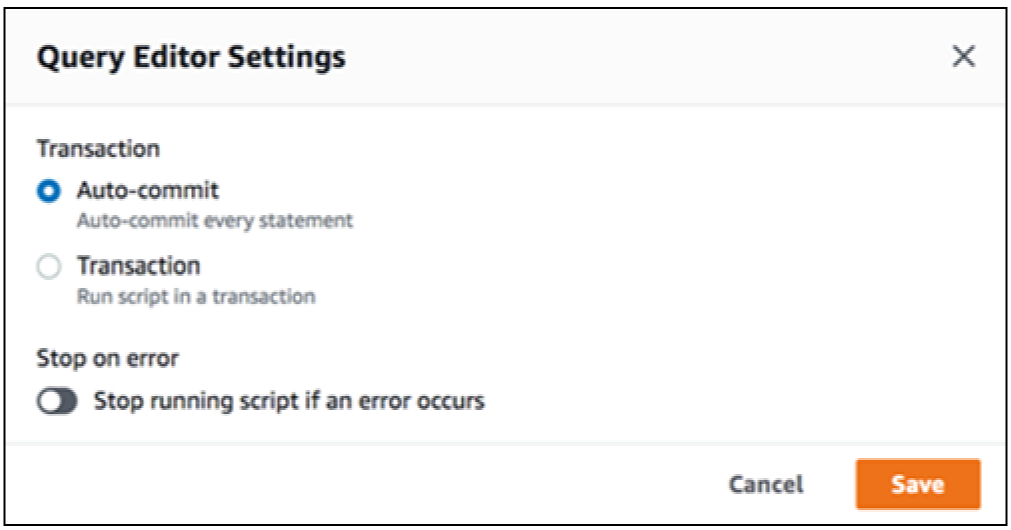
> **Note**
>
> 연결되면 연결 및 인증 정보가 AWS Secrets Manager에 저장됩니다. 따라서 연결 정보를 다시 입력할 필요가 없습니다.

7. 쿼리 편집기에서 데이터베이스에 대해 실행할 SQL 쿼리를 입력합니다.



각 SQL 문은 자동으로 커밋되거나, 트랜잭션의 일부로 스크립트에서 SQL 문을 실행할 수 있습니다. 이 동작을 제어하려면 쿼리 창 위의 기어 모양 아이콘을 선택하십시오.

Query Editor Settings(쿼리 편집기 설정) 창이 나타납니다.



Auto-commit(자동 커밋)을 선택하면 모든 SQL 문이 자동으로 커밋됩니다. 트랜잭션을 선택하면 스크립트에서 문 그룹을 실행할 수 있습니다. 문은 그 전에 명시적으로 커밋되거나 롤백되지 않는 한 스크립트가 끝날 때 자동으로 커밋됩니다. 또한 Stop on error(오류 시 중지)를 활성화하여 오류가 발생하면 실행 중인 스크립트를 중지하도록 선택할 수 있습니다.

> **Note**
>
> 한 그룹의 문에서 데이터 정의 언어(DDL) 문으로 인해 이전 데이터 조작 언어(DML) 문이 커밋될 수 있습니다. 또한 스크립트의 문 그룹에 `COMMIT` 및 `ROLLBACK` 문을 포함할 수 있습니다.

Query Editor Settings(쿼리 편집기 설정) 창에서 선택한 후 저장을 선택하십시오.

8. 실행을 선택하거나 Ctrl+Enter를 누르면 쿼리 편집기에 쿼리 결과가 표시됩니다.

   쿼리를 실행한 후 저장을 선택하여 쿼리를 저장된 쿼리에 저장합니다.

   Export to csv(csv로 내보내기)를 선택하여 쿼리 결과를 스프레드시트 형식으로 내보냅니다.

이전 쿼리를 찾고, 편집하고, 재실행할 수 있습니다. 이렇게 하려면 Recent(최근) 탭이나 저장된 쿼리 탭을 선택하고 해당 쿼리 텍스트를 선택한 후, 실행을 선택합니다.

데이터베이스를 변경하려면 데이터베이스 변경을 선택하세요.

Database Query Metadata Service(DBQMS) API 참조

Database Query Metadata Service(dbqms)는 내부 전용 서비스입니다. Amazon RDS를 포함한 여러 AWS 서비스에 사용되는 AWS Management Console의 쿼리 편집기에 대해 최근 쿼리와 저장된 쿼리를 제공합니다 .

지원되는 DBQMS 작업

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries
- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs

- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

새 즐겨찾기 쿼리를 저장합니다. 각 사용자는 저장된 쿼리를 최대 1,000개까지 생성할 수 있습니다. 이 제한은 향후 변경될 수 있습니다.

CreateQueryHistory

새 쿼리 기록 항목을 저장합니다.

CreateTab

새 쿼리 탭을 저장합니다. 각 사용자는 쿼리 탭을 최대 10개까지 생성할 수 있습니다.

DeleteFavoriteQueries

저장된 쿼리를 하나 이상 삭제합니다.

DeleteQueryHistory

쿼리 기록 항목을 삭제합니다.

DeleteTab

쿼리 탭 항목을 삭제합니다.

DescribeFavoriteQueries

지정된 계정에서 사용자가 생성한 저장된 쿼리를 나열합니다.

DescribeQueryHistory

쿼리 기록 항목을 나열합니다.

DescribeTabs

지정된 계정에서 사용자가 생성한 쿼리 탭을 나열합니다.

GetQueryString

쿼리 ID에서 전체 쿼리 텍스트를 검색합니다.

UpdateFavoriteQuery

쿼리 문자열, 설명, 이름 또는 만료 날짜를 업데이트합니다.

UpdateQueryHistory

쿼리 기록의 상태를 업데이트합니다.

UpdateTab

쿼리 탭 이름과 쿼리 문자열을 업데이트합니다.

Amazon Aurora 기계 학습 사용

Amazon Aurora 기계 학습을 사용하면 필요에 따라 Aurora DB 클러스터를 다음 AWS 기계 학습 서비스 중 하나와 통합할 수 있습니다. 각 서비스마다 특정 기계 학습 사용 사례를 지원합니다.

Amazon Bedrock

Amazon Bedrock은 생성형 AI 애플리케이션을 구축하고 확장하는 데 도움이 되는 개발자 도구와 함께 API를 통해 AI 회사의 주요 기반 모델을 사용할 수 있게 해주는 완전 관리형 서비스입니다. Amazon Bedrock을 사용할 경우, 모든 타사 파운데이션 모델에서 비용을 지불하고 추론을 실행할 수 있습니다. 요금은 입력 토큰과 출력 토큰의 양, 그리고 모델에 대해 프로비저닝된 처리량을 구매했는지 여부에 따라 결정됩니다. 자세한 내용은 Amazon Bedrock 사용 설명서의 Amazon Bedrock 이란 무엇인가요?를 참조하세요.

Amazon Comprehend

Amazon Comprehend는 문서에서 인사이트를 추출하는 데 사용되는 관리형 자연어 처리(NLP) 서비스입니다. Amazon Comprehend를 사용하면 주체, 핵심 문구, 언어 및 기타 특징을 분석하여 문서 내용을 기반으로 감정을 추론할 수 있습니다. 자세한 내용은 Amazon Comprehend 개발자 가이드의 Amazon Comprehend란?을 참조하세요.

SageMaker AI

Amazon SageMaker AI는 완전관리형 기계 학습 서비스입니다. 데이터 과학자와 개발자는 Amazon SageMaker AI를 사용하여 사기 탐지 및 제품 추천과 같은 다양한 추론 작업을 위한 기계 학습 모델을 구축하고 훈련하고 테스트합니다. 기계 학습 모델을 프로덕션에서 사용할 준비가 되면 Amazon SageMaker AI 호스팅 환경에 배포할 수 있습니다. 자세한 내용은 Amazon SageMaker AI Developer Guide의 What Is Amazon SageMaker AI?를 참조하세요.

Amazon Comprehend를 Aurora DB 클러스터와 함께 사용하면 SageMaker AI를 사용할 때보다 예비 설정이 적습니다. AWS 기계 학습이 처음이라면 먼저 Amazon Comprehend를 살펴보는 것이 좋습니다.

주제

Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용

Amazon Aurora 기계 학습을 Aurora MySQL DB 클러스터와 함께 사용하면 필요에 따라 Amazon Bedrock, Amazon Comprehend 또는 Amazon SageMaker AI를 사용할 수 있습니다. 서비스마다 다른 기계 학습 사용 사례를 지원합니다.

목차

Aurora MySQL과 함께 Aurora 기계 학습을 사용할 때 요구 사항

AWS 기계 학습 서비스는 자체 프로덕션 환경에서 설정되고 실행되는 관리형 서비스입니다. Aurora 기계 학습은 Amazon Bedrock, Amazon Comprehend, SageMaker AI와의 통합을 지원합니다. Aurora 기계 학습을 사용하도록 Aurora MySQL DB 클러스터를 설정하기 전에 다음 요구 사항 및 사전 조건을 이해해야 합니다.

- 기계 학습 서비스가 Aurora MySQL DB 클러스터와 동일한 AWS 리전에서 실행되어야 합니다. 다른 리전의 Aurora MySQL DB 클러스터에서는 기계 학습 서비스를 사용할 수 없습니다.
- Aurora MySQL DB 클러스터가 Amazon Bedrock, Amazon Comprehend 및 SageMaker AI 서비스와 다른 Virtual Public Cloud(VPC)에 있는 경우 VPC의 보안 그룹은 대상 Aurora 기계 학습 서비스에 대한 아웃바운드 연결을 허용해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 AWS 리소스에 대한 트래픽 제어를 참조하세요.
- 해당 클러스터와 Aurora Machine Learning을 함께 사용하려는 경우 이전 버전의 Aurora MySQL을 실행하는 Aurora 클러스터를 지원되는 상위 버전으로 업그레이드할 수 있습니다. 자세한 내용은 Amazon Aurora MySQL에 대한 데이터베이스 엔진 업데이트 단원을 참조하십시오.
- Aurora MySQL DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용해야 합니다. 사용하려는 각 Aurora 기계 학습 서비스의 설정 프로세스가 끝날 때 해당 서비스에 대해 생성된 관련 IAM 역할의 Amazon 리소스 이름(ARN)을 추가합니다. Aurora MySQL용 사용자 지정 DB 클러스터 파라미터 그룹을 미리 생성하고 이를 사용하도록 Aurora MySQL DB 클러스터를 구성하여 설정 프로세스 종료 시 바로 수정할 수 있도록 하는 것이 좋습니다.
- SageMaker AI의 경우:
  - 추론에 사용할 기계 학습 구성 요소를 설정하고 사용할 준비가 되어 있어야 합니다. Aurora MySQL DB 클러스터에 대한 구성 프로세스에서 SageMaker AI 엔드포인트의 ARN을 사용할 수 있어야 합니다. 팀의 데이터 과학자는 SageMaker AI와 협력하여 모델을 준비하고 그 외의 작업을 처리하는 데 가장 잘 대처할 수 있을 것입니다. Amazon SageMaker AI를 시작하려면 Get Started with Amazon SageMaker AI를 참조하세요. 추론 및 엔드포인트에 대한 자세한 내용은 Real-time inference(실시간 추론)를 참조하세요.
  - SageMaker AI를 자체 훈련 데이터와 함께 사용하려면 Aurora 기계 학습에 대한 Aurora MySQL 구성의 일부로 Amazon S3 버킷을 설정해야 합니다. 이렇게 하려면 SageMaker AI 통합을 설정하는 것과 동일한 일반 프로세스를 따라야 합니다. 이 선택적 설정 프로세스를 요약한 내용은 Amazon S3 for SageMaker AI를 사용하도록 Aurora MySQL DB 클러스터 설정(선택 사항) 섹션을 참조하세요.
- Aurora Global Database의 경우 Aurora Global Database를 구성하는 모든 AWS 리전에서 사용할 Aurora 기계 학습 서비스를 설정합니다. 예를 들어 Aurora 글로벌 데이터베이스에 SageMaker AI와 함께 Aurora 기계 학습을 사용하려면 모든 AWS 리전에 대해 다음 작업을 수행하세요.

- 동일한 SageMaker AI 훈련 모델 및 엔드포인트로 Amazon SageMaker AI 서비스를 설정합니다. 이름도 동일해야 합니다.
- Aurora 기계 학습을 사용하도록 Aurora MySQL DB 클러스터 설정에 설명된 대로 IAM 역할을 생성합니다.
- 모든 AWS 리전에서 각 Aurora MySQL DB 클러스터의 사용자 지정 DB 클러스터 파라미터 그룹에 IAM 역할의 ARN을 추가합니다.

이러한 작업을 수행하려면 Aurora Global Database를 구성하는 모든 AWS 리전의 Aurora MySQL 버전에서 Aurora 기계 학습을 사용할 수 있어야 합니다.

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다.

- Aurora MySQL과 함께 사용하는 Amazon Comprehend 및 Amazon SageMaker AI의 버전 및 리전 가용성에 대한 자세한 정보는 Aurora MySQL을 사용하는 Aurora 기계 학습 섹션을 참조하세요.
- Amazon Bedrock은 Aurora MySQL 버전 3.06 이상에서만 지원됩니다.

  Amazon Bedrock의 리전 가용성에 대한 자세한 내용은 Amazon Bedrock 사용 설명서에서 AWS 리전에서 지원하는 모델을 참조하세요.

Aurora MySQL을 사용하는 Aurora 기계 학습의 지원 기능 및 제한 사항

Aurora 기계 학습과 함께 Aurora MySQL 사용하는 경우 다음과 같은 제한 사항이 적용됩니다.

- Aurora 기계 학습 확장은 벡터 인터페이스를 지원하지 않습니다.
- Aurora 기계 학습 통합은 트리거에 사용될 경우 지원되지 않습니다.
- Aurora 기계 학습 함수는 바이너리 로깅(binlog) 복제와 호환되지 않습니다.
  - `--binlog-format=STATEMENT` 설정은 Aurora Machine Learning 함수 호출에 대해 예외를 발생시킵니다.
  - Aurora 기계 학습 함수는 비결정적이며 비결정적 저장 함수는 binlog 형식과 호환되지 않습니다.

  자세한 내용은 MySQL 설명서의 Binary Logging Formats를 참조하세요.
- generated-always 열이 있는 테이블을 호출하는 저장 함수는 지원되지 않습니다. 이는 모든 Aurora MySQL 저장 함수에 적용됩니다. 이 열 유형에 대해 자세히 알아보려면 MySQL 설명서에서 CREATE TABLE and Generated Columns(CREATE TABLE 및 생성된 열)를 참조하세요.

- Amazon Bedrock 함수는 `RETURNS JSON`을 지원하지 않습니다. 필요한 경우 `CONVERT` 또는 `CAST`를 사용하여 `TEXT`에서 `JSON`으로 변환할 수 있습니다.
- Amazon Bedrock은 배치 요청을 지원하지 않습니다.
- Aurora MySQL은 `text/csv`의 `ContentType`을 통해 쉼표로 구분된 값(CSV) 형식을 읽고 쓸 수 있는 모든 SageMaker AI 엔드포인트를 지원합니다. 이 형식은 다음과 같은 내장 SageMaker AI 알고리즘에서 사용할 수 있습니다.
  - Linear Learner
  - 랜덤 컷 포레스트
  - XGBoost

  이러한 알고리즘에 대해 자세히 알아보려면 Amazon SageMaker AI Developer Guide의 Choose an Algorithm을 참조하세요.

Aurora 기계 학습을 사용하도록 Aurora MySQL DB 클러스터 설정

다음 주제에서는 이러한 Aurora 기계 학습 서비스 각각에 대한 별도의 설정 절차를 찾을 수 있습니다.

주제

Amazon Bedrock을 사용하도록 Aurora MySQL DB 클러스터 설정

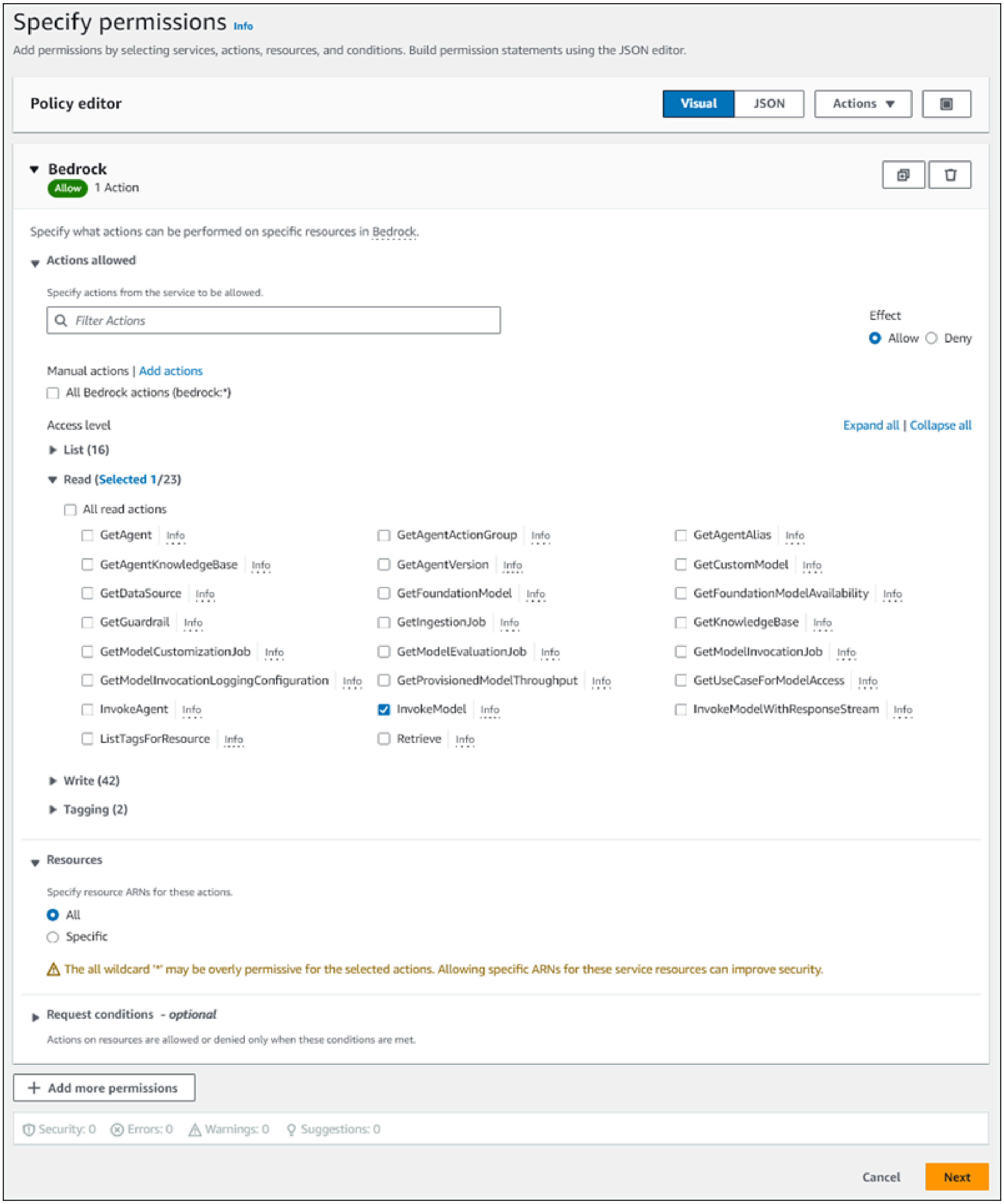
Aurora 기계 학습은 AWS Identity and Access Management(IAM) 역할과 정책을 기반으로 Aurora MySQL DB 클러스터가 Amazon Bedrock 서비스에 액세스하고 사용할 수 있도록 합니다. 다음 절차는 DB 클러스터를 Amazon Bedrock과 통합할 수 있도록 IAM 권한 정책 및 역할을 생성합니다.

IAM 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택합니다.
4. 권한 지정 페이지의 서비스 선택에서 Bedrock을 선택합니다.

   Amazon Bedrock 권한이 표시됩니다.
5. 읽기를 확장한 다음 InvokeModel을 선택합니다.
6. 리소스에서 모두를 선택합니다.

   권한 지정 페이지는 다음 그림과 비슷합니다.



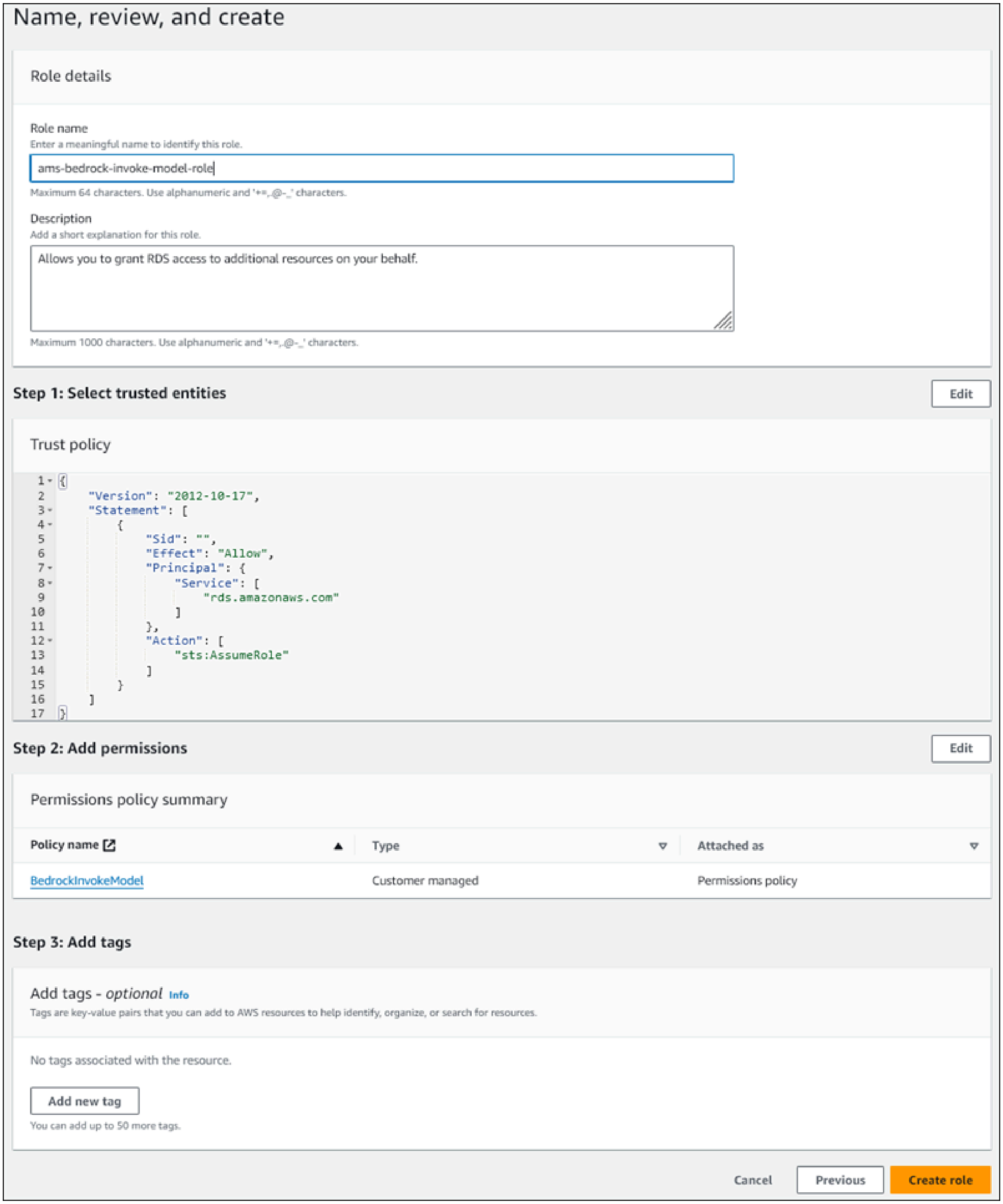
7. Next(다음)를 선택합니다.
8. 검토 및 생성 페이지에 **`BedrockInvokeModel`**과 같이 정책 이름을 입력합니다.
9. 정책을 검토한 후 정책 생성을 선택합니다.

다음으로 Amazon Bedrock 권한 정책을 사용하는 IAM 역할을 생성합니다.

IAM 역할을 만들려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지의 사용 사례에서 RDS를 선택합니다.
5. RDS - 데이터베이스에 역할 추가를 선택한 후 다음을 선택합니다.
6. 권한 추가 페이지의 권한 정책에서 생성한 IAM 정책을 선택한 후 다음을 선택합니다.
7. 이름, 검토 및 생성 페이지에 **ams-bedrock-invoke-model-role**과 같이 역할 이름을 입력합니다.

   역할은 다음 그림과 비슷합니다.



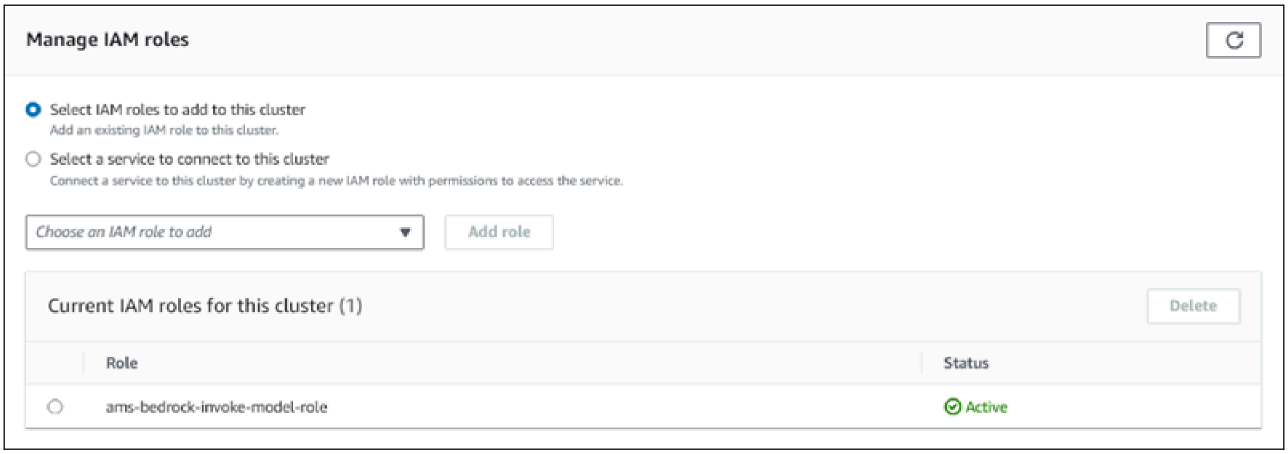
8. 역할을 검토한 후 역할 생성을 선택합니다.

다음으로 Amazon Bedrock IAM 역할을 DB 클러스터와 연결합니다.

IAM 역할을 DB 클러스터에 연결하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Amazon Bedrock 서비스에 연결할 Aurora MySQL DB 클러스터를 선택합니다.
4. 연결 및 보안(Connectivity & security) 탭을 선택합니다.
5. IAM 역할 관리 섹션에서 이 클러스터에 추가할 IAM 선택을 선택합니다.
6. 생성한 IAM을 선택한 다음 역할 추가를 선택합니다.

   IAM 역할이 DB 클러스터와 연결되며, 처음에는 보류 중 상태였다가 활성 상태가 됩니다. 프로세스가 완료되면 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.



Aurora MySQL DB 클러스터와 연결된 사용자 지정 DB 클러스터 파라미터 그룹의 `aws_default_bedrock_role` 파라미터에 이 IAM 역할의 ARN을 추가해야 합니다. Aurora MySQL DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용하지 않는 경우 Aurora MySQL DB 클러스터와 함께 사용할 그룹을 만들어 통합을 완료해야 합니다. 자세한 내용은 Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹 단원을 참조하십시오.

DB 클러스터 파라미터를 구성하는 방법

1. Amazon RDS 콘솔에서 Aurora MySQL DB 클러스터의 구성 탭을 엽니다.
2. 클러스터에 대해 구성한 DB 클러스터 파라미터 그룹을 찾습니다. 링크를 선택하여 사용자 지정 DB 클러스터 파라미터 그룹을 연 다음 편집을 선택합니다.
3. 사용자 지정 DB 클러스터 파라미터 그룹에서 `aws_default_bedrock_role` 파라미터를 찾습니다.

4. 값 필드에 IAM 역할의 ARN을 입력합니다.
5. 변경 사항 저장을 선택하여 설정을 저장합니다.
6. 이 파라미터 설정이 적용되도록 Aurora MySQL DB 클러스터의 기본 인스턴스를 재부팅합니다.

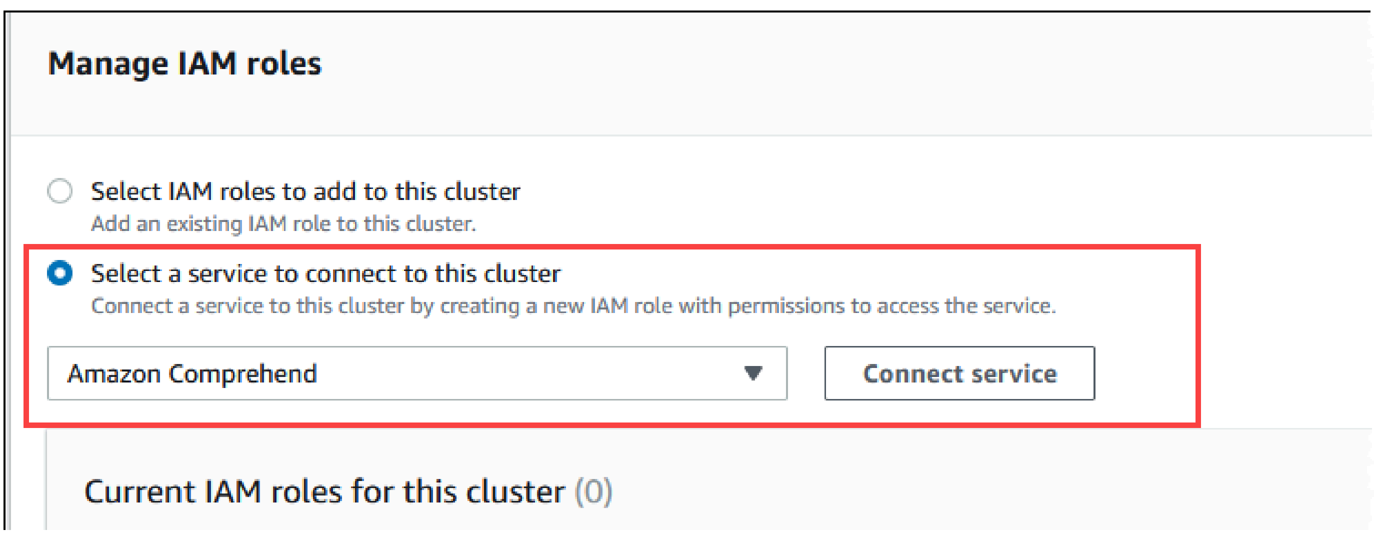
Amazon Bedrock에 대한 IAM 통합이 완료되었습니다. 데이터베이스 사용자에게 Aurora 기계 학습에 대한 액세스 권한 부여를 통해 Aurora MySQL DB 클러스터를 Amazon Bedrock와 함께 사용할 수 있도록 계속해서 설정하세요.

Amazon Comprehend를 사용하도록 Aurora MySQL DB 클러스터 설정

Aurora 기계 학습은 AWS Identity and Access Management 역할과 정책을 기반으로 Aurora MySQL DB 클러스터가 Amazon Comprehend 서비스에 액세스하고 사용할 수 있도록 합니다. 다음 절차는 Amazon Comprehend를 사용할 수 있도록 클러스터의 IAM 역할 및 정책을 자동으로 생성합니다.

Amazon Comprehend를 사용하도록 Aurora MySQL DB 클러스터를 설정하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. Amazon Comprehend 서비스에 연결할 Aurora MySQL DB 클러스터를 선택합니다.
4. 연결 및 보안(Connectivity & security) 탭을 선택합니다.
5. IAM 역할 관리 섹션에서 이 클러스터에 연결할 서비스 선택을 선택합니다.
6. 메뉴에서 Amazon Comprehend를 선택한 다음 서비스 연결을 선택합니다.

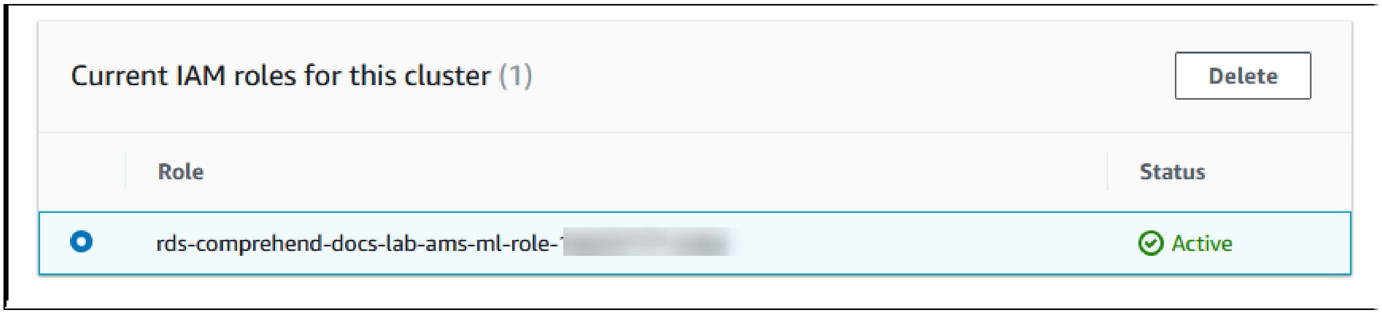


7. 클러스터를 Amazon Comprehend에 연결 대화 상자에는 추가 정보가 필요하지 않습니다. 하지만 Aurora와 Amazon Comprehend 간의 통합이 현재 미리 보기 중임을 알리는 메시지가 표시될 수

있습니다. 계속하기 전에 메시지를 반드시 읽어보세요. 계속 진행하지 않으려면 취소를 선택하면 됩니다.

8. 서비스 연결을 선택하여 통합 프로세스를 완료합니다.

   Aurora가 IAM 역할을 생성합니다. 또한 Aurora MySQL DB 클러스터가 Amazon Comprehend 서비스를 사용하도록 허용하는 정책을 생성하고 해당 역할에 정책을 연결합니다. 프로세스가 완료되면 다음 이미지와 같이 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.



   Aurora MySQL DB 클러스터와 연결된 사용자 지정 DB 클러스터 파라미터 그룹의 `aws_default_comprehend_role` 파라미터에 이 IAM 역할의 ARN을 추가해야 합니다. Aurora MySQL DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용하지 않는 경우 Aurora MySQL DB 클러스터와 함께 사용할 그룹을 만들어 통합을 완료해야 합니다. 자세한 내용은 [Amazon Aurora DB 클러스터의 DB 클러스터 파라미터 그룹](#) 단원을 참조하십시오.

   사용자 지정 DB 클러스터 파라미터 그룹을 만들어 Aurora MySQL DB 클러스터와 연결한 후에 다음 단계를 계속 수행할 수 있습니다.

   클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용하는 경우 다음 작업을 수행하세요.

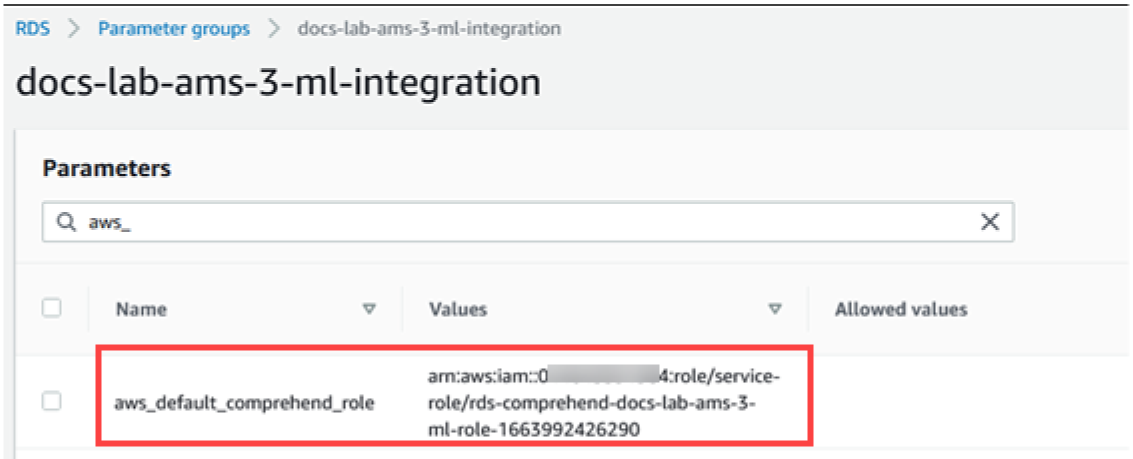
   a. Amazon RDS 콘솔에서 Aurora MySQL DB 클러스터의 구성 탭을 엽니다.

   b. 클러스터에 대해 구성한 DB 클러스터 파라미터 그룹을 찾습니다. 링크를 선택하여 사용자 지정 DB 클러스터 파라미터 그룹을 연 다음 편집을 선택합니다.

   c. 사용자 지정 DB 클러스터 파라미터 그룹에서 `aws_default_comprehend_role` 파라미터를 찾습니다.

   d. 값 필드에 IAM 역할의 ARN을 입력합니다.

   e. 변경 사항 저장을 선택하여 설정을 저장합니다. 다음 이미지에서 예시를 볼 수 있습니다.

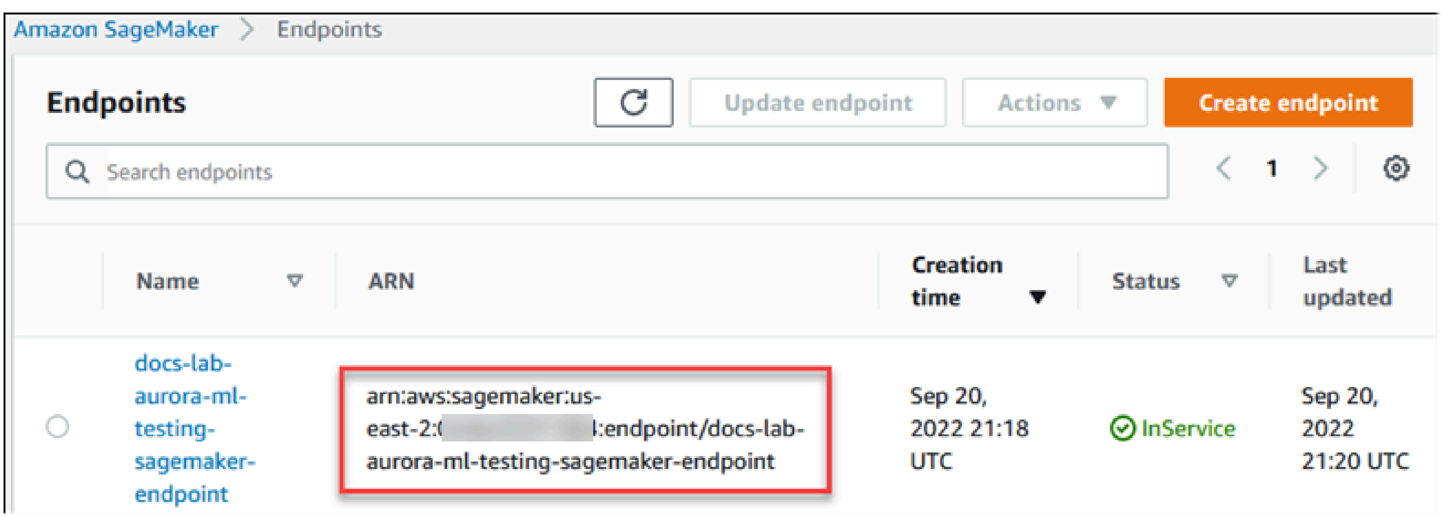


이 파라미터 설정이 적용되도록 Aurora MySQL DB 클러스터의 기본 인스턴스를 재부팅합니다.

Amazon Comprehend IAM 통합이 완료되었습니다. 적절한 데이터베이스 사용자에게 액세스 권한을 부여하여 Amazon Comprehend와 함께 작동하도록 Aurora MySQL DB 클러스터 설정을 계속하세요.

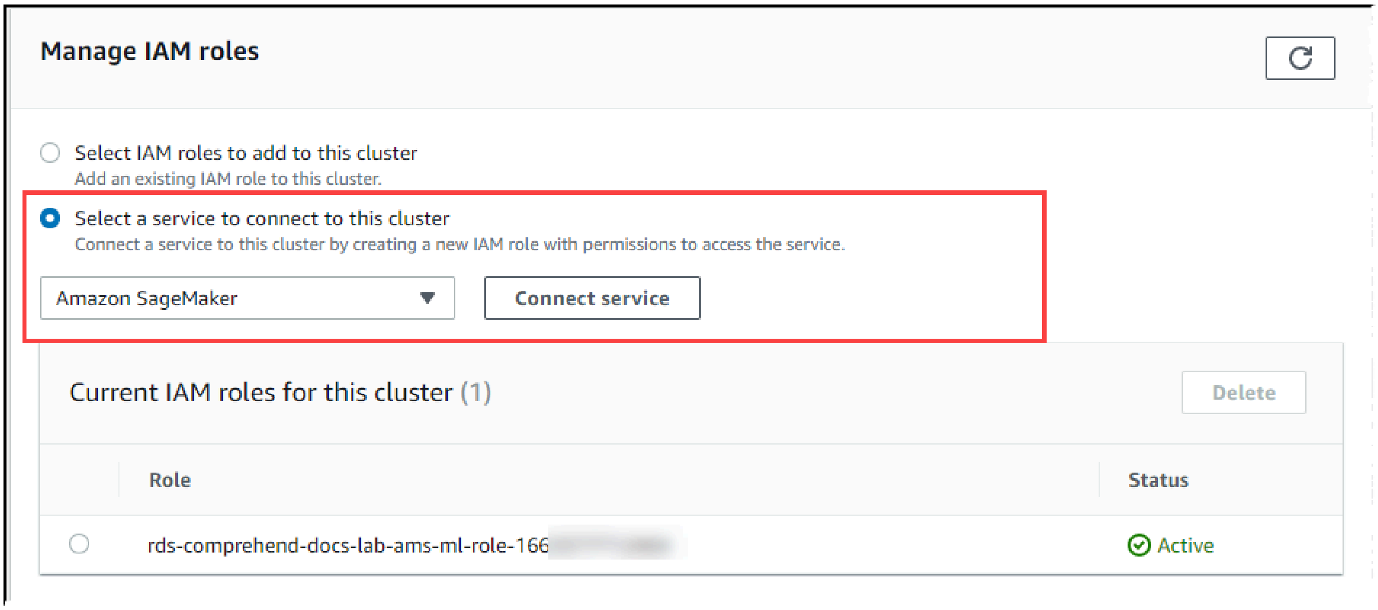
SageMaker AI를 사용하도록 Aurora MySQL DB 클러스터 설정

다음 프로시저는 SageMaker AI를 사용할 수 있도록 Aurora MySQL DB 클러스터의 IAM 역할 및 정책을 자동으로 만듭니다. 이 프로시저를 따르기 전에 필요할 때 입력할 수 있도록 SageMaker AI 엔드포인트를 준비하세요. 일반적으로 팀의 데이터 과학자가 Aurora MySQL DB 클러스터에서 사용할 수 있는 엔드포인트를 만드는 작업을 수행합니다. SageMaker AI 콘솔에서 이러한 엔드포인트를 찾을 수 있습니다. 탐색 창에서 추론 메뉴를 열고 엔드포인트를 선택합니다. 다음 이미지에서 예시를 볼 수 있습니다.

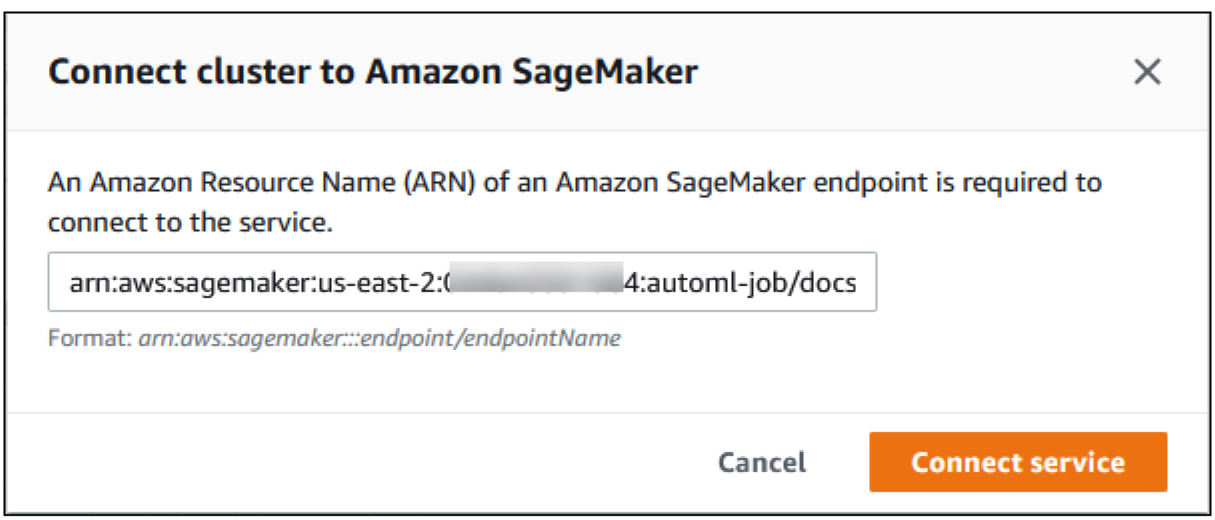


SageMaker AI를 사용하도록 Aurora MySQL DB 클러스터를 설정하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 탐색 메뉴에서 데이터베이스를 선택한 다음 SageMaker AI 서비스에 연결하려는 Aurora MySQL DB 클러스터를 선택합니다.
3. 연결 및 보안(Connectivity & security) 탭을 선택합니다.
4. IAM 역할 관리 섹션으로 스크롤한 후 이 클러스터에 연결할 서비스 선택을 선택합니다. 선택기에서 SageMaker AI를 선택합니다.



5. Connect service(서비스 연결)를 선택합니다.
6. SageMaker AI에 클러스터 연결 대화 상자에서 SageMaker AI 엔드포인트의 ARN을 입력합니다.



7. Aurora가 IAM 역할을 생성합니다. 또한 Aurora MySQL DB 클러스터가 SageMaker AI 서비스를 사용하도록 허용하는 정책을 만들고 해당 역할에 정책을 연결합니다. 프로세스가 완료되면 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.
8. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
9. AWS Identity and Access Management 탐색 메뉴의 액세스 관리 섹션에서 역할을 선택합니다.
10. 나열된 역할 중에서 역할을 찾습니다. 이름은 다음 패턴을 사용합니다.

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. 역할의 요약 페이지를 열고 ARN을 찾습니다. ARN을 기록하거나 복사 위젯을 사용하여 복사합니다.
12. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
13. Aurora MySQL DB 클러스터를 선택한 다음 구성 탭을 선택합니다.
14. DB 클러스터 파라미터 그룹을 찾아 링크를 선택하여 사용자 지정 DB 클러스터 파라미터 그룹을 엽니다. `aws_default_sagemaker_role` 파라미터를 찾아 값 필드에 IAM 역할의 ARN을 입력하고 설정을 저장합니다.
15. 이 파라미터 설정이 적용되도록 Aurora MySQL DB 클러스터의 기본 인스턴스를 재부팅합니다.

이제 IAM 설정이 완료되었습니다. 적절한 데이터베이스 사용자에게 액세스 권한을 부여하여 SageMaker AI와 함께 작동하도록 Aurora MySQL DB 클러스터 설정을 계속하세요.

사전 구축된 SageMaker AI 구성 요소를 사용하는 대신 훈련에 SageMaker AI 모델을 사용하려면 아래 Amazon S3 for SageMaker AI를 사용하도록 Aurora MySQL DB 클러스터 설정(선택 사항)에 설명된 것과 같이 Amazon S3 버킷을 Aurora MySQL DB 클러스터에 추가해야 합니다.

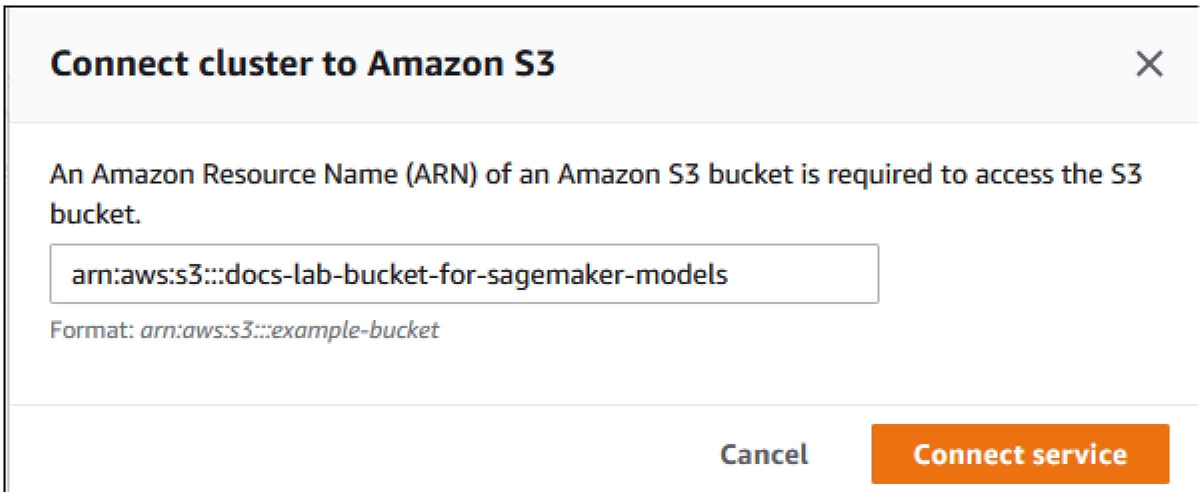
Amazon S3 for SageMaker AI를 사용하도록 Aurora MySQL DB 클러스터 설정(선택 사항)

SageMaker AI에서 제공하는 사전 구축된 구성 요소를 사용하는 대신 자체 모델과 함께 SageMaker AI를 사용하려면 Aurora MySQL DB 클러스터에서 사용할 Amazon S3 버킷을 설정해야 합니다. Amazon S3 버킷 생성에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 생성을 참조하세요.

SageMaker AI용 Amazon S3 버킷을 사용하도록 Aurora MySQL DB 클러스터를 설정하는 방법

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.

2. Amazon RDS 탐색 메뉴에서 데이터베이스를 선택한 다음 SageMaker AI 서비스에 연결하려는 Aurora MySQL DB 클러스터를 선택합니다.
3. 연결 및 보안(Connectivity & security) 탭을 선택합니다.
4. IAM 역할 관리 섹션으로 스크롤한 후 이 클러스터에 연결할 서비스 선택을 선택합니다. 선택기에서 Amazon S3를 선택합니다.
5. Connect service(서비스 연결)를 선택합니다.
6. Amazon S3에 클러스터 연결 대화 상자에서 다음 이미지와 같이 Amazon S3 버킷의 ARN을 입력합니다.



7. 서비스 연결을 선택하여 이 프로세스를 완료합니다.

SageMaker AI와 함께 Amazon S3 버킷을 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide의 Specify an Amazon S3 Bucket to Upload Training Datasets and Store Output Data를 참조하세요. SageMaker AI를 사용하는 방법에 대해 자세히 알아보려면 Amazon SageMaker AI Developer Guide의 Get Started with Amazon SageMaker AI Notebook Instances를 참조하세요.

데이터베이스 사용자에게 Aurora 기계 학습에 대한 액세스 권한 부여

데이터베이스 사용자에게 Aurora 기계 학습 함수를 간접 호출할 수 있는 권한을 부여해야 합니다. 권한 부여 방법은 아래에 설명된 대로 Aurora MySQL DB 클러스터에 사용하는 MySQL 버전에 따라 다릅니다. 이 작업을 수행하는 방법은 Aurora MySQL DB 클러스터에서 사용하는 MySQL 버전에 따라 달라집니다.

- Aurora MySQL 버전 3(MySQL 8.0 호환)의 경우 데이터베이스 사용자에게 적절한 데이터베이스 역할을 부여해야 합니다. 자세한 내용은 MySQL 8.0 Reference Manual의 Using Roles를 참조하세요.

- Aurora MySQL 버전 2(MySQL 5.7 호환)의 경우 데이터베이스 사용자에게 권한을 부여합니다. 자세한 내용은 MySQL 5.7 참조 매뉴얼의 Access Control and Account Management를 참조하세요.

다음 테이블에는 데이터베이스 사용자가 기계 학습 기능을 사용하는 데 필요한 역할과 권한이 나와 있습니다.

Aurora MySQL 버전 3(역할)	Aurora MySQL 버전 2(권한)
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

Amazon Bedrock 함수에 대한 액세스 권한 부여

데이터베이스 사용자에게 Amazon Bedrock 함수에 대한 액세스 권한을 부여하려면 다음 SQL 문을 사용하세요.

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

Amazon Bedrock을 사용하기 위해 생성하는 함수에 대한 `EXECUTE` 권한도 데이터베이스 사용자에게 부여해야 합니다.

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

마지막으로 데이터베이스 사용자의 역할은 다음과 같이 `AWS_BEDROCK_ACCESS`로 설정해야 합니다.

```
SET ROLE AWS_BEDROCK_ACCESS;
```

이제 Amazon Bedrock 함수를 사용할 수 있습니다.

Amazon Comprehend 함수에 대한 액세스 권한 부여

데이터베이스 사용자에게 Amazon Comprehend 함수에 대한 액세스 권한을 부여하려면 사용 중인 Aurora MySQL 버전에 해당하는 문을 사용하세요.

- Aurora MySQL 버전 3(MySQL 8.0 호환)

```
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL 버전 2(MySQL 5.7 호환)

```
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
```

이제 Amazon Comprehend 함수를 사용할 수 있습니다. 사용 예시는 [Aurora MySQL DB 클러스터와 함께 Amazon Comprehend 사용](#) 섹션을 참조하세요.

SageMaker AI 함수에 대한 액세스 권한 부여

데이터베이스 사용자에게 SageMaker AI 함수에 대한 액세스 권한을 부여하려면 사용 중인 Aurora MySQL 버전에 해당하는 문을 사용하세요.

- Aurora MySQL 버전 3(MySQL 8.0 호환)

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL 버전 2(MySQL 5.7 호환)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

SageMaker AI로 작업하기 위해 만드는 함수에 대한 EXECUTE 권한도 데이터베이스 사용자에게 부여해야 합니다. SageMaker AI 엔드포인트의 서비스를 간접 호출하기 위해 `db1.anomoly_score` 및 `db2.company_forecasts` 함수를 만들었다고 가정해 보겠습니다. 다음 예시에서처럼 실행 권한을 부여합니다.

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

이제 SageMaker AI 함수를 사용할 수 있습니다. 사용 예시는 [Aurora MySQL DB 클러스터와 함께 SageMaker AI 사용](#) 섹션을 참조하세요.

Aurora MySQL DB 클러스터와 함께 Amazon Bedrock 사용

Amazon Bedrock을 사용하려면 모델을 간접 호출하는 사용자 정의 함수(UDF)를 Aurora MySQL 데이터베이스에 생성해야 합니다. 자세한 내용은 Amazon Bedrock 사용 설명서의 Amazon Bedrock에서 지원되는 모델을 참조하세요.

UDF는 다음 구문을 사용합니다.

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Amazon Bedrock 함수는 `RETURNS JSON`을 지원하지 않습니다. 필요한 경우 `CONVERT` 또는 `CAST`를 사용하여 `TEXT`에서 `JSON`으로 변환할 수 있습니다.
- `CONTENT_TYPE` 또는 `ACCEPT`를 지정하지 않으면 기본값은 `application/json`입니다.
- `TIMEOUT_MS`를 지정하지 않으면 `aurora_ml_inference_timeout`의 값이 사용됩니다.

예를 들어, 다음 UDF는 Amazon Titan Text Express 모델을 간접 호출합니다.

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

DB 사용자가 이 함수를 사용할 수 있도록 허용하려면 다음 SQL 명령을 사용하세요.

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

그러면 사용자는 다음 예시에서와 같이 다른 함수처럼 `invoke_titan`을 직접 호출할 수 있습니다. Amazon Titan 텍스트 모델에 따라 요청 본문의 형식을 지정해야 합니다.

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
            "Price": "$15"
        },
        {
            "Row Number": "4",
```
```

```
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}
```

사용하는 다른 모델의 경우 요청 본문의 형식을 해당 모델에 맞게 적절히 지정해야 합니다. 자세한 내용은 Amazon Bedrock 사용 설명서의 파운데이션 모델의 추론 파라미터를 참조하세요.

## Aurora MySQL DB 클러스터와 함께 Amazon Comprehend 사용

Aurora MySQL의 경우 Aurora 기계 학습은 Amazon Comprehend 및 텍스트 데이터를 사용하기 위한 다음 두 가지 내장 함수를 제공합니다. 분석할 텍스트(`input_data`)를 제공하고 언어(`language_code`)를 지정합니다.

aws_comprehend_detect_sentiment

이 함수는 텍스트에 긍정적, 부정적, 중립적 또는 혼합된 감정 태도가 있는 것으로 식별합니다. 이 함수의 참조 문서는 다음과 같습니다.

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

자세히 알아보려면 Amazon Comprehend 개발자 안내서의 Sentiment(감정)를 참조하세요.

aws_comprehend_detect_sentiment_confidence

이 함수는 주어진 텍스트에 대해 감지된 감정의 신뢰도를 측정합니다. aws_comprehend_detect_sentiment 함수에서 텍스트에 할당한 감정의 신뢰도를 나타내는 값(유형,
```

double)을 반환합니다. 신뢰도는 0과 1 사이의 통계 지표입니다. 신뢰도가 높을수록 결과에 더 많은 가중치를 줄 수 있습니다. 함수 설명서의 요약은 다음과 같습니다.

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

두 함수(aws_comprehend_detect_sentiment_confidence, aws_comprehend_detect_sentiment) 모두 아무것도 지정되지 않을 경우 max_batch_size는 기본값인 25를 사용합니다. 배치 크기는 항상 0보다 커야 합니다. max_batch_size를 사용하여 Amazon Comprehend 함수 호출의 성능을 조정할 수 있습니다. 대용량 배치는 Aurora MySQL DB 클러스터의 더 큰 메모리 사용으로 인한 더 빠른 성능을 상쇄합니다. 자세한 내용은 Aurora MySQL을 사용하는 Aurora 기계 학습의 성능 고려 사항 단원을 참조하십시오.

Amazon Comprehend의 감성 감지 함수에 대한 파라미터 및 반환 유형에 관한 자세한 내용은 DetectSentiment를 참조하세요.

Example 예: Amazon Comprehend 함수를 사용하는 간단한 쿼리

다음은 고객이 지원 팀에 얼마나 만족하는지 확인하기 위해 이 두 함수를 호출하는 간단한 쿼리의 예입니다. 도움 요청이 있을 때마다 고객 피드백을 저장하는 데이터베이스 테이블(support)이 있다고 가정해 봅시다. 이 예시 쿼리는 두 내장 함수를 테이블 feedback 열의 텍스트에 모두 적용하고 결과를 출력합니다. 함수에서 반환되는 신뢰도 값은 0.0~1.0의 double입니다. 더 읽기 쉬운 출력을 위해 이 쿼리는 결과를 소수점 6개로 반올림합니다. 더 쉽게 비교할 수 있도록 이 쿼리는 신뢰도가 가장 높은 결과부터 먼저 결과를 내림차순으로 정렬합니다.

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+----------------------------------------------------------+-----------+------------+
| Customer feedback                                        | Sentiment | Confidence |
+----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!            | POSITIVE  |   0.999771 |
| The latest version of this product stinks!               | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.      | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.  | MIXED     |   0.957958 |
```

```
| Your support tech helped me in fifteen minutes.          | POSITIVE  |   0.949491 |
| My problem was never resolved!                           | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?   | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                             | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                     | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.            | NEGATIVE  |   0.481805 |
+----------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example 예: 특정 신뢰도를 초과하는 텍스트의 평균 감정 결정

일반적인 Amazon Comprehend 쿼리는 감성이 특정 값이고 신뢰도 수준이 특정 숫자보다 더 큰 행을 찾습니다. 예를 들어 다음 쿼리에서는 데이터베이스 내 문서의 평균 감성을 확인하는 방법을 보여줍니다. 이 쿼리에서는 평가 신뢰도가 80% 이상인 문서만 고려합니다.

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

Aurora MySQL DB 클러스터와 함께 SageMaker AI 사용

Aurora MySQL DB 클러스터에서 SageMaker AI 기능을 사용하려면 SageMaker AI 엔드포인트에 대한 직접 호출 및 해당 추론 기능을 내장하는 저장 함수를 만들어야 합니다. 이렇게 하려면 일반적으로 Aurora MySQL DB 클러스터의 다른 처리 작업과 동일한 방식으로 MySQL의 `CREATE FUNCTION`을 사용합니다.

추론을 위해 SageMaker AI에 배포된 모델을 사용하려면 저장 함수에 대해 MySQL 데이터 정의 언어 (DDL) 문을 사용하여 사용자 정의 함수를 만듭니다. 각 저장 함수는 이 모델을 호스팅하는 SageMaker AI 엔드포인트를 대표합니다. 이러한 함수를 정의할 때 모델에 대한 입력 파라미터, 간접 호출할 특정 SageMaker AI 엔드포인트, 반환 유형을 지정합니다. 이 함수에서는 입력 파라미터에 모델을 적용한 후에 SageMaker AI 엔드포인트가 계산한 추론을 반환합니다.

모든 Aurora Machine Learning 저장 함수에서는 숫자 형식 또는 `VARCHAR`를 반환합니다. `BIT`를 제외한 모든 숫자형을 사용할 수 있습니다. `JSON`, `BLOB`, `TEXT`, `DATE` 등 다른 유형은 허용되지 않습니다.

다음 예시는 SageMaker AI를 사용하기 위한 `CREATE FUNCTION` 구문을 보여줍니다.

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

이는 일반 `CREATE FUNCTION` DDL 명령문의 확장입니다. SageMaker AI 함수를 정의하는 `CREATE FUNCTION` 문에서 함수 본문을 정의해서는 안 됩니다. 그 대신에 키워드인 `ALIAS`를 함수 본문이 일반적으로 배치되는 곳에 지정합니다. 현재 Aurora Machine Learning은 이 확장 구문에 대해 `aws_sagemaker_invoke_endpoint`만 지원합니다. `endpoint_name` 파라미터를 지정해야 합니다. SageMaker AI 엔드포인트는 모델마다 다양한 특성이 있을 수 있습니다.

> **Note**
>
> `CREATE FUNCTION`에 대한 자세한 내용은 MySQL 8.0 참조 매뉴얼의 [CREATE PROCEDURE 및 CREATE FUNCTION 문에 관한 문서]를 참조하세요.

`max_batch_size` 파라미터는 선택 항목입니다. 기본적으로 최대 배치 크기는 10,000입니다. 함수에서 이 파라미터를 사용하여 SageMaker AI에 대한 배치 요청에서 처리되는 입력의 최대 개수를 제한할 수 있습니다. `max_batch_size` 파라미터는 너무 큰 입력으로 인한 오류를 방지하거나 SageMaker AI가 응답을 더 빨리 반환하게 하는 데 도움이 될 수 있습니다. 이 파라미터는 SageMaker AI 요청 처리에 사용되는 내부 버퍼의 크기에 영향을 미칩니다. `max_batch_size`에 너무 큰 값을 지정하면 DB 인스턴스에 상당한 메모리 오버헤드가 발생할 수 있습니다.

`MANIFEST` 설정을 기본값인 `OFF`로 두는 것이 좋습니다. 사용자는 `MANIFEST ON` 옵션을 사용할 수 있지만 일부 SageMaker AI 기능에서는 이 옵션을 통해 내보낸 CSV를 직접 사용할 수 없습니다. 매니페스트 형식은 SageMaker AI의 예상 매니페스트 형식과 호환되지 않습니다.

각 SageMaker AI 모델에 대해 별도 저장 함수를 만듭니다. 이렇게 함수를 모델에 매핑해야 하는 이유는 엔드포인트가 특정 모델과 연결되어 있고 각 모델은 서로 다른 파라미터를 받아들이기 때문입니다. 모델 입력 및 모델 출력 유형에 SQL 유형을 사용하면 AWS 서비스 간 데이터 전달 과정에서 발생하는 유형 변환 오류를 방지하는 데 도움이 됩니다. 모델을 누가 적용할 수 있는지 제어할 수 있습니다. 또한 최대 배치 크기를 나타내는 파라미터를 지정하여 런타임 특성을 제어할 수 있습니다.

현재 모든 Aurora Machine Learning 함수에는 NOT DETERMINISTIC 속성이 있습니다. 이 속성을 명시적으로 지정하지 않으면 Aurora가 NOT DETERMINISTIC으로 자동 설정합니다. 이러한 요구 사항이 있는 이유는 SageMaker AI 모델이 데이터베이스에 대한 알림 없이 변경될 수 있기 때문입니다. 변경된 경우 Aurora Machine Learning 함수에 대한 호출을 통해 단일 트랜잭션 내에서 동일 입력에 대해 다른 결과가 반환될 수 있습니다.

CONTAINS SQL 문에서는 NO SQL, READS SQL DATA, MODIFIES SQL DATA 또는 CREATE FUNCTION 특성을 사용할 수 없습니다.

다음은 이상을 감지하기 위해 SageMaker AI 엔드포인트를 간접 호출하는 방법에 관한 예시입니다. SageMaker AI 엔드포인트 random-cut-forest-model이 있습니다. 이에 상응하는 모델은 이미 random-cut-forest 알고리즘의 교육을 받았습니다. 각 입력에 대해 이 모델은 이상 점수를 반환합니다. 이 예에서는 평균 점수 기준 표준 편차 3개를 초과하는 점수(약 99.9 백분위수)를 지닌 데이터 포인트를 보여줍니다.

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

문자열을 반환하는 SageMaker AI 함수에 대한 문자 집합 요구 사항

문자열을 반환하는 SageMaker AI 함수에 대한 반환 유형으로 utf8mb4 문자 집합을 지정하는 것이 좋습니다. 그렇게 하기 어렵다면 utf8mb4 문자 집합에서 나타내는 값을 담을 수 있을 만큼 충분히 큰 문자열 길이를 반환 유형에 사용하십시오. 다음 예에서는 함수에 utf8mb4 문자 집합을 선언하는 방법을 보여줍니다.

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

현재 문자열을 반환하는 각 SageMaker AI 함수에서는 반환 값에 대해 utf8mb4 문자 집합을 사용합니다. 반환 값에서는 SageMaker AI 함수가 반환 유형에 다른 문자 집합을 묵시적으로 또는 명시적으로 선언한다 하더라도 이러한 문자 집합을 사용합니다. SageMaker AI 함수가 반환 값에 대해 다른 문자 집합을 선언하는 경우 반환된 데이터는 충분히 길지 않은 테이블 열에 저장하면 자동으로 잘릴 수 있습니다. 예를 들어 DISTINCT 절을 이용한 쿼리로 인해 임시 테이블이 생성됩니다. 따라서 SageMaker AI 함수 결과는 쿼리 중에 문자열이 내부적으로 처리되는 방식으로 인해 잘릴 수 있습니다.

SageMaker AI 모델 훈련을 위해 Amazon S3로 데이터 내보내기(고급)

제공된 알고리즘 중 일부를 사용하여 Aurora 기계 학습과 SageMaker AI를 시작하고, 팀의 데이터 과학자가 SQL 코드와 함께 사용할 수 있는 SageMaker AI 엔드포인트를 제공하는 것이 좋습니다. 다음에서는 자체 SageMaker AI 모델 및 Aurora MySQL DB 클러스터와 함께 자체 Amazon S3 버킷을 사용하는 방법에 대한 간략한 정보를 찾을 수 있습니다.

기계 학습은 학습과 추론이라는 두 가지 주요 단계로 구성됩니다. SageMaker AI 모델을 훈련하려면 데이터를 Amazon S3 버킷으로 내보냅니다. Amazon S3 버킷은 Jupyter SageMaker AI 노트북 인스턴스가 모델을 배포하기 전에 모델을 훈련하는 데 사용합니다. `SELECT INTO OUTFILE S3` 문을 사용하여 Aurora MySQL DB 클러스터에서 데이터를 쿼리한 후 Amazon S3 버킷에 저장된 텍스트 파일에 직접 저장할 수 있습니다. 그런 다음 노트북 인스턴스는 교육용 Amazon S3 버킷에서 데이터를 소비합니다.

Aurora Machine Learning은 Aurora MySQL의 기존 `SELECT INTO OUTFILE` 구문을 확장하여 데이터를 CSV 형식으로 내보냅니다. 생성된 CSV 파일은 교육 목적으로 이 형식이 필요한 모델이 직접 소비할 수 있습니다.

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

이 확장에서는 표준 CSV 형식을 지원합니다.

- `TEXT` 형식은 기존 MySQL 내보내기 형식과 동일합니다. 이것은 기본 형식입니다.
- `CSV` 형식은 새로 도입된 형식으로서, RFC-4180의 사양을 따릅니다.
- 선택적 키워드인 `HEADER`를 지정하는 경우 출력 파일에는 헤더 행이 한 줄 포함됩니다. 이 헤더 행의 레이블은 `SELECT` 문의 열 이름에 해당합니다.
- `CSV` 및 `HEADER`라는 키워드를 식별자로 계속 사용할 수 있습니다.

`SELECT INTO`의 확장 구문 및 문법은 현재 다음과 같습니다.

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
```

```
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

Aurora MySQL을 사용하는 Aurora 기계 학습의 성능 고려 사항

Amazon Bedrock, Amazon Comprehend 및 SageMaker AI 서비스는 Aurora 기계 학습 함수에서 간접 호출할 때 대부분의 작업을 수행합니다. 즉, 필요에 따라 독립적으로 리소스를 확장할 수 있습니다. Aurora MySQL DB 클러스터의 경우 함수 호출을 최대한 효율적으로 만들 수 있습니다. 다음은 Aurora 기계 학습을 사용할 때 참고해야 할 몇 가지 성능 고려 사항입니다.

모델 및 프롬프트

Amazon Bedrock을 사용할 때의 성능은 사용하는 모델 및 프롬프트에 따라 크게 달라집니다. 사용 사례에 가장 적합한 모델과 프롬프트를 선택하세요.

쿼리 캐시

Aurora MySQL 쿼리 캐시는 Aurora 기계 학습 함수에 대해 작동하지 않습니다. Aurora MySQL은 Aurora 기계 학습 함수를 호출하는 모든 SQL 문에 대한 쿼리 결과를 쿼리 캐시에 저장하지 않습니다.

Aurora Machine Learning 함수 호출을 위한 배치 최적화

Aurora 클러스터에서 영향을 미칠 수 있는 주요 Aurora Machine Learning 성능 측면은 Aurora Machine Learning 저장 함수에 대한 호출의 배치 모드 설정입니다. 기계 학습 함수는 일반적으로 상당한 오버헤드가 필요하므로 각 행에 대해 외부 서비스를 별도로 호출하는 것은 실용적이지 않습니다. Aurora Machine Learning은 많은 행에 대한 외부 Aurora Machine Learning 서비스 호출을 단일 배치로 결합하여 이러한 오버헤드를 최소화할 수 있습니다. Aurora Machine Learning에서 모든 입력 행에 대한 응답을 수신한 다음 응답을 실행 중인 쿼리에 한 번에 한 행씩 다시 전달합니다. 이러한 최적화를 통해 결과를 변경하지 않고도 Aurora 쿼리의 처리량 및 지연 시간을 개선할 수 있습니다.

SageMaker AI 엔드포인트에 연결된 Aurora 저장 함수를 만들 때 배치 크기 파라미터를 정의합니다. 이 파라미터는 SageMaker AI에 대한 모든 기본 직접 호출을 위해 전송되는 행의 수에 영향을 미칩니다. 대량의 행을 처리하는 쿼리의 경우 각 행에 대해 별도의 SageMaker AI 직접 호출을 수행하는 데 드는 오버헤드는 상당히 클 수 있습니다. 저장 프로시저에서 처리하는 데이터 세트의 용량이 크면 클수록 배치 크기를 더 크게 확장할 수 있습니다.

배치 모드 최적화를 SageMaker AI 함수에 적용할 수 있는 경우 `EXPLAIN PLAN` 문에서 산출하는 쿼리 계획을 확인함으로써 이를 구별할 수 있습니다. 이 경우 실행 계획의 `extra` 열에는 `Batched` machine learning이 포함됩니다. 다음 예에서는 배치 모드를 사용하는 SageMaker AI 함수 직접 호출을 보여줍니다.

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                    |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
1 row in set, 1 warning (0.01 sec)
```

내장된 Amazon Comprehend 함수 중 하나를 호출할 때 선택 사항인 max_batch_size 파라미터를 지정하여 배치 크기를 제어할 수 있습니다. 이 파라미터는 각 배치에서 처리되는 input_text 값의 최대 수를 제한합니다. 한 번에 여러 항목을 전송함으로써 Aurora 및 Amazon Comprehend 간 왕복 횟수를 줄입니다. 배치 크기 제한은 LIMIT 절을 이용한 쿼리와 같은 상황에서 유용합니다. max_batch_size에 작은 값을 사용함으로써 입력 텍스트를 얻는 횟수보다 더 많이 Amazon Comprehend를 호출하는 것을 방지할 수 있습니다.

Aurora Machine Learning 함수 평가를 위한 배치 최적화는 다음 경우에 적용됩니다.

- 선택 목록 또는 SELECT 문의 WHERE 절 내 함수 호출
- INSERT 및 REPLACE 문의 VALUES 목록 내 함수 호출
- UPDATE 문 내 SET 값의 SageMaker AI 함수

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

Aurora Machine Learning 모니터링

다음 예시와 같이 여러 글로벌 변수를 쿼리하여 Aurora 기계 학습 배치 작업을 모니터링할 수 있습니다.

```
show status like 'Aurora_ml%';
```

`FLUSH STATUS` 문을 사용하여 상태 변수를 재설정할 수 있습니다. 따라서 모든 숫자는 변수를 마지막으로 재설정한 이후의 합계, 평균 등을 나타냅니다.

`Aurora_ml_logical_request_cnt`

마지막 상태 재설정 이후 DB 인스턴스가 Aurora 기계 학습 서비스로 전송되어야 한다고 평가한 논리적 요청의 수입니다. 배치 사용 여부에 따라 이 값은 `Aurora_ml_actual_request_cnt`보다 높을 수 있습니다.

`Aurora_ml_logical_response_cnt`

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 응답 횟수입니다.

`Aurora_ml_actual_request_cnt`

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에 전송하는 총 요청 수입니다.

`Aurora_ml_actual_response_cnt`

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 응답 횟수입니다.

`Aurora_ml_cache_hit_cnt`

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 Aurora 기계 학습 서비스에서 수신하는 총 내부 캐시 히트 횟수입니다.

`Aurora_ml_retry_request_cnt`

마지막 상태 재설정 이후 DB 인스턴스가 Aurora 기계 학습 서비스로 전송한 재시도 요청의 수입니다.

`Aurora_ml_single_request_cnt`

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 비일괄 모드로 평가되는 Aurora 기계 학습 함수의 총 개수입니다.

Aurora 기계 학습 함수에서 직접 호출하는 SageMaker AI 작업의 성능을 모니터링하는 방법에 대한 자세한 내용은 Monitor Amazon SageMaker AI를 참조하세요.

Aurora PostgreSQL과 함께 Amazon Aurora 기계 학습 사용

Amazon Aurora 기계 학습을 Aurora PostgreSQL DB 클러스터와 함께 사용하면 필요에 따라 Amazon Comprehend, Amazon SageMaker AI 또는 Amazon Bedrock을 사용할 수 있습니다. 이러한 서비스는 각각 구체적인 기계 학습 사용 사례를 지원합니다.

Aurora 기계 학습은 특정 AWS 리전 및 Aurora PostgreSQL의 특정 버전에서만 지원됩니다. Aurora 기계 학습을 설정하기 전에 사용 중인 리전에서 Aurora PostgreSQL 버전의 사용 가능 여부를 확인하세요. 세부 정보는 Aurora PostgreSQL을 사용하는 Aurora 기계 학습을 참조하세요.

주제

Aurora PostgreSQL과 함께 Aurora 기계 학습을 사용할 때 요구 사항

AWS 기계 학습 서비스는 자체 프로덕션 환경에서 설정되고 실행되는 관리형 서비스입니다. Aurora 기계 학습은 Amazon Comprehend, SageMaker AI, Amazon Bedrock과의 통합을 지원합니다. Aurora 기계 학습을 사용하도록 Aurora PostgreSQL DB 클러스터를 설정하기 전에 다음 요구 사항 및 사전 조건을 이해해야 합니다.

- Amazon Comprehend, SageMaker AI, Amazon Bedrock 서비스는 Aurora PostgreSQL DB 클러스터와 동일한 AWS 리전에서 실행되어야 합니다. 다른 리전의 Aurora PostgreSQL DB 클러스터에서는 Amazon Comprehend, Amazon SageMaker AI 또는 Amazon Bedrock 서비스를 사용할 수 없습니다.

- Aurora PostgreSQL DB 클러스터가 Amazon Comprehend 및 SageMaker AI 서비스와 다른 Amazon VPC 서비스 기반 Virtual Public Cloud(VPC)에 있는 경우 VPC의 보안 그룹은 대상 Aurora 기계 학습 서비스에 대한 아웃바운드 연결을 허용해야 합니다. 자세한 내용은 Amazon Aurora에서 다른 AWS 서비스로의 네트워크 통신 활성화 단원을 참조하십시오.
- SageMaker AI의 경우 추론에 사용할 기계 학습 구성 요소를 설정하고 사용할 준비가 되어 있어야 합니다. Aurora PostgreSQL DB 클러스터에 대한 구성 프로세스 동안 SageMaker AI 엔드포인트의 Amazon 리소스 이름(ARN)을 사용할 수 있어야 합니다. 팀의 데이터 과학자는 SageMaker AI와 협력하여 모델을 준비하고 그 외의 작업을 처리하는 데 가장 잘 대처할 수 있을 것입니다. Amazon SageMaker AI를 시작하려면 Get Started with Amazon SageMaker AI를 참조하세요. 추론 및 엔드포인트에 대한 자세한 내용은 Real-time inference(실시간 추론)를 참조하세요.
- Amazon Bedrock의 경우 Aurora PostgreSQL DB 클러스터 구성 과정에서 추론에 사용할 수 있는 Bedrock 모델의 모델 ID가 있어야 합니다. 팀의 데이터 과학자가 Bedrock을 사용하여 활용할 모델을 결정하고, 필요한 경우 미세 조정하고, 그 외의 작업을 처리하는 업무를 가장 잘 수행할 수 있습니다. Amazon Bedrock을 시작하려면 Bedrock을 설정하는 방법을 참조하세요.
- Amazon Bedrock 사용자가 모델을 사용하려면 우선 모델에 대한 액세스 권한을 요청해야 합니다. 텍스트, 채팅, 이미지 생성을 위한 추가 모델을 추가하려면 Amazon Bedrock의 모델에 대한 액세스 권한을 요청해야 합니다. 자세한 내용은 모델 액세스를 참조하세요.

Aurora PostgreSQL을 사용하는 Aurora 기계 학습의 지원 기능 및 제한 사항

Aurora 기계 학습은 `ContentType`의 `text/csv` 값을 통해 쉼표로 구분된 값(CSV) 형식을 읽고 쓸 수 있는 모든 SageMaker AI 엔드포인트를 지원합니다. 현재 이 형식을 허용하는 내장 SageMaker AI 알고리즘은 다음과 같습니다.

- Linear Learner
- 랜덤 컷 포레스트
- XGBoost

  이러한 알고리즘에 대해 자세히 알아보려면 Amazon SageMaker AI Developer Guide의 Choose an Algorithm을 참조하세요.

Aurora 기계 학습과 함께 Amazon Bedrock을 사용하는 경우 다음과 같은 제한 사항이 적용됩니다.

- 사용자 정의 함수(UDF)는 Amazon Bedrock과 상호 작용하는 기본 방법을 제공합니다. UDF에는 특정 요청 또는 응답 요구 사항이 없으므로, 모든 모델을 사용할 수 있습니다.

- UDF를 사용하여 원하는 워크플로를 구축할 수 있습니다. 예를 들어, 기본 프리미티브(예: `pg_cron`)를 결합하여 쿼리를 실행하고 데이터를 가져오며 추론을 생성하고 테이블에 기록하여 쿼리를 직접 제공할 수 있습니다.
- UDF는 일괄 또는 병렬 호출을 지원하지 않습니다.
- Aurora 기계 학습 확장은 벡터 인터페이스를 지원하지 않습니다. 확장의 일환으로 모델 응답의 임베딩을 `float8[]` 형식으로 출력하여 Aurora에 저장하는 기능을 사용할 수 있습니다. `float8[]` 사용에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터와 함께 Amazon Bedrock 사용 섹션을 참조하세요.

Aurora 기계 학습을 사용하도록 Aurora PostgreSQL DB 클러스터 설정

Aurora 기계 학습이 Aurora PostgreSQL DB 클러스터와 함께 작동하려면 사용하려는 각 서비스에 대해 AWS Identity and Access Management(IAM) 역할을 생성해야 합니다. IAM 역할을 사용하면 Aurora PostgreSQL DB 클러스터가 클러스터를 대신하여 Aurora 기계 학습 서비스를 사용할 수 있습니다. 또한 Aurora 기계 학습 확장을 설치해야 합니다. 다음 주제에서는 이러한 Aurora 기계 학습 서비스 각각에 대한 설정 절차를 찾을 수 있습니다.

주제

Amazon Bedrock을 사용하도록 Aurora PostgreSQL 설정

다음 절차에서는 먼저 클러스터를 대신하여 Amazon Bedrock을 사용하도록 Aurora PostgreSQL에 권한을 부여하는 IAM 역할 및 정책을 생성합니다. 그런 다음 Aurora PostgreSQL DB 클러스터가 Amazon Bedrock과 함께 작동하기 위해 사용하는 IAM 역할에 정책을 연결합니다. 간소화를 위해 이 절차에서는 AWS Management Console을 사용하여 모든 작업을 완료합니다.

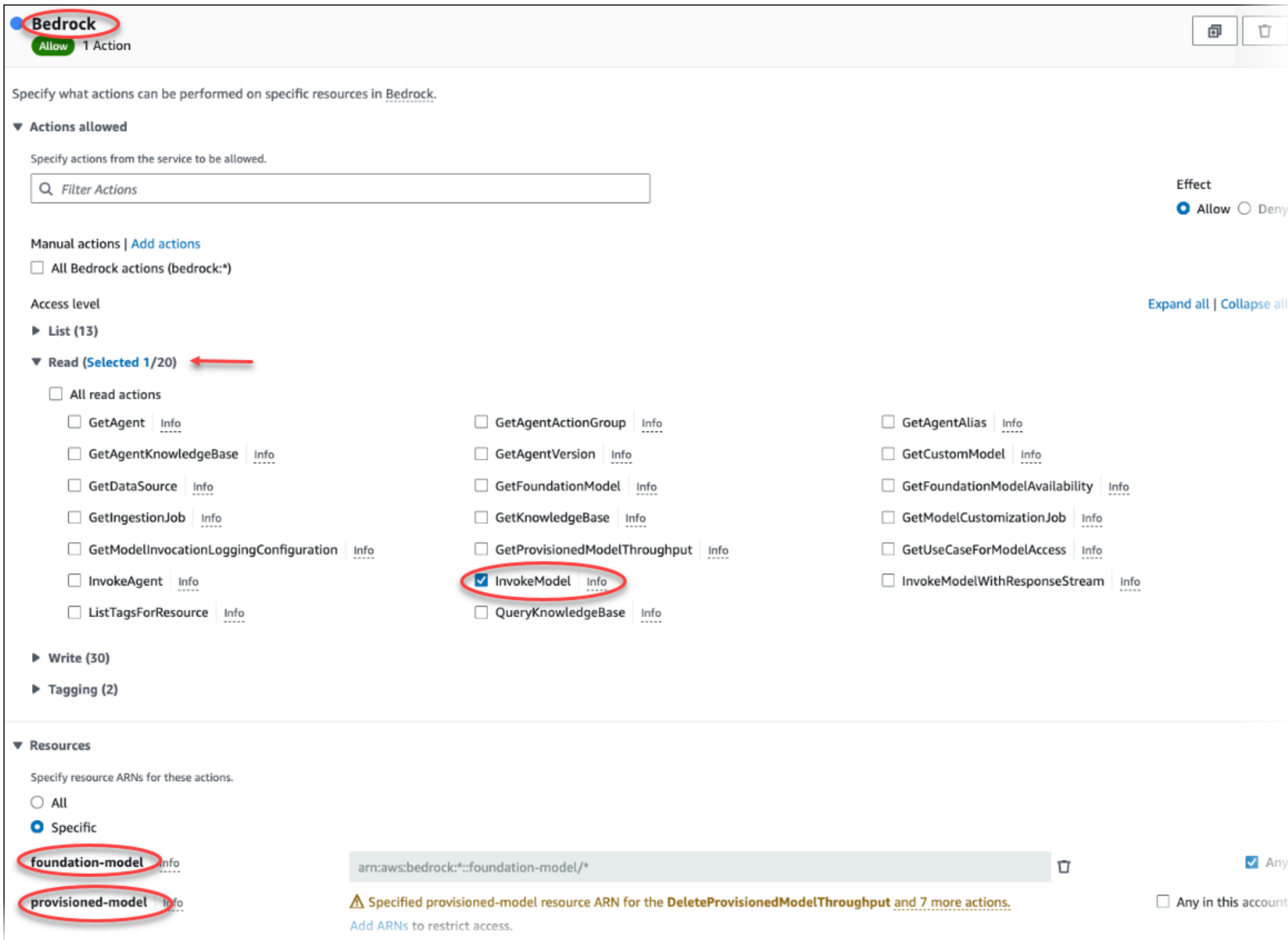
Amazon Bedrock을 사용하도록 Aurora PostgreSQL DB 클러스터를 설정하는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.

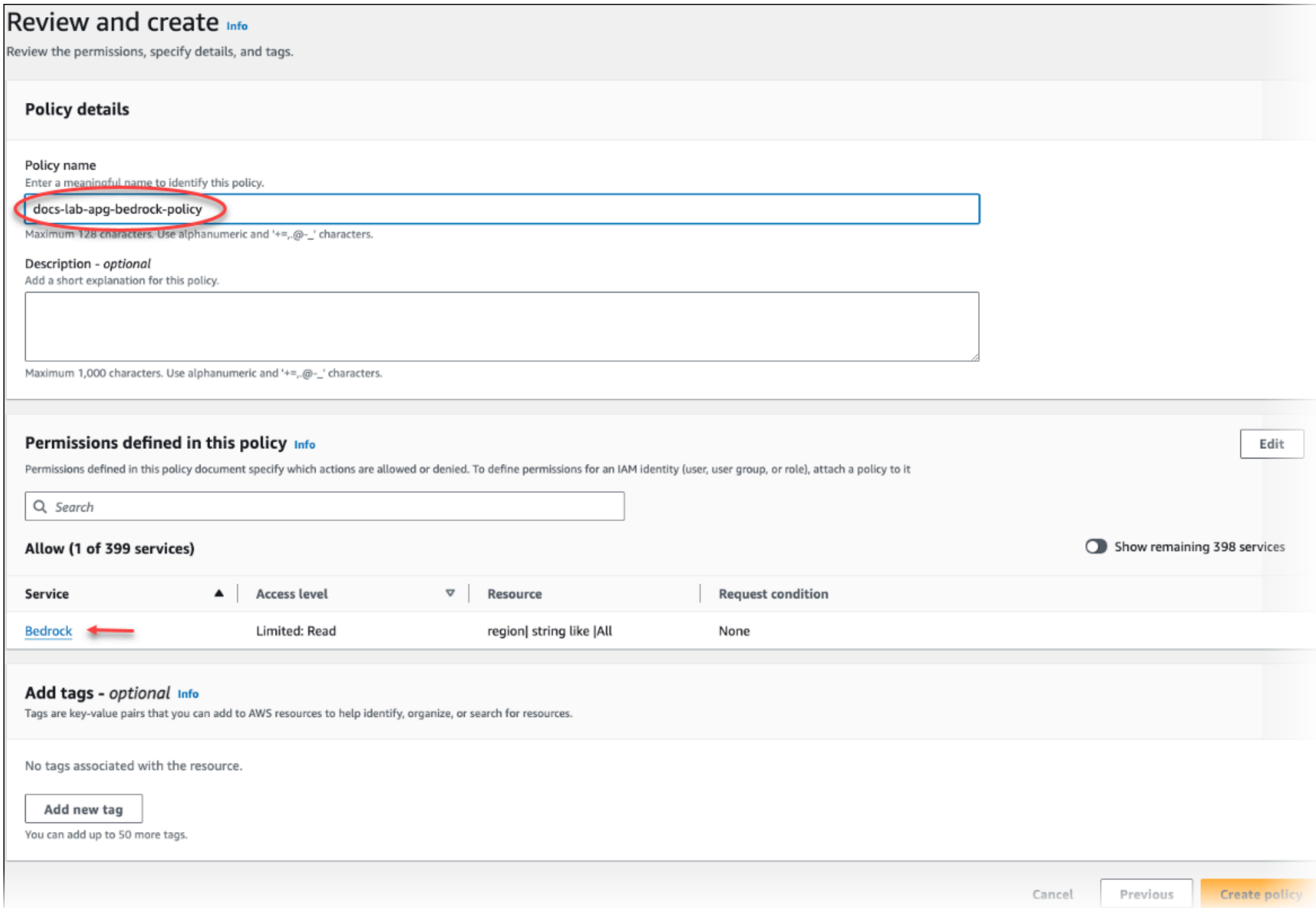
2. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
3. AWS Identity and Access Management(IAM) 콘솔 메뉴에서 정책(액세스 관리 아래)을 선택합니다.

   a. 정책 생성을 선택합니다. 비주얼 에디터 페이지에서 서비스를 선택한 다음 서비스 선택 필드에 Bedrock을 입력합니다. 읽기 액세스 수준을 확장합니다. Amazon Bedrock 읽기 설정에서 InvokeModel을 선택합니다.

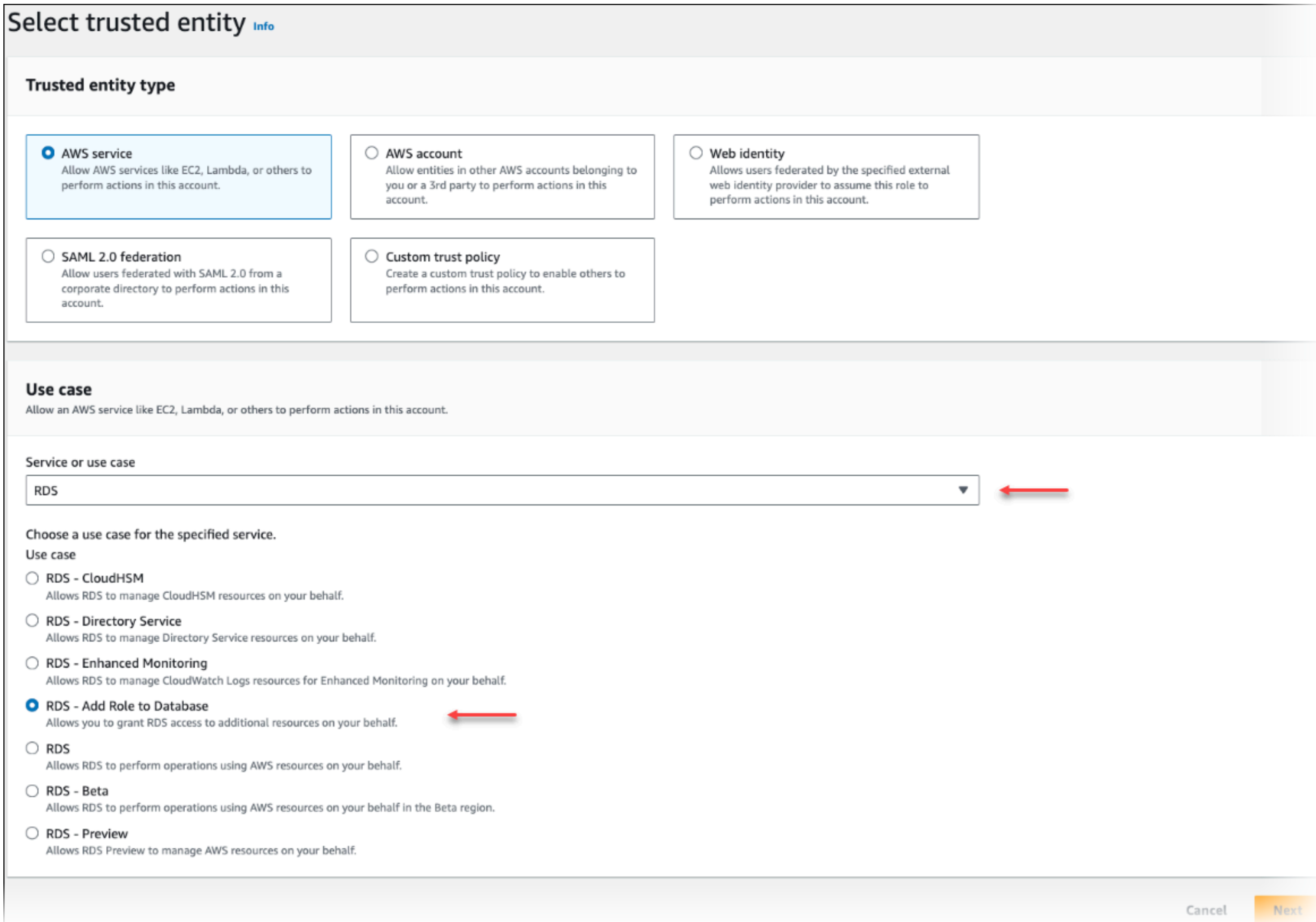
   b. 정책을 통해 읽기 액세스 권한을 부여하려는 파운데이션/프로비저닝 모델을 선택합니다.



4. 다음: 태그를 선택하고 원하는 태그를 정의합니다(선택 사항). 다음: 검토를 선택합니다. 다음 이미지와 같이 정책의 이름과 설명을 입력합니다.

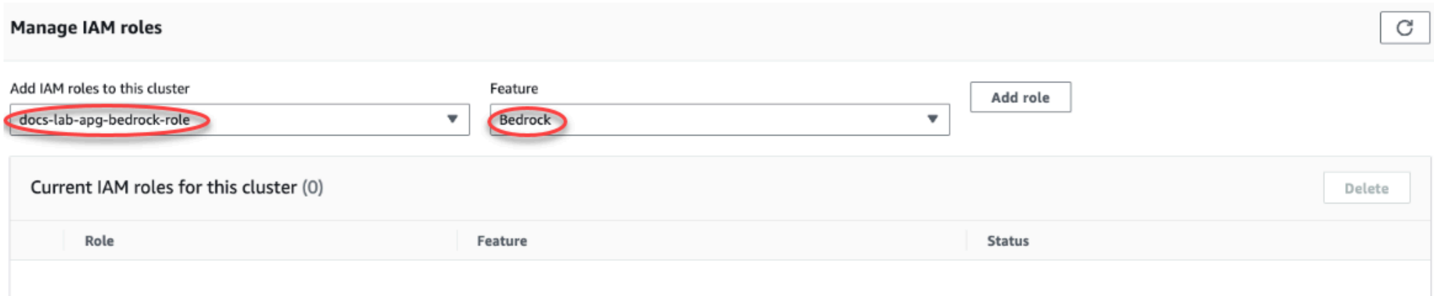


5. 정책 생성을 선택합니다. 정책이 저장되면 콘솔에 알림이 표시됩니다. 정책 목록에서 저장한 정책을 찾을 수 있습니다.
6. IAM 콘솔에서 역할(액세스 관리 아래)을 선택합니다.
7. 역할 생성을 선택합니다.
8. 신뢰할 수 있는 엔터티 선택 페이지에서 AWS 서비스 타일을 선택한 다음 RDS를 선택하여 선택기를 엽니다.
9. RDS – 데이터베이스에 역할 추가를 선택합니다.



10. Next(다음)를 선택합니다. 권한 추가 페이지에서 이전 단계에서 만든 정책을 찾아 나열된 정책 중에서 선택합니다. Next(다음)를 선택합니다.
11. 다음: 검토. 이미지의 이름과 설명을 입력합니다.
12. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
13. Aurora PostgreSQL DB 클러스터가 있는 AWS 리전 위치로 이동합니다.
14. 탐색 창에서 데이터베이스를 선택한 다음 Bedrock과 함께 사용할 Aurora PostgreSQL DB 클러스터를 선택합니다.
15. 연결 및 보안 탭을 선택하고 스크롤하여 페이지의 IAM 역할 관리 섹션을 찾습니다. 이 클러스터에 IAM 역할 추가 선택기에서 이전 단계에서 생성한 역할을 선택합니다. 기능 선택기에서 Bedrock을 선택한 다음 역할 추가를 선택합니다.

역할(정책 포함)은 Aurora PostgreSQL DB 클러스터와 연결됩니다. 프로세스가 완료되면 다음 이미지와 같이 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.



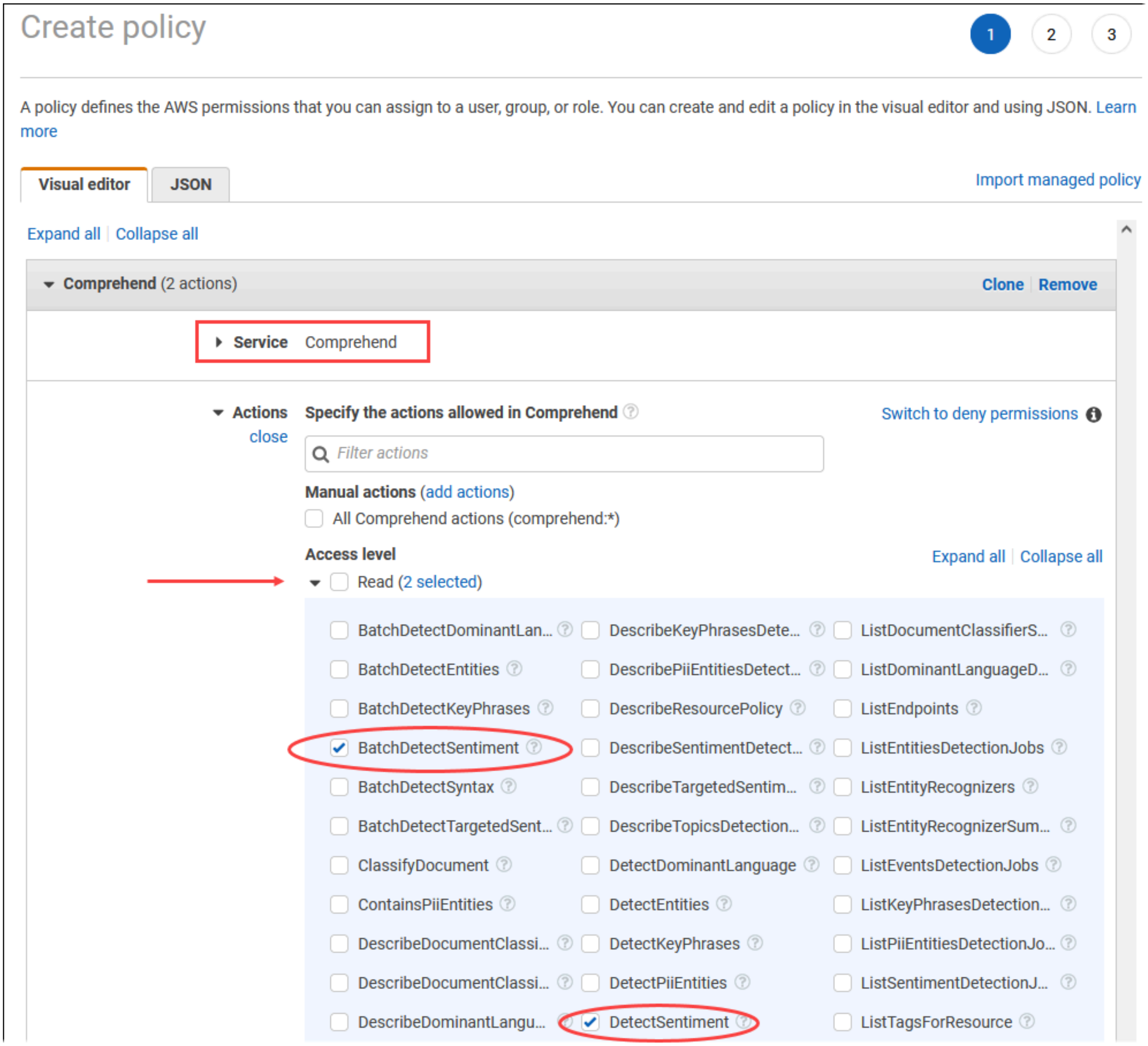
Amazon Bedrock에 대한 IAM 설정이 완료되었습니다. Aurora 기계 학습 확장 설치에 설명된 대로 확장을 설치하여 Aurora 기계 학습과 함께 작동하도록 Aurora PostgreSQL 설정을 계속하세요.

Amazon Comprehend를 사용하도록 Aurora PostgreSQL 설정

다음 절차에서는 먼저 클러스터를 대신하여 Amazon Comprehend를 사용하도록 Aurora PostgreSQL에 권한을 부여하는 IAM 역할 및 정책을 생성합니다. 그런 다음 Aurora PostgreSQL DB 클러스터가 Amazon Comprehend와 함께 작동하기 위해 사용하는 IAM 역할에 정책을 연결합니다. 간단히 하기 위해 이 절차에서는 AWS Management Console을 사용하여 모든 작업을 완료합니다.

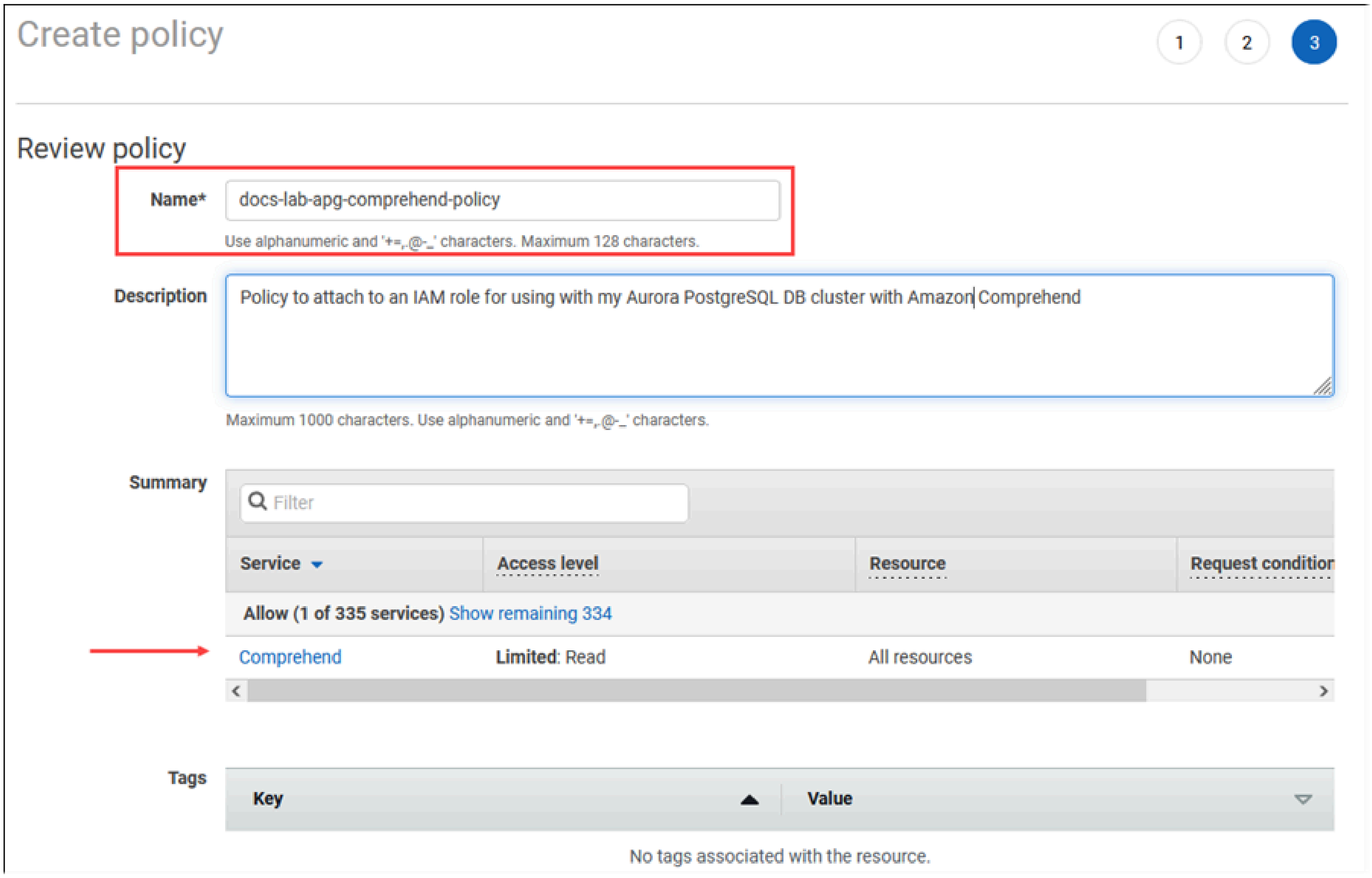
Amazon Comprehend를 사용하도록 Aurora PostgreSQL DB 클러스터를 설정하는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
3. AWS Identity and Access Management(IAM) 콘솔 메뉴에서 정책(액세스 관리 아래)을 선택합니다.

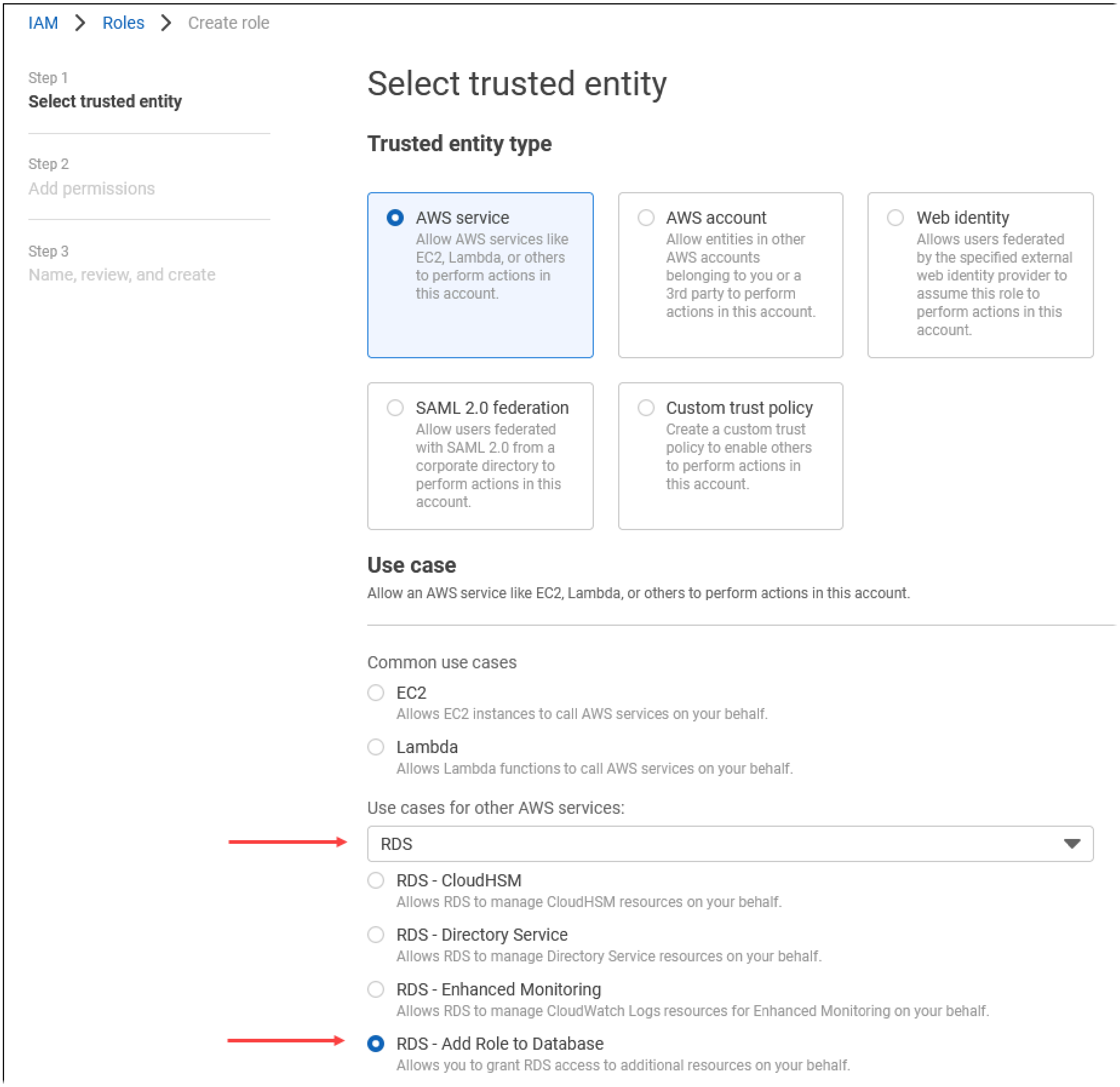


4. 정책 생성을 선택합니다. 비주얼 에디터 페이지에서 서비스를 선택한 다음 서비스 선택 필드에 Comprehend를 입력합니다. 읽기 액세스 수준을 확장합니다. Amazon Comprehend 읽기 설정에서 BatchDetectSentiment 및 DetectSentiment를 선택합니다.

5. 다음: 태그를 선택하고 원하는 태그를 정의합니다(선택 사항). 다음: 검토를 선택합니다. 다음 이미지와 같이 정책의 이름과 설명을 입력합니다.



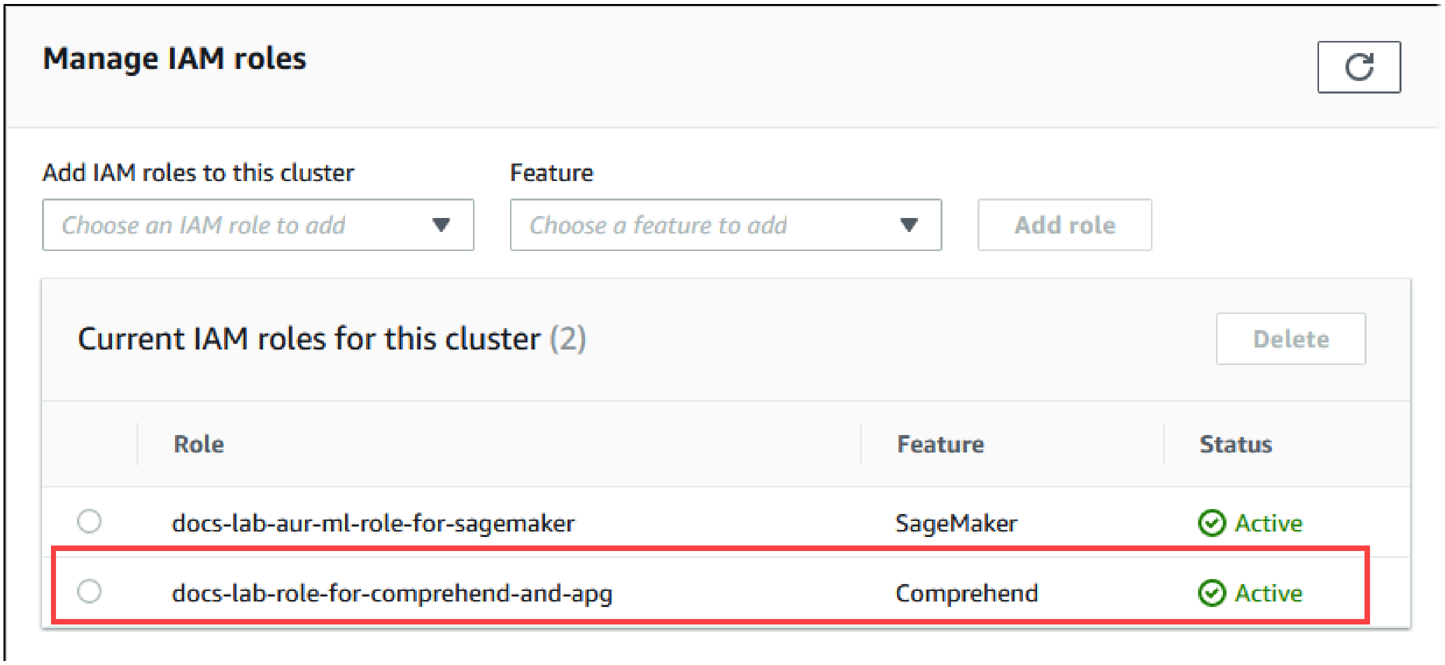
6. 정책 생성을 선택합니다. 정책이 저장되면 콘솔에 알림이 표시됩니다. 정책 목록에서 저장한 정책을 찾을 수 있습니다.
7. IAM 콘솔에서 역할(액세스 관리 아래)을 선택합니다.
8. 역할 생성을 선택합니다.
9. 신뢰할 수 있는 엔터티 선택 페이지에서 AWS 서비스 타일을 선택한 다음 RDS를 선택하여 선택기를 엽니다.
10. RDS – 데이터베이스에 역할 추가를 선택합니다.



11. Next(다음)를 선택합니다. 권한 추가 페이지에서 이전 단계에서 만든 정책을 찾아 나열된 정책 중에서 선택합니다. 다음을 선택합니다.
12. 다음: 검토. 이미지의 이름과 설명을 입력합니다.
13. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
14. Aurora PostgreSQL DB 클러스터가 있는 AWS 리전 위치로 이동합니다.
15. 탐색 창에서 데이터베이스를 선택한 다음 Amazon Comprehend와 함께 사용하려는 Aurora PostgreSQL DB 클러스터를 선택합니다.

16. 연결 및 보안 탭을 선택하고 스크롤하여 페이지의 IAM 역할 관리 섹션을 찾습니다. 이 클러스터에 IAM 역할 추가 선택기에서 이전 단계에서 생성한 역할을 선택합니다. 기능 선택기에서 Comprehend를 선택한 다음 역할 추가를 선택합니다.

역할(정책 포함)은 Aurora PostgreSQL DB 클러스터와 연결됩니다. 프로세스가 완료되면 다음 이미지와 같이 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.



Amazon Comprehend에 대한 IAM 설정이 완료되었습니다. Aurora 기계 학습 확장 설치에 설명된 대로 확장을 설치하여 Aurora 기계 학습과 함께 작동하도록 Aurora PostgreSQL 설정을 계속하세요.

Amazon SageMaker AI를 사용하도록 Aurora PostgreSQL 설정

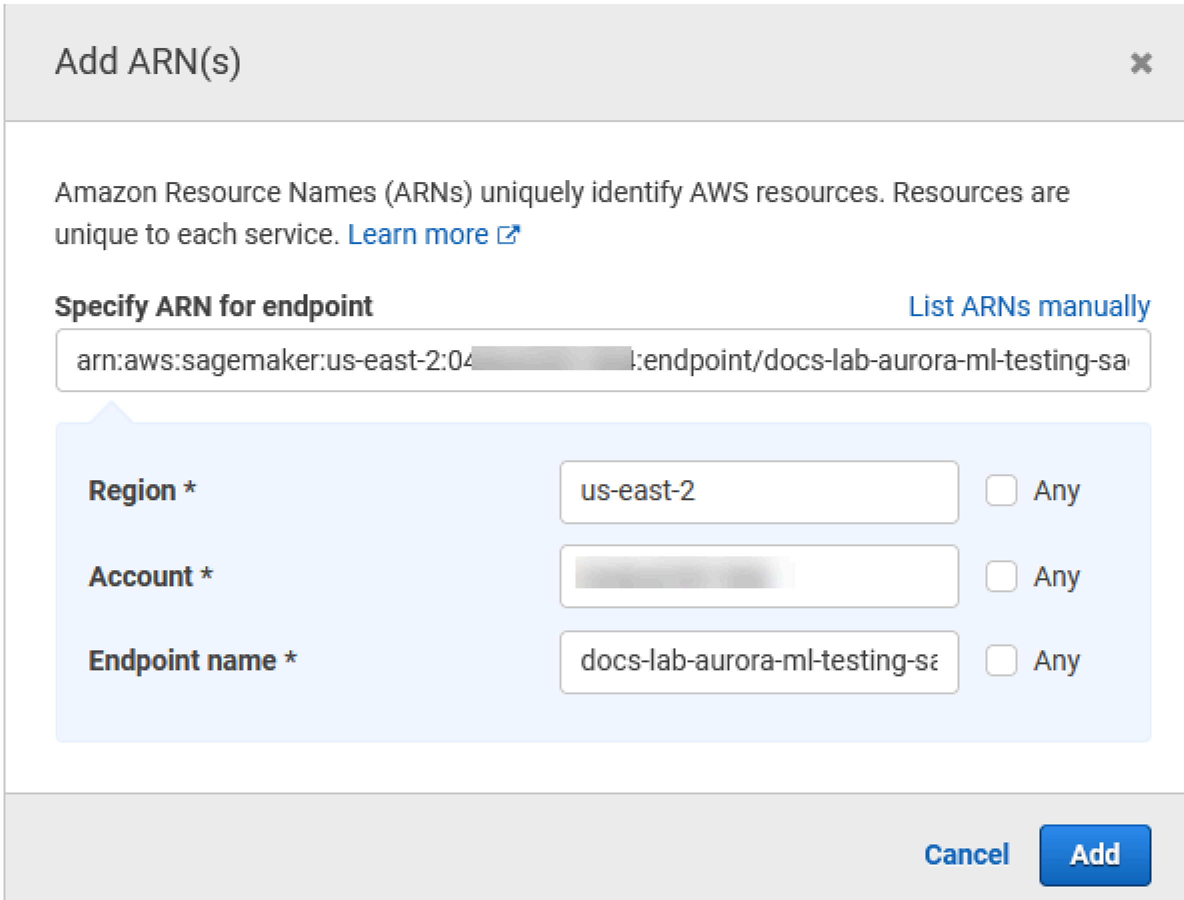
Aurora PostgreSQL DB 클러스터의 IAM 정책과 역할을 만들려면 먼저 SageMaker AI 모델을 설정하고 엔드포인트를 사용할 수 있어야 합니다.

SageMaker AI를 사용하도록 Aurora PostgreSQL DB 클러스터를 설정하는 방법

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.

2. AWS Identity and Access Management(IAM) 콘솔 메뉴에서 정책(액세스 관리 아래)을 선택한 후 정책 생성을 선택합니다. 비주얼 에디터의 서비스에서 SageMaker를 선택합니다. 작업의 경우 읽기 선택기(액세스 수준 아래)를 열고 InvokeEndpoint를 선택합니다. 이렇게 하면 경고 아이콘이 표시됩니다.

3. 리소스 선택기를 열고 InvokeEndpoint 작업에 대한 엔드포인트 리소스 ARN 지정 아래에서 액세스를 제한할 ARN 추가 링크를 선택합니다.
4. SageMaker AI 리소스의 AWS 리전 및 엔드포인트 이름을 입력합니다. AWS 계정이 미리 입력되었습니다.



5. 추가를 선택하여 저장합니다. 다음: 태그 및 다음: 검토를 선택하여 정책 생성 프로세스의 마지막 페이지로 이동합니다.
6. 정책의 이름과 설명을 입력한 다음 정책 생성을 선택합니다. 정책이 생성되고 정책 목록에 추가됩니다. 그러면 콘솔에 알림이 표시됩니다.
7. IAM 콘솔에서 역할을 선택합니다.
8. 역할 생성을 선택합니다.
9. 신뢰할 수 있는 엔터티 선택 페이지에서 AWS 서비스 타일을 선택한 다음 RDS를 선택하여 선택기를 엽니다.
10. RDS – 데이터베이스에 역할 추가를 선택합니다.
11. Next(다음)를 선택합니다. 권한 추가 페이지에서 이전 단계에서 만든 정책을 찾아 나열된 정책 중에서 선택합니다. 다음을 선택합니다.

12. 다음: 검토. 이미지의 이름과 설명을 입력합니다.
13. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
14. Aurora PostgreSQL DB 클러스터가 있는 AWS 리전 위치로 이동합니다.
15. 탐색 창에서 데이터베이스를 선택한 다음 SageMaker AI와 함께 사용하려는 Aurora PostgreSQL DB 클러스터를 선택합니다.
16. 연결 및 보안 탭을 선택하고 스크롤하여 페이지의 IAM 역할 관리 섹션을 찾습니다. 이 클러스터에 IAM 역할 추가 선택기에서 이전 단계에서 생성한 역할을 선택합니다. 기능 선택기에서 SageMaker AI를 선택한 다음 역할 추가를 선택합니다.

역할(정책 포함)은 Aurora PostgreSQL DB 클러스터와 연결됩니다. 프로세스가 완료되면 이 클러스터의 현재 IAM 역할 목록에서 역할을 찾을 수 있습니다.

SageMaker AI에 대한 IAM 설정이 완료되었습니다. Aurora 기계 학습 확장 설치에 설명된 대로 확장을 설치하여 Aurora 기계 학습과 함께 작동하도록 Aurora PostgreSQL 설정을 계속하세요.

Amazon S3 for SageMaker AI를 사용하도록 Aurora PostgreSQL 설정(고급)

SageMaker AI에서 제공하는 사전 구축된 구성 요소를 사용하는 대신 자체 모델과 함께 SageMaker AI를 사용하려면 Aurora PostgreSQL DB 클러스터에서 사용할 Amazon Simple Storage Service(Amazon S3) 버킷을 설정해야 합니다. 이 주제는 고급 주제이며, 이 Amazon Aurora 사용 설명서에 완전히 설명되어 있지는 않습니다. 일반적인 프로세스는 다음과 같이 SageMaker AI 지원 통합 프로세스와 동일합니다.

1. Amazon S3 대한 IAM 정책 및 역할을 생성합니다.
2. Aurora PostgreSQL DB 클러스터의 연결 및 보안 탭에서 IAM 역할과 Amazon S3 가져오기 또는 내보내기를 기능으로 추가합니다.
3. Aurora DB 클러스터의 사용자 지정 DB 클러스터 파라미터 그룹에 역할의 ARN을 추가합니다.

기본적인 사용 정보는 SageMaker AI 모델 훈련을 위해 Amazon S3로 데이터 내보내기(고급) 섹션을 참조하세요.

Aurora 기계 학습 확장 설치

Aurora 기계 학습 확장 `aws_ml 1.0`은 Amazon Comprehend를 간접 호출하는 데 사용할 수 있는 2가지 함수를 제공하고, SageMaker AI 서비스와 `aws_ml 2.0`은 Amazon Bedrock 서비스를 간접 호출하는 데 사용할 수 있는 2가지 추가 함수를 제공합니다. Aurora PostgreSQL DB 클러스터에 확장을 설치하면 해당 기능에 대한 관리자 역할도 생성됩니다.

> **Note**
>
> 이러한 함수의 사용은 [Aurora 기계 학습을 사용하도록 Aurora PostgreSQL DB 클러스터 설정]에 자세히 설명된 대로 Aurora 기계 학습 서비스(Amazon Comprehend, SageMaker AI, Amazon Bedrock)에 대한 IAM 설정을 완료했는지에 따라 달라집니다.

- aws_comprehend.detect_sentiment - 이 함수를 사용하여 Aurora PostgreSQL DB 클러스터의 데이터베이스에 저장된 텍스트에 감성 분석을 적용합니다.
- aws_sagemaker.invoke_endpoint - SQL 코드에서 이 함수를 사용하여 클러스터에서 SageMaker AI 엔드포인트와 통신합니다.
- aws_bedrock.invoke_model - SQL 코드에서 이 함수를 사용하여 클러스터에 있는 Bedrock 모델과 통신합니다. 이 함수의 응답은 TEXT 형식이므로, 모델이 JSON 본문 형식으로 응답하면 이 함수의 출력은 문자열 형식으로 최종 사용자에게 전달됩니다.
- aws_bedrock.invoke_model_get_embeddings – SQL 코드에서 이 함수를 사용하여 JSON 응답 내에서 출력 임베딩을 반환하는 Bedrock 모델을 간접 호출할 수 있습니다. 이 기능은 json-key와 직접 연결된 임베딩을 추출하여 자체 관리형 워크플로로 응답을 간소화하려는 경우에 활용할 수 있습니다.

Aurora PostgreSQL DB 클러스터에 Aurora 기계 학습 확장을 설치하는 방법

- `psql`을 사용하여 Aurora PostgreSQL DB 클러스터의 라이터 인스턴스에 연결합니다. `aws_ml` 확장을 설치할 특정 데이터베이스에 연결합니다.

  ```
  psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
  port=5432 --username=postgres --password --dbname=labdb
  ```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

`aws_ml` 확장을 설치하면 다음과 같이 `aws_ml` 관리 역할과 3개의 스키마도 새로 생성됩니다.

- `aws_comprehend` - Amazon Comprehend 서비스에 대한 스키마와 `detect_sentiment` 함수 소스(`aws_comprehend.detect_sentiment`)

- aws_sagemaker - SageMaker AI 서비스에 대한 스키마와 invoke_endpoint 함수 소스 (aws_sagemaker.invoke_endpoint)
- aws_bedrock - Amazon Bedrock 서비스에 대한 스키마와 invoke_model(aws_bedrock.invoke_model) 및 invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings) 함수 소스입니다.

rds_superuser 역할에는 aws_ml 관리자 역할이 부여되며 이 3개의 Aurora 기계 학습 스키마의 OWNER가 됩니다. 다른 데이터베이스 사용자가 Aurora 기계 학습 함수에 액세스할 수 있도록 하려면 rds_superuser가 Aurora 기계 학습 함수에 대한 EXECUTE 권한을 부여해야 합니다. 기본적으로 두 Aurora 기계 학습 스키마의 함수에 대한 EXECUTE 권한이 PUBLIC으로부터 취소됩니다.

멀티 테넌트 데이터베이스 구성에서는 보호하려는 특정 Aurora 기계 학습 스키마에서 REVOKE USAGE를 사용하여 테넌트가 Aurora 기계 학습 함수에 액세스하지 못하도록 할 수 있습니다.

Aurora PostgreSQL DB 클러스터와 함께 Amazon Bedrock 사용

Aurora PostgreSQL의 경우 Aurora 기계 학습은 텍스트 데이터를 사용하기 위한 다음 Amazon Bedrock 함수를 제공합니다. 이 함수는 aws_ml 2.0 확장을 설치하고 모든 설정 절차를 완료한 후에만 사용할 수 있습니다. 자세한 내용은 Aurora 기계 학습을 사용하도록 Aurora PostgreSQL DB 클러스터 설정 단원을 참조하십시오.

aws_bedrock.invoke_model

이 함수는 JSON 형식의 텍스트를 입력으로 받아 Amazon Bedrock에서 호스팅되는 다양한 모델에 맞게 처리하고 모델로부터 JSON 텍스트 응답을 가져옵니다. 이 응답에는 텍스트, 이미지 또는 임베딩이 포함될 수 있습니다. 함수 설명서의 요약은 다음과 같습니다.

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

이 함수의 입력과 출력은 다음과 같습니다.

- model_id – 모델 식별자입니다.

- `content_type` – Bedrock 모델에 대한 요청 유형입니다.
- `accept_type` – Bedrock 모델에서 기대할 수 있는 응답 유형입니다. 일반적으로 대부분의 모델에는 애플리케이션/JSON이 사용됩니다.
- `model_input` – 프롬프트로, content_type에 지정된 형식의 모델에 대한 특정 입력 집합입니다. 모델이 허용하는 요청 형식/구조에 대한 자세한 내용은 [파운데이션 모델의 추론 파라미터]를 참조하세요.
- `model_output` - Bedrock 모델의 텍스트 출력입니다.

다음 예제는 invoke_model을 사용하여 Bedrock에 대한 Anthropic Claude 2 모델을 간접 호출하는 방법을 보여줍니다.

Example 예: Amazon Bedrock 함수를 사용하는 간단한 쿼리

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

모델 출력은 경우에 따라 벡터 임베딩을 가리킬 수 있습니다. 모델마다 응답이 다르므로 invoke_model과 똑같이 작동하지만, 적절한 json-key를 지정하여 임베딩을 출력하는 invoke_model_get_embeddings라는 또 다른 함수를 활용할 수 있습니다.

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

이 함수의 입력과 출력은 다음과 같습니다.

- `model_id` – 모델 식별자입니다.
- `content_type` – Bedrock 모델에 대한 요청 유형입니다. 여기서는 accept_type이 기본값 (application/json)으로 설정되어 있습니다.
- `model_input` – 프롬프트로, content_type에 지정된 형식의 모델에 대한 특정 입력 집합입니다. 모델이 허용하는 요청 형식/구조에 대한 자세한 내용은 [파운데이션 모델의 추론 파라미터]를 참조하세요.
- `json_key` - 임베딩을 추출할 필드에 대한 참조입니다. 임베딩 모델이 변경되면 달라질 수 있습니다.
- `model_output` – Bedrock 모델의 출력은 16비트 십진수로 구성된 임베딩 배열입니다.

다음 예제는 PostgreSQL I/O 모니터링 뷰라는 문구에 대한 Titan Embeddings G1 – Text 임베딩 모델을 사용하여 임베딩을 생성하는 방법을 보여줍니다.

Example 예: Amazon Bedrock 함수를 사용하는 간단한 쿼리

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
   content_type  := 'application/json',
   json_key      := 'embedding',
   model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

Aurora PostgreSQL DB 클러스터와 함께 Amazon Comprehend 사용

Aurora PostgreSQL의 경우 Aurora 기계 학습은 텍스트 데이터를 사용하기 위한 다음 Amazon Comprehend 함수를 제공합니다. 이 함수는 `aws_ml` 확장을 설치하고 모든 설정 절차를 완료한 후에만 사용할 수 있습니다. 자세한 내용은 [Aurora 기계 학습을 사용하도록 Aurora PostgreSQL DB 클러스터 설정] 단원을 참조하십시오.

aws_comprehend.detect_sentiment

이 함수는 텍스트를 입력으로 받아 텍스트의 감정 태도가 긍정적인지, 부정적인지, 중립적인지 또는 혼합되어 있는지 평가합니다. 그런 다음 평가를 위한 신뢰도와 함께 이 감정을 출력합니다. 함수 설명서의 요약은 다음과 같습니다.

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
```

```
    OUT confidence real)
```

이 함수의 입력과 출력은 다음과 같습니다.

- input_text - 감정을 평가하고 할당할 텍스트(부정, 긍정, 중립, 혼합)
- language_code - 해당하는 경우 필요에 따라 리전 하위 태그가 있는 2자리 ISO 639-1 식별자 또는 ISO 639-2 세 글자 코드를 사용하여 식별된 input_text의 언어. 예를 들어, en은 영어 코드이고 zh는 중국어 간체 코드입니다. 자세한 내용은 Amazon Comprehend 개발자 안내서의 지원되는 언어를 참조하세요.
- max_rows_per_batch - 배치 모드 처리를 위한 배치당 최대 행 수 자세한 내용은 배치 모드 및 Aurora 기계 학습 함수의 이해 단원을 참조하십시오.
- sentiment - 긍정, 부정, 중립 또는 혼합으로 식별되는 입력 텍스트의 감정
- confidence - 지정된 sentiment의 정확성에 대한 신뢰도. 값의 범위는 0.0~1.0입니다.

아래에서 이 함수를 사용하는 방법의 예를 찾아볼 수 있습니다.

Example 예: Amazon Comprehend 함수를 사용하는 간단한 쿼리

다음은 이 함수를 호출하여 지원 팀에 대한 고객 만족도를 평가하는 간단한 쿼리의 예입니다. 도움 요청이 있을 때마다 고객 피드백을 저장하는 데이터베이스 테이블(support)이 있다고 가정해 봅시다. 이 예시 쿼리는 테이블 feedback 열의 텍스트에 aws_comprehend.detect_sentiment 함수를 적용하고 감정 및 감정에 대한 신뢰도를 출력합니다. 이 쿼리는 결과를 내림차순으로 출력합니다.

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                                  | sentiment | confidence
 ----------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!            | POSITIVE  |   0.999771
 The latest version of this product stinks!               | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.      | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.  | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.          | POSITIVE  |   0.949491
 My problem was never resolved!                           | NEGATIVE  |   0.920644
 When will the new version of this product be released?   | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                             | NEGATIVE  |   0.895219
 Your support tech talked down to me.                     | NEGATIVE  |   0.868598
 It took me way too long to get a real person.            | NEGATIVE  |   0.481805
```

```
(10 rows)
```

테이블 행당 두 번 이상 감성 분석 요금이 청구되지 않도록 결과를 구체화할 수 있습니다. 관심있는 행에 대해 이 작업을 수행하십시오. 예를 들어, 임상의의 메모를 업데이트하여 프랑스어(`fr`)로 된 텍스트만 감정 감지 함수를 사용할 수 있도록 합니다.

```
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

함수 호출 최적화에 대한 자세한 내용은 Aurora PostgreSQL을 사용하는 Aurora 기계 학습의 성능 고려 사항 단원을 참조하십시오.

Aurora PostgreSQL DB 클러스터와 함께 SageMaker AI 사용

Amazon SageMaker AI를 사용하도록 Aurora PostgreSQL 설정 에서 설명된 대로 SageMaker AI 환경을 설정하고 Aurora PostgreSQL과 통합한 후에 `aws_sagemaker.invoke_endpoint` 함수를 사용하여 작업을 간접 호출할 수 있습니다. `aws_sagemaker.invoke_endpoint` 함수는 동일한 AWS 리전의 모델 엔드포인트에만 연결됩니다. 데이터베이스 인스턴스가 여러 AWS 리전에 복제본을 가지고 있는 경우 각 SageMaker AI 모델을 AWS 리전마다 설정하고 배포해야 합니다.

`aws_sagemaker.invoke_endpoint`에 대한 직접 호출은 Aurora PostgreSQL DB 클러스터를 설정 프로세스 중에 제공한 SageMaker AI 서비스 및 엔드포인트와 연결하도록 설정한 IAM 역할을 사용하여 인증됩니다. SageMaker AI 모델 엔드포인트는 개별 계정으로 범위가 지정되며 공개되지 않습니다. `endpoint_name` URL에 계정 ID를 포함하지 않습니다. SageMaker AI는 데이터베이스 인스턴스의 SageMaker AI IAM 역할에서 제공하는 인증 토큰에서 계정 ID를 확인합니다.

aws_sagemaker.invoke_endpoint

이 함수는 SageMaker AI 엔드포인트를 입력으로 사용하고 일괄 처리해야 하는 행 수를 사용합니다. 또한 SageMaker AI 모델 엔드포인트에서 기대하는 다양한 파라미터를 입력으로 사용합니다. 이 함수의 참조 문서는 다음과 같습니다.

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
```

```
    IN max_rows_per_batch int,
    VARIADIC model_input "any",
    OUT model_output varchar
    )
```

이 함수의 입력과 출력은 다음과 같습니다.

- endpoint_name - AWS 리전에 독립적인 엔드포인트 URL
- max_rows_per_batch - 배치 모드 처리를 위한 배치당 최대 행 수 자세한 내용은 배치 모드 및 Aurora 기계 학습 함수의 이해 단원을 참조하십시오.
- model_input - 모델에 대한 하나 이상의 입력 파라미터. SageMaker AI 모델에 필요한 모든 데이터 형식이 될 수 있습니다. PostgreSQL을 사용하면 함수에 대해 최대 100개의 입력 파라미터를 지정할 수 있습니다. 배열 데이터 형식은 1차원이어야 하지만 SageMaker AI 모델에서 예상하는 만큼 많은 요소를 포함할 수 있습니다. SageMaker AI 모델에 대한 입력 수는 SageMaker AI 메시지 크기 제한인 6MB로만 제한됩니다.
- model_output - SageMaker AI 모델의 텍스트 출력

SageMaker AI 모델을 간접 호출하는 사용자 정의 함수 만들기

각 SageMaker AI 모델에 대해 aws_sagemaker.invoke_endpoint를 직접 호출하는 별도의 사용자 정의 함수를 만듭니다. 사용자 정의 함수는 이 모델을 호스팅하는 SageMaker AI 엔드포인트를 나타냅니다. 이 aws_sagemaker.invoke_endpoint 함수는 사용자 정의 함수 내에서 실행됩니다. 사용자 정의 함수는 다음과 같은 많은 장점을 제공합니다.

- 모든 SageMaker AI 모델에 대해 aws_sagemaker.invoke_endpoint를 직접 호출하는 대신 SageMaker AI 모델에 고유한 이름을 지정할 수 있습니다.
- SQL 애플리케이션 코드의 한 곳에만 모델 엔드포인트 URL을 지정할 수 있습니다.
- 각 Aurora 기계 학습 함수에 대한 EXECUTE 권한을 독립적으로 제어할 수 있습니다.
- SQL 유형을 사용하여 모델 입력 및 출력 유형을 선언할 수 있습니다. SQL은 SageMaker AI 모델에 전달된 인수의 수와 유형을 적용하고 필요한 경우 유형 변환을 수행합니다. SQL 유형을 사용하면 SQL NULL이 SageMaker AI 모델에서 예상되는 적절한 기본값으로 변환됩니다.
- 처음 몇 행을 조금 더 빨리 반환하려면 최대 배치 크기를 줄일 수 있습니다.

사용자 정의 함수를 지정하려면 SQL 데이터 정의 언어(DDL) 문 CREATE FUNCTION을 사용합니다. 함수를 정의할 때 다음을 지정합니다.

- 모델에 대한 입력 파라미터
- 간접 호출할 특정 SageMaker AI 엔드포인트
- 반환 유형

사용자 정의 함수에서는 입력 파라미터에 근거하여 모델을 실행한 후에 SageMaker AI 엔드포인트가 계산한 추론을 반환합니다. 다음 예시에서는 두 개의 입력 파라미터가 있는 SageMaker AI 모델에 대해 사용자 정의 함수를 만듭니다.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2                        -- model inputs are separate arguments
        )::INT                            -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

다음을 참조하십시오.

- `aws_sagemaker.invoke_endpoint` 함수 입력은 모든 데이터 형식에 대한 하나 이상의 파라미터가 될 수 있습니다.
- 이 예제에서는 INT 출력 유형을 사용합니다. `varchar` 유형에서 다른 유형으로 출력을 캐스팅하는 경우 `INTEGER`, `REAL`, `FLOAT` 또는 `NUMERIC` 등의 PostgreSQL 기본 제공 스칼라 유형으로 캐스팅해야 합니다. 이러한 유형에 대한 자세한 내용은 PostgreSQL 설명서의 [데이터 형식]을 참조하십시오.
- 병렬 쿼리 처리를 활성화하려면 `PARALLEL SAFE`를 지정합니다. 자세한 내용은 [병렬 쿼리 처리를 통한 응답 시간 개선] 섹션을 참조하세요.
- 함수 실행 비용을 예상하도록 `COST 5000`을 지정합니다. 함수의 예상 실행 비용을 `cpu_operator_cost` 단위로 제공하는 양수를 사용합니다.

SageMaker AI 모델에 대한 입력으로 배열 전달

`aws_sagemaker.invoke_endpoint` 함수는 PostgreSQL 함수의 한계인 최대 100개의 입력 파라미터를 가질 수 있습니다. SageMaker AI 모델에 동일한 유형의 파라미터가 100개 이상 필요한 경우 모델 파라미터를 배열로 전달하세요.

다음 예시에서는 배열을 SageMaker AI 회귀 모델에 대한 입력으로 전달하는 함수를 정의합니다. 출력은 `REAL` 값으로 변환됩니다.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

SageMaker AI 모델 간접 호출 시 배치 크기 지정

다음 예시에서는 배치 크기 기본값을 NULL로 설정하는 SageMaker AI 모델에 대한 사용자 정의 함수를 만듭니다. 이 함수를 사용하면 호출할 때 다른 배치 크기를 제공할 수도 있습니다.

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

다음을 참조하십시오.

- 선택적 `max_rows_per_batch` 파라미터를 사용하여 배치 모드 함수 호출에 대한 행 수를 제어합니다. NULL 값을 사용하는 경우 쿼리 최적화 프로그램이 자동으로 최대 배치 크기를 선택합니다. 자세한 내용은 [배치 모드 및 Aurora 기계 학습 함수의 이해](#) 단원을 참조하십시오.
- 기본적으로 파라미터의 값으로 NULL을 전달하면 SageMaker AI에 전달하기 전에 빈 문자열로 변환됩니다. 이 예제의 경우 입력에 다양한 유형이 있습니다.
- 텍스트가 아닌 입력 또는 빈 문자열이 아닌 다른 값으로 기본값을 설정해야 하는 텍스트 입력이 있는 경우 `COALESCE` 문을 사용합니다. `COALESCE`를 호출할 때 `aws_sagemaker.invoke_endpoint`를 사용하여 NULL을 원하는 NULL 대체 값으로 변환합니다. 이 예제에 있는 `amount` 파라미터의 경우 NULL 값은 0.0으로 변환됩니다.

다중 출력이 있는 SageMaker AI 모델 간접 호출

다음 예시에서는 다중 출력을 반환하는 SageMaker AI 모델에 대한 사용자 정의 함수를 만듭니다. 함수는 `aws_sagemaker.invoke_endpoint` 함수의 출력을 해당 데이터 형식으로 캐스팅해야 합니다. 예를 들어 (x,y) 쌍이나 사용자 정의 복합 유형에 대해 기본 제공 PostgreSQL 포인트 유형을 사용할 수 있습니다.

이 사용자 정의 함수는 출력에 대한 복합 유형을 사용하여 여러 출력을 반환하는 모델에서 값을 반환합니다.

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

복합 유형의 경우 모델 출력에 나타나는 순서와 동일한 순서로 필드를 사용하고 `aws_sagemaker.invoke_endpoint`의 출력을 복합 유형으로 캐스팅합니다. 호출자는 이름 또는 PostgreSQL ".*" 표기법으로 개별 필드를 추출할 수 있습니다.

SageMaker AI 모델 훈련을 위해 Amazon S3로 데이터 내보내기(고급)

자체 모델을 훈련시키는 대신 제공된 알고리즘과 예시를 사용하여 Aurora 기계 학습과 SageMaker AI에 익숙해지는 것이 좋습니다. 자세한 내용은 [Get Started with Amazon SageMaker AI](#)를 참조하세요.

SageMaker AI 모델을 훈련하려면 데이터를 Amazon S3 버킷으로 내보냅니다. Amazon S3 버킷은 모델이 배포되기 전에 SageMaker AI에서 모델을 훈련하는 데 사용됩니다. Aurora PostgreSQL DB 클러스터에서 데이터를 쿼리한 후 Amazon S3 버킷에 저장된 텍스트 파일에 직접 저장할 수 있습니다. 그런 다음 SageMaker AI에서 훈련을 위해 Amazon S3 버킷의 데이터를 사용합니다. SageMaker AI 모델 훈련에 대한 자세한 내용은 [Train a model with Amazon SageMaker AI](#)를 참조하세요.

> **Note**
>
> SageMaker AI 모델 훈련 또는 배치 점수 계산을 위해 Amazon S3 버킷을 만들 때는 Amazon S3 버킷 이름에 `sagemaker`를 포함하세요. 자세한 내용은 Amazon SageMaker AI Developer Guide의 Specify a Amazon S3 Bucket to Upload Training Datasets and Store Output Data를 참조하세요.

데이터 내보내기에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기 단원을 참조하십시오.

Aurora PostgreSQL을 사용하는 Aurora 기계 학습의 성능 고려 사항

Amazon Comprehend 및 SageMaker AI 서비스는 Aurora 기계 학습 함수에서 간접 호출할 때 대부분의 작업을 수행합니다. 즉, 필요에 따라 독립적으로 리소스를 확장할 수 있습니다. Aurora PostgreSQL DB 클러스터의 경우 함수 호출을 최대한 효율적으로 만들 수 있습니다. 다음은 Aurora PostgreSQL에서 Aurora 기계 학습을 사용할 때 참고해야 할 몇 가지 성능 고려 사항입니다.

주제

배치 모드 및 Aurora 기계 학습 함수의 이해

일반적으로 PostgreSQL은 한 번에 한 행씩 함수를 실행합니다. Aurora 기계 학습은 배치 모드 실행이라는 접근 방식을 사용하여 많은 행에 대한 외부 Aurora 기계 학습 서비스 호출을 배치로 결합하여 이러한 오버헤드를 줄일 수 있습니다. 배치 모드에서는 Aurora 기계 학습에서 입력 행 배치에 대한 응답을 수신한 다음 응답을 실행 중인 쿼리에 한 번에 한 행씩 다시 전달합니다. 이 최적화는 PostgreSQL 쿼리 최적화 프로그램을 제한하지 않고 Aurora 쿼리의 처리량을 향상시킵니다.

함수가 `SELECT` 목록, `WHERE` 절 또는 `HAVING` 절에서 참조되는 경우 Aurora는 자동으로 배치 모드를 사용합니다. 최상위 단순 `CASE` 표현식은 배치 모드 실행에 적합합니다. 또한 최상위 검색 `CASE` 표현식은 첫 번째 `WHEN` 절이 배치 모드 함수 호출이 있는 간단한 술어인 경우 배치 모드 실행에 적합합니다.

사용자 정의 함수는 `LANGUAGE SQL` 함수여야 하며 `PARALLEL SAFE` 및 `COST 5000`을 지정해야 합니다.

SELECT 문에서 FROM 절로 함수 마이그레이션

일반적으로 배치 모드 실행에 적합한 aws_ml 함수는 Aurora에 의해 자동으로 FROM 절로 마이그레이션됩니다.

적합한 배치 모드 함수를 FROM 절로 마이그레이션하는 작업은 쿼리별 수준에서 수동으로 검사할 수 있습니다. 이렇게 하려면 EXPLAIN 문(및 ANALYZE와 VERBOSE)을 사용하고 각 배치 모드 Function Scan에서 "배치 처리" 정보를 찾습니다. 쿼리를 실행하지 않고 EXPLAIN(VERBOSE와 함께)을 사용할 수도 있습니다. 그런 다음 함수에 대한 호출이 원래 문에 지정되지 않은 중첩 루프 조인 아래에 Function Scan으로 표시되는지 여부를 관찰합니다.

다음 예에서 계획의 중첩 루프 조인 연산자는 Aurora가 anomaly_score 함수를 마이그레이션했음을 보여줍니다. 이 함수를 SELECT 목록에서 배치 모드 실행에 적합한 FROM 절로 마이그레이션했습니다.

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                         QUERY PLAN
-------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
         Function Call: anomaly_score((r.description)::text)
```

배치 모드 실행을 비활성화하려면 apg_enable_function_migration 파라미터를 false로 설정합니다. 이렇게 하면 aws_ml 함수가 SELECT에서 FROM 절로 마이그레이션되는 것을 방지할 수 있습니다. 아래에서는 이 작업을 수행하는 방법을 보여줍니다.

```
SET apg_enable_function_migration = false;
```

apg_enable_function_migration 파라미터는 쿼리 계획 관리를 위해 Aurora PostgreSQL apg_plan_mgmt 확장에서 인식하는 Grand Unified Configuration(GUC) 파라미터입니다. 세션에서 함수 마이그레이션을 비활성화하려면 쿼리 계획 관리를 사용하여 결과 계획을 approved 계획으로 저장합니다. 런타임 시 쿼리 계획 관리는 해당 approved 설정으로 apg_enable_function_migration 계획을 적용합니다. 이 적용은 apg_enable_function_migration GUC 파라미터 설정에 관계없이 발생합니다. 자세한 내용은 Aurora PostgreSQL용 쿼리 실행 계획 관리 섹션을 참조하세요.

max_rows_per_batch 파라미터 사용

`aws_comprehend.detect_sentiment` 및 `aws_sagemaker.invoke_endpoint` 함수 모두에 `max_rows_per_batch` 파라미터가 있습니다. 이 파라미터는 Aurora 기계 학습 서비스에 전송할 수 있는 행 수를 지정합니다. 함수에 의해 처리되는 데이터 세트가 클수록 배치 크기가 커질 수 있습니다.

배치 모드 함수는 많은 수의 행에 Aurora 기계 학습 함수 호출 비용을 분산시키는 행 배치를 구축하여 효율성을 향상시킵니다. 그러나 `SELECT` 절로 인해 `LIMIT` 문이 일찍 끝나는 경우 쿼리에서 사용하는 것보다 많은 행에 배치를 구성할 수 있습니다. 이 방법을 사용하면 AWS 계정에 추가 요금이 부과될 수 있습니다. 배치 모드 실행의 이점을 이용하면서 너무 큰 배치를 작성하지 않으려면 함수 호출에서 `max_rows_per_batch` 파라미터에 대해 더 작은 값을 사용하십시오.

배치 모드 실행을 사용하는 쿼리의 `EXPLAIN`(`VERBOSE`, `ANALYZE`)을 수행하면 중첩 루프 조인 아래에 `FunctionScan` 연산자가 표시됩니다. `EXPLAIN`에서 보고한 루프 수는 `FunctionScan` 연산자에서 행을 가져온 횟수와 같습니다. 명령문에서 LIMIT 절을 사용하는 경우 가져오기 수가 일정합니다. 배치 크기를 최적화하려면 `max_rows_per_batch` 파라미터를 이 값으로 설정합니다. 그러나 배치 모드 함수가 `WHERE` 절 또는 `HAVING` 절의 술어에서 참조되면 가져오기 수를 미리 알 수 없습니다. 이 경우 루프를 지침으로 사용하고 `max_rows_per_batch`를 사용하여 성능을 최적화하는 설정을 찾습니다.

배치 모드 실행 확인

배치 모드에서 함수가 실행되었는지 확인하려면 `EXPLAIN ANALYZE`를 사용합니다. 배치 모드 실행이 사용된 경우 쿼리 계획은 "일괄 처리" 섹션에 정보를 포함합니다.

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                avg/min/max batch call time=146.273/146.273/146.273
```

이 예에서는 3,333개의 행을 포함하는 배치가 1개 있었는데 처리하는 데 146.273ms가 걸렸습니다. "배치 처리" 섹션에는 다음이 표시됩니다.

- 이 함수 스캔 작업과 관련된 배치 수
- 배치 크기 평균, 최소 및 최대값
- 배치 실행 시간 평균, 최소 및 최대값

일반적으로 최종 배치는 나머지 배치보다 작기 때문에 평균보다 훨씬 작은 최소 배치 크기가 됩니다.

처음 몇 개의 행을 더 빨리 반환하려면 `max_rows_per_batch` 파라미터를 더 작은 값으로 설정하십시오.

사용자 정의 함수에서 LIMIT를 사용할 때 ML 서비스에 대한 배치 모드 호출 수를 줄이려면 max_rows_per_batch 파라미터를 더 작은 값으로 설정하십시오.

병렬 쿼리 처리를 통한 응답 시간 개선

많은 수의 행에서 최대한 빠르게 결과를 얻으려면 병렬 쿼리 처리와 배치 모드 처리를 함께 사용할 수 있습니다. SELECT, CREATE TABLE AS SELECT 및 CREATE MATERIALIZED VIEW 문에 대해 병렬 쿼리 처리를 사용할 수 있습니다.

> **Note**
>
> PostgreSQL은 아직 데이터 조작 언어(DML) 문에 대한 병렬 쿼리를 지원하지 않습니다.

병렬 쿼리 처리는 데이터베이스 내 및 ML 서비스 내에서 모두 발생합니다. 데이터베이스의 인스턴스 클래스에 있는 코어 수는 쿼리 실행 중에 사용할 수 있는 병렬 처리 정도를 제한합니다. 데이터베이스 서버는 병렬 작업자 세트 간에 작업을 분할하는 병렬 쿼리 실행 계획을 구성할 수 있습니다. 그런 다음 각 작업자는 수만 개의 행(또는 각 서비스에서 허용하는 만큼)을 포함하는 배치 요청을 작성할 수 있습니다.

모든 병렬 작업자의 배치 요청은 SageMaker AI의 엔드포인트로 전송됩니다. 엔드포인트가 지원할 수 있는 병렬 처리 정도는 이를 지원하는 인스턴스의 수와 유형으로 제한됩니다. K 정도의 병렬 처리를 위해서는 코어가 K 이상인 데이터베이스 인스턴스 클래스가 필요합니다. 또한 충분한 고성능 인스턴스 클래스의 K 초기 인스턴스가 존재하도록 모델에 대해 SageMaker AI 엔드포인트를 구성해야 합니다.

병렬 쿼리 처리를 사용하려면 전달하려는 데이터가 포함된 테이블의 parallel_workers 스토리지 파라미터를 설정할 수 있습니다. parallel_workers를 aws_comprehend.detect_sentiment와 같은 배치 모드 함수로 설정합니다. 최적화 프로그램에서 병렬 쿼리 계획을 선택하는 경우 AWS 기계 학습 서비스를 배치 및 병렬로 호출할 수 있습니다.

aws_comprehend.detect_sentiment 함수와 함께 다음 파라미터를 사용하여 4방향 병렬 처리가 있는 계획을 얻을 수 있습니다. 다음 두 파라미터 중 하나를 변경할 경우 변경 사항을 적용하려면 데이터베이스 인스턴스를 다시 시작해야 합니다.

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
```

```
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

병렬 쿼리 제어에 대한 자세한 내용은 PostgreSQL 설명서의 Parallel plans(병렬 계획)를 참조하세요.

구체화된 보기 및 구체화된 열 사용

데이터베이스에서 SageMaker AI 또는 Amazon Comprehend와 같은 AWS 서비스를 간접 호출하면 해당 서비스의 요금 정책에 따라 계정에 요금이 청구됩니다. 계정에 대한 요금을 최소화하기 위해 AWS 서비스를 호출한 결과를 구체화된 열로 구체화하여 AWS 서비스가 입력 행당 두 번 이상 호출되지 않도록 할 수 있습니다. 원하는 경우 `materializedAt` 타임스탬프 열을 추가하여 열이 구체화된 시간을 기록할 수 있습니다.

일반적인 단일 행 `INSERT` 문의 지연 시간은 일반적으로 배치 모드 함수를 호출하는 지연 시간보다 훨씬 짧습니다. 따라서 애플리케이션이 수행하는 모든 단일 행 `INSERT`에 대해 배치 모드 함수를 호출하면 애플리케이션의 지연 시간 요구 사항을 충족하지 못할 수 있습니다. AWS 서비스를 구체화된 열로 호출한 결과를 구체화하려면 일반적으로 고성능 애플리케이션이 구체화된 열을 채워야 합니다. 이를 위해 동시에 많은 행 배치에서 작동하는 `UPDATE` 문을 주기적으로 발행합니다.

`UPDATE`는 실행 중인 애플리케이션에 영향을 줄 수 있는 행 수준 잠금을 사용합니다. 따라서 `SELECT ... FOR UPDATE SKIP LOCKED`를 사용하거나 `MATERIALIZED VIEW`를 사용해야 할 수도 있습니다.

실시간으로 많은 수의 행에서 작동하는 분석 쿼리에서는 배치 모드 구체화와 실시간 처리를 함께 사용할 수 있습니다. 이를 위해 이러한 쿼리는 구체화된 결과가 아직 없는 행에 대한 쿼리를 사용하여 미리 구체화된 결과 중 `UNION ALL` 하나를 어셈블합니다. 경우에 따라 여러 위치에서 이러한 `UNION ALL`이 필요하거나 타사 애플리케이션에서 쿼리가 생성됩니다. 이 경우 `VIEW`를 생성하여 `UNION ALL` 작업을 캡슐화하면 이 세부 정보가 나머지 SQL 애플리케이션에 노출되지 않도록 할 수 있습니다.

구체화된 보기를 사용하여 스냅샷에서 임의의 `SELECT` 문의 결과를 구체화할 수 있습니다. 또한 나중에 언제든지 구체화된 보기를 새로 고칠 때도 사용할 수 있습니다. 현재 PostgreSQL은 증분 새로 고침을 지원하지 않으므로 구체화된 보기를 새로 고칠 때마다 구체화된 보기가 완전히 다시 계산됩니다.

배타적 잠금을 수행하지 않고 구체화된 보기의 콘텐츠를 업데이트하는 CONCURRENTLY 옵션을 사용하여 구체화된 보기를 새로 고칠 수 있습니다. 이렇게 하면 SQL 애플리케이션이 구체화된 보기를 새로 고치는 동안 구체화된 보기에서 읽을 수 있습니다.

Aurora 기계 학습 모니터링

사용자 지정 DB 클러스터 파라미터 그룹의 `track_functions` 파라미터를 `all`로 설정하여 `aws_ml` 함수를 모니터링할 수 있습니다. 기본적으로 이 파라미터는 프로시저 언어 함수만 추적하도록 `pl`로 설정됩니다. 이를 `all`로 변경하면 `aws_ml` 함수도 추적됩니다. 자세한 내용은 PostgreSQL 설명서에서 [Run-time Statistics](런타임 통계)를 참조하세요.

Aurora 기계 학습 함수에서 직접 호출하는 SageMaker AI 작업의 성능을 모니터링하는 방법에 대한 자세한 내용은 Amazon SageMaker AI Developer Guide에서 [Monitor Amazon SageMaker AI]를 참조하세요.

`track_functions`를 `all`로 설정하면 `pg_stat_user_functions` 보기를 쿼리하여 Aurora 기계 학습 서비스를 호출하기 위해 정의하고 사용하는 함수에 대한 통계를 가져올 수 있습니다. 보기에는 각 함수에 대한 `calls`, `total_time` 및 `self_time`의 숫자가 표시됩니다.

`aws_sagemaker.invoke_endpoint` 및 `aws_comprehend.detect_sentiment` 함수의 통계를 보려면 다음 쿼리를 사용하여 스키마 이름을 기준으로 결과를 필터링할 수 있습니다.

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

통계를 지우려면 다음 작업을 수행하세요.

```
SELECT pg_stat_reset();
```

PostgreSQL `pg_proc` 시스템 카탈로그를 쿼리하여 `aws_sagemaker.invoke_endpoint` 함수를 호출하는 SQL 함수의 이름을 가져올 수 있습니다. 이 카탈로그에는 함수, 절차 등에 대한 정보가 저장됩니다. 자세한 내용은 PostgreSQL 설명서의 [pg_proc]를 참조하세요. 다음은 소스(`prosrc`)에 invoke_endpoint라는 텍스트가 포함된 함수(`proname`)의 이름을 가져오기 위해 테이블을 쿼리하는 예입니다.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

AWS SDK를 사용한 Aurora용 코드 예제

다음 코드 예제는 Aurora를 AWS 소프트웨어 개발 키트(SDK)와 함께 사용하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 교차 서비스 예시에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

교차 서비스 예시는 여러 AWS 서비스 전반에서 작동하는 샘플 애플리케이션입니다.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 AWS SDK와 함께 이 서비스 사용을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작하기

Hello Aurora

다음 코드 예제에서는 Aurora를 사용하여 시작하는 방법을 보여줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;
```

```
public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\tCluster: database: {cluster.DatabaseName}
 identifier: {cluster.DBClusterIdentifier}.");
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusters를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)
```

```
target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_aurora.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
```

```
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBClusters를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package main

import (
 "context"
 "fmt"
```

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

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBClusters를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBClusters를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DescribeDBClusters를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)

# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DescribeDBClusters를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
```

```
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeDBClusters를 참조하십시오.

코드 예시

- [Aurora 서버리스 작업 항목 트래커 만들기]

AWS SDK를 사용한 Aurora 작업

다음 코드 예제에서는 AWS SDK를 통해 개별 Amazon 작업을 수행하는 방법을 보여줍니다. 이들 발췌문은 Aurora API를 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. 각 예제에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 [Amazon Aurora API 참조]를 참조하세요.

예시

- AWS SDK 또는 CLI와 함께 CreateDBCluster 사용
- AWS SDK 또는 CLI와 함께 CreateDBClusterParameterGroup 사용
- AWS SDK 또는 CLI와 함께 CreateDBClusterSnapshot 사용
- AWS SDK 또는 CLI와 함께 CreateDBInstance 사용
- AWS SDK 또는 CLI와 함께 DeleteDBCluster 사용
- AWS SDK 또는 CLI와 함께 DeleteDBClusterParameterGroup 사용
- AWS SDK 또는 CLI와 함께 DeleteDBInstance 사용
- AWS SDK 또는 CLI와 함께 DescribeDBClusterParameterGroups 사용
- AWS SDK 또는 CLI와 함께 DescribeDBClusterParameters 사용
- AWS SDK 또는 CLI와 함께 DescribeDBClusterSnapshots 사용
- AWS SDK 또는 CLI와 함께 DescribeDBClusters 사용
- AWS SDK 또는 CLI와 함께 DescribeDBEngineVersions 사용
- AWS SDK 또는 CLI와 함께 DescribeDBInstances 사용
- AWS SDK 또는 CLI와 함께 DescribeOrderableDBInstanceOptions 사용
- AWS SDK 또는 CLI와 함께 ModifyDBClusterParameterGroup 사용

AWS SDK 또는 CLI와 함께 `CreateDBCluster` 사용

다음 코드 예제는 `CreateDBCluster`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
```

```
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 [CreateDBCluster](#)를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
```

```
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateDBCluster를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
```

```
    DBClusterIdentifier:          aws.String(clusterName),
    Engine:                       aws.String(dbEngine),
    DBClusterParameterGroupName: aws.String(parameterGroupName),
    DatabaseName:                 aws.String(dbName),
    EngineVersion:                aws.String(dbEngineVersion),
    MasterUserPassword:           aws.String(adminPassword),
    MasterUsername:               aws.String(adminName),
   })
   if err != nil {
    log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
    return nil, err
   } else {
    return output.DBCluster, err
   }
  }
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateDBCluster를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static String createDBCluster(RdsClient rdsClient, String
 dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
 CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
```

```
                    .build();

            CreateDbClusterResponse response =
 rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateDBCluster를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }
```

```
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateDBCluster를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
```

```
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return cluster
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateDBCluster를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
```

```
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing 
 Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
```

```
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
 ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }
```

```
            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
 Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
```

```
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
```

```
                true
            })
            .return_once(|cluster, name, class, _| {
                Ok(CreateDbInstanceOutput::builder()
                    .db_instance(
                        DbInstance::builder()
                            .db_cluster_identifier(cluster)
                            .db_instance_identifier(name)
                            .db_instance_class(class)
                            .build(),
                    )
                    .build())
            });

        mock_rds
            .expect_describe_db_clusters()
            .with(eq("RustSDKCodeExamplesDBCluster"))
            .return_once(|id| {
                Ok(DescribeDbClustersOutput::builder()

  .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                    .build())
            });

        mock_rds
            .expect_describe_db_instance()
            .with(eq("RustSDKCodeExamplesDBInstance"))
            .return_once(|name| {
                Ok(DescribeDbInstancesOutput::builder()
                    .db_instances(
                        DbInstance::builder()
                            .db_instance_identifier(name)
                            .db_instance_status("Available")
                            .build(),
                    )
                    .build())
            });

        mock_rds
            .expect_describe_db_cluster_endpoints()
            .with(eq("RustSDKCodeExamplesDBCluster"))
            .return_once(|_| {
                Ok(DescribeDbClusterEndpointsOutput::builder()
```

```
 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
```

```
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 CreateDBCluster를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **CreateDBClusterParameterGroup** 사용

다음 코드 예제는 `CreateDBClusterParameterGroup`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create a custom cluster parameter group.
```

```
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- API 세부 정보는 AWS SDK for .NET SDK for Rust API 참조의 CreateDBClusterParameterGroup를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
```

```
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
 created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- API 세부 정보는 AWS SDK for C++ SDK for Rust API 참조의 CreateDBClusterParameterGroup를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
```

```
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- API 세부 정보는 AWS SDK for Go SDK for Rust API 참조의 CreateDBClusterParameterGroup를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
 dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
 CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
 rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
 response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x SDK for Rust API 참조의 CreateDBClusterParameterGroup를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
```

```
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 [CreateDBClusterParameterGroup](#)를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateDBClusterParameterGroup를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS코드 예시 리포지토리]에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
```

```
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
 SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(
```

```
CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 CreateDBClusterParameterGroup을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 `CreateDBClusterSnapshot` 사용

다음 코드 예제는 `CreateDBClusterSnapshot`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
 CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
 snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateDBClusterSnapshot을 참조하십시오.

C++

SDK for C++

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateDBClusterSnapshot을 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateDBClusterSnapshot을 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void createDBClusterSnapshot(RdsClient rdsClient, String
 dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
 CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
 rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
 response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateDBClusterSnapshot을 참조하십시오.

Kotlin

SDK for Kotlin

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateDBClusterSnapshot을 참조하십시오.

Python

SDK for Python (Boto3)

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateDBClusterSnapshot을 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get a list of allowed engine versions.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
 family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
 Status == 'available'.
    // Get a list of instance classes available for the selected engine
 and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
 EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
 ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
```

```
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
```

```
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
 ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }
```

```
            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
```

```
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 [CreateDBClusterSnapshot](#)을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **CreateDBInstance** 사용

다음 코드 예제는 `CreateDBInstance`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
 DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
```

```
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
 or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateDBInstance를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
```

```
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
 "",
                                 client);
            return false;
        }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateDBInstance를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateDBInstance를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
 CreateDbInstanceRequest.builder()
```

```
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
 rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
 response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateDBInstance를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
```

```
            dbInstanceClass = instanceClassVal
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateDBInstance를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 CreateDBInstance를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get a list of allowed engine versions.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
 family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
 Status == 'available'.
    // Get a list of instance classes available for the selected engine
 and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
 EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
 ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
```

```
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
 Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
```

```
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
 ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
 Some("Available"));
```

```
            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
```

```
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
```

```
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
```

```
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
```

```
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
```

```
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
```

```
 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 CreateDBInstance를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 `DeleteDBCluster` 사용

다음 코드 예제는 `DeleteDBCluster`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
 dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
```

```
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBCluster를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                result = false;
            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteDBCluster를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteDBCluster를 참조하십시오.

Java

SDK for Java 2.x

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void deleteCluster(RdsClient rdsClient, String
 dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
 DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteDBCluster를 참조하십시오.

Kotlin

SDK for Kotlin

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteDBCluster를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
```

```
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteDBCluster를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
```

```
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
 self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
 self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
 {status}");
                        continue;
                    }
                    None => {
```

```
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
 rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
```

```
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteDBCluster를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 `DeleteDBClusterParameterGroup` 사용

다음 코드 예제는 `DeleteDBClusterParameterGroup`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
 groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
 _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBClusterParameterGroup을 참조하십시오.

C++

SDK for C++

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteDBClusterParameterGroup을 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteDBClusterParameterGroup을 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](https://github.com/awsdocs/aws-doc-sdk-examples)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
            throws InterruptedException {
        try {
            boolean isDataDel = false;
            boolean didFind;
            String instanceARN;

            // Make sure that the database has been deleted.
            while (!isDataDel) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }
```

```
            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 [DeleteDBClusterParameterGroup](#)을 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
```

```
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteDBClusterParameterGroup을 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteDBClusterParameterGroup을 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
```

```
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
 self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
 self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
 {status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
```

```
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
 rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
```

```
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
```

```
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
```

```
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteDBClusterParameterGroup을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DeleteDBInstance** 사용

다음 코드 예제는 `DeleteDBInstance`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBInstance를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteDBInstance를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteDBInstance를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void deleteDatabaseInstance(RdsClient rdsClient, String
 dbInstanceIdentifier) {
        try {
```

```
            DeleteDbInstanceRequest deleteDbInstanceRequest =
 DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
 rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
 response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteDBInstance를 참조하십시오.

Kotlin

SDK for Kotlin

> Note
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
```

```
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteDBInstance를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
```

```
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DeleteDBInstance를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
```

```
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
 self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
 self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
 {status}");
```

```
                    continue;
                }
                None => {
                    warn!("No status for DB instance");
                    break;
                }
            }
        }
    }

    // Delete the DB cluster. rds.DeleteDbCluster.
    let delete_db_cluster = self
        .rds
        .delete_db_cluster(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
 rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
```

```
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
 {status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
```

```
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
```

```
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
```

```
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
```

```
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DeleteDBInstance를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBClusterParameterGroups** 사용

다음 코드 예제는 DescribeDBClusterParameterGroups의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterParameterGroups를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBClusterParameterGroups를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBClusterParameterGroups를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
 String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
 DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
 rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
 group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 [DescribeDBClusterParameterGroups](#)를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeDBClusterParameterGroups를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return parameter_group
```

- API 세부 정보는 Python(Boto3)용 AWS SDK API 참조의 DescribeDBClusterParameterGroups를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBClusterParameters** 사용

다음 코드 예제는 DescribeDBClusterParameters의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();
```

```
        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
 _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterParameters를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);
```

```
//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
```

```
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBClusterParameters를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
```

```
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBClusterParameters를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
 dbCLusterGroupName, int flag) {
        try {
```

```
            DescribeDbClusterParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBClusterParameters를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
```

```
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 [DescribeDBClusterParameters](https://sdk.amazonaws.com/kotlin/api/latest/index.html)를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](https://github.com/awsdocs/aws-doc-sdk-examples)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
```

```
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
 filtered
                            to contain only parameters that start with this
 prefix.
        :param source: When specified, only parameters from this source are
 retrieved.
                       For example, a source of 'user' retrieves only parameters
 that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
 self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return parameters
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 [DescribeDBClusterParameters](#)를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS코드 예시 리포지토리](#)에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get the parameter group. rds.DescribeDbClusterParameterGroups
    // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
    // Parse the ParameterName, Description, and AllowedValues values and display
them.
    pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
```

```
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
 FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
 SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
```

```
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- API 세부 정보는 AWS SDK for Rust SDK API 참조의 DescribeDBClusterParameters를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBClusterSnapshots** 사용

다음 코드 예제는 DescribeDBClusterSnapshots의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
 DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
 DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
```

```
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterSnapshots를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보십시오.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
```

```
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBClusterSnapshots를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
```

```
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 [DescribeDBClusterSnapshots](#)를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void waitForSnapshotReady(RdsClient rdsClient, String
 dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
 DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
 rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
 response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
```

```
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBClusterSnapshots를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
```

```
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeDBClusterSnapshots를 참조하십시오.

Python

SDK for Python (Boto3)

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- API 세부 정보는 Python(Boto3)용 AWS SDK API 참조의 DescribeDBClusterSnapshots를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 [AWS SDK와 함께 이 서비스 사용](#) 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBClusters** 사용

다음 코드 예제는 `DescribeDBClusters`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
```

```
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusters를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
```

```
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBClusters를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBClusters를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void describeDbClusterParameters(RdsClient rdsClient, String
 dbCLusterGroupName, int flag) {
        try {
            DescribeDbClusterParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
```

```
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBClusters를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeDBClusters를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_cluster(self, cluster_name):
        """
```

```
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
 %s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DescribeDBClusters를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
 Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
 ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
 Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
```

```
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
```

```
            .expect_describe_db_cluster_endpoints()
            .return_once(|_| {
                Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                    .build())
            });

        let mut scenario = AuroraScenario::new(mock_rds);
        scenario.engine_version = Some("aurora-mysql8.0".into());
        scenario.instance_class = Some("m5.large".into());
        scenario.username = Some("test username".into());
        scenario.password = Some(SecretString::new("test password".into()));

        tokio::time::pause();
        let assertions = tokio::spawn(async move {
            let create = scenario.start_cluster_and_instance().await;
            assert!(create.is_ok());
        });

        tokio::time::advance(Duration::from_secs(1)).await;
        tokio::time::advance(Duration::from_secs(1)).await;
        tokio::time::resume();
        let _ = assertions.await;
    }
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeDBClusters를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBEngineVersions** 사용

다음 코드 예제는 `DescribeDBEngineVersions`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBEngineVersions을 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBEngineVersions을 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
```

```
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBEngineVersions을 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
 DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
```

```
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBEngineVersions을 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
```

```
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeDBEngineVersions을 참조하십시오.

Python

SDK for Python (Boto3)

> Note
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
```

```
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
 list of
                                       engine versions to those that are
 compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DescribeDBEngineVersions을 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS코드 예시 리포지토리]에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
        let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
```

```
                    _ => None,
                },
            )
            .for_each(|(family, version)|
 versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
```

```
                         .db_parameter_group_family("f2")
                         .engine_version("f2a")
                         .build(),
                 )
                 .db_engine_versions(DbEngineVersion::builder().build())
                 .build())
         });

     let scenario = AuroraScenario::new(mock_rds);

     let versions_map = scenario.get_engines().await;

     assert_eq!(
         versions_map,
         Ok(HashMap::from([
             ("f1".into(), vec!["f1a".into(), "f1b".into()]),
             ("f2".into(), vec!["f2a".into()])
         ]))
     );
 }

 #[tokio::test]
 async fn test_scenario_get_engines_failed() {
     let mut mock_rds = MockRdsImpl::default();

     mock_rds
         .expect_describe_db_engine_versions()
         .with(eq("aurora-mysql"))
         .return_once(|_| {
             Err(SdkError::service_error(
                 DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                     ErrorKind::Other,
                     "describe_db_engine_versions error",
                 ))),
                 Response::new(StatusCode::try_from(400).unwrap(),
  SdkBody::empty()),
             ))
         });

     let scenario = AuroraScenario::new(mock_rds);

     let versions_map = scenario.get_engines().await;
     assert_matches!(
         versions_map,
```

```
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeDBEngineVersions를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeDBInstances** 사용

다음 코드 예제는 `DescribeDBInstances`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
 dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
```

```
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBInstances를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
```

```
                                              Aws::RDS::Model::DBInstance
 &instanceResult,
                                              const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeDBInstances를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DescribeDBInstances를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
 dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
 DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
 rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeDBInstances를 참조하십시오.

Kotlin

SDK for Kotlin

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeDBInstances를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](https://github.com/awsdocs/aws-doc-sdk-examples)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 DescribeDBInstances를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
```

```
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
 self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
 self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
 {status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
```

```
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
 rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
```

```
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
```

```
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
```

```
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Cluster
    tokio::time::resume();
```

```
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));
```

```
    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
```

```
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeDBInstances를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **DescribeOrderableDBInstanceOptions** 사용

다음 코드 예제는 `DescribeOrderableDBInstanceOptions`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
 DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
 engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
 _amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
 paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeOrderableDBInstanceOptions를 참조하십시오.

C++

SDK for C++

Note

GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
```

```
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeOrderableDBInstanceOptions를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
```

```
    return instances, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 [DescribeOrderableDBInstanceOptions]를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리]에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
                System.out.println("The version number of the database engine " +
engineOb.engineVersion());
            }
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeOrderableDBInstanceOptions를 참조하십시오.

PowerShell

PowerShell용 도구

예 1: 이 예는 AWS 리전에서 특정 DB 인스턴스 클래스를 지원하는 DB 엔진 버전을 나열합니다.

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

예 2: 이 예는 AWS 리전에서 특정 DB 엔진 버전을 지원하는 DB 인스턴스 클래스를 나열합니다.

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조의 DescribeOrderableDBInstanceOptions를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
        :return: The list of DB instance options that can be used to create a
 compatible DB instance.
        """
        try:
```

```
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DescribeOrderableDBInstanceOptions를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
 ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
```

```
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
 o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
 classes", &err))
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()
```

```
 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
```

```
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeOrderableDBInstanceOptions를 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 함께 **ModifyDBClusterParameterGroup** 사용

다음 코드 예제는 `ModifyDBClusterParameterGroup`의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- DB 클러스터 시작하기

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
```

```
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 [ModifyDBClusterParameterGroup](#)을 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);
```

```
        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
 modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ModifyDBClusterParameterGroup을 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
```

```
        log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
        return err
    } else {
        return nil
    }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 [ModifyDBClusterParameterGroup](https://)을 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
 String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
 DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
 rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
 group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 [ModifyDBClusterParameterGroup](#)을 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
```

```
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ModifyDBClusterParameterGroup를 참조하십시오.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
```

```
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API 세부 정보는 AWSSDK for Python (Boto3) API 참조의 ModifyDBClusterParameterGroup를 참조하십시오.

Rust

SDK for Rust

Note

GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
```

```
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
 SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
```

```
            Err(SdkError::service_error(
 
 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- API 세부 정보는 AWS SDK for RUST API 참조의 ModifyDBClusterParameterGroup을 참조하세요.

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하는 Aurora 시나리오

다음 코드 예제는 AWS SDK를 사용하여 Aurora에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. 이러한 시나리오에서는 Aurora 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여줍니다. 각 시나리오에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

예시

- AWS SDK를 사용하여 Aurora DB 클러스터 시작하기

AWS SDK를 사용하여 Aurora DB 클러스터 시작하기

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 사용자 지정 Aurora DB 클러스터 파라미터 그룹을 만들고 파라미터 값을 설정합니다.

- 파라미터 그룹을 사용하는 DB 클러스터를 생성합니다.
- 데이터베이스가 포함된 DB 인스턴스를 생성합니다.
- DB 클러스터의 스냅샷을 만든 다음, 리소스를 정리합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
```

```
    2.  Select an engine family and create a custom DB cluster parameter group
using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
method.
    4.  Get some parameters in the group using the
DescribeDBClusterParametersAsync method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
method.
    11. Display and select from a list of instance classes available for the
selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
    13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
    14. Display the connection endpoint string for the new DB cluster.
    15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
    16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
    17. Delete the DB instance using the DeleteDBInstanceAsync method.
    18. Delete the DB cluster using the DeleteDBClusterAsync method.
    19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
    20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
    */

    private static readonly string sepBar = new('-', 80);
    private static AuroraWrapper auroraWrapper = null!;
    private static ILogger logger = null!;
    private static readonly string engine = "aurora-mysql";
    static async Task Main(string[] args)
    {
```

```
        // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
                    .AddTransient<AuroraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger<AuroraScenario>();

        auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });
```

```
            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)

        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
```

```
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
 options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
 auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
 group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
 by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
```

```
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
 new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);
```

```
        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
```

```
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
 group.");

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
 "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
 choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
 dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
 dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
 family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
```

```
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);
```

```
        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
  clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
  auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
```

```
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;


        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
```

```
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {

            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
```

```
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
 auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await
 auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
```

```
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
 auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
 {snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
 {newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
```

```
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }
```

```
    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Aurora 작업을 관리하기 위해 시나리오가 직접적으로 호출하는 래퍼 메서드.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
```

```
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
```

```
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
 _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
 _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
```

```
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
 DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
 engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
 _amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
 paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
 groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
 _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
 dbInstanceIdentifier = null)
```

```
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
```

```
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
```

```
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
 DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
 DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
 dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
```

```
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters

- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();
```

```
    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
 std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }
```

```
        std::cout << "Getting available parameter group families for " <<
DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

```
    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
              << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB cluster parameter group.
    if (!getDBCLusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                                NO_SOURCE,
                                autoIncrementParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value between ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
```

```
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBCLusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
```

```
                << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
    else {
        std::cout << "Let's create a DB cluster." << std::endl;
        const Aws::String administratorName = askQuestion(
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of engine versions for the parameter group family.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group family are:"
                  << std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
```

```
            std::cout << "  " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                        1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
```

```
            return false;
        }
    }

    std::cout << "Waiting for the DB cluster to become available." << std::endl;

    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
```

```
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
 "",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;

        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
 "",
                             client);
            return false;
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }
```

```
    // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
    displayConnection(dbCluster);

    printAsterisksLine();

    if (askYesNoQuestion(
            "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
        Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                               Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for the snapshot to become available." <<
std::endl;

        Aws::RDS::Model::DBClusterSnapshot snapshot;
        counter = 0;
        do {
```

```
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }
```

```
    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
```

```
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
```

```
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
```

```
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
```

```
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
```

```
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
 parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

```
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
 parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
 counter
                      << " seconds." << std::endl;
            return false;
        }
```

```
        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
 std::endl;
            }

            if (clusterDeleting) {
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
```

```
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
```

```
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(dbEngine string, parameterGroupName
 string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(dbEngine, parameterGroupName)
 scenario.SetUserParameters(parameterGroupName)
 cluster := scenario.CreateCluster(clusterName, dbEngine, dbName, parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(clusterName)
 scenario.Cleanup(dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(parameterGroupName)
 if err != nil {
```

```
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
```

```
func (scenario GetStartedClusters) SetUserParameters(parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
 userParameters, err := scenario.dbClusters.GetParameters(parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
```

```
// parameter group.
func (scenario GetStartedClusters) CreateCluster(clusterName string, dbEngine
 string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(clusterName)
   if err != nil {
```

```
   panic(err)
  }
  log.Println("Cluster created and available.")
 }
}
log.Println("Cluster data:")
log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
log.Printf("\tStatus: %v\n", *cluster.Status)
log.Printf("\tEngine: %v\n", *cluster.Engine)
log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
log.Println(strings.Repeat("-", 88))
return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(cluster *types.DBCluster)
 *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
```

```
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err =
 scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
   if err != nil {
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
 and tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
   "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
```

```
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(dbInstance *types.DBInstance, cluster
 *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {
```

```
 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(*dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(*cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err =
 scenario.dbClusters.GetInstance(*dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(*cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err =
 scenario.dbClusters.DeleteParameterGroup(*parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

Aurora 작업을 관리하기 위해 시나리오가 호출하는 함수를 정의합니다.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
```

```
    DBParameterGroupFamily:      aws.String(parameterGroupFamily),
    Description:                 aws.String(description),
   })
  if err != nil {
   log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
   return nil, err
  } else {
   return output.DBClusterParameterGroup, err
  }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
  _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
   &rds.DeleteDBClusterParameterGroupInput{
    DBClusterParameterGroupName: aws.String(parameterGroupName),
   })
  if err != nil {
   log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
   return err
  } else {
   return nil
  }
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

  var output *rds.DescribeDBClusterParametersOutput
  var params []types.Parameter
  var err error
  parameterPaginator :=
  rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
   &rds.DescribeDBClusterParametersInput{
    DBClusterParameterGroupName: aws.String(parameterGroupName),
    Source:                      aws.String(source),
```

```
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
```

```
    err = nil
   } else {
    log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
   }
   return nil, err
  } else {
   return &output.DBClusters[0], err
  }
 }



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
```

```
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
```

```
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
```

```
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
```

```
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster

- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
```

```
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = "\n" +
                "Usage:\n" +
                "    <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
 <dbSnapshotIdentifier><secretName>"
                +
                "Where:\n" +
                "    dbClusterGroupName - The name of the DB cluster parameter
 group. \n" +
                "    dbParameterGroupFamily - The DB cluster parameter group
 family name (for example, aurora-mysql5.7). \n"
                +
```

```
                "    dbInstanceClusterIdentifier - The instance cluster 
identifier value.\n" +
                "    dbInstanceIdentifier - The database instance identifier.\n" 
+
                "    dbName - The database name.\n" +
                "    dbSnapshotIdentifier - The snapshot identifier.\n" +
                "    secretName - The name of the AWS Secrets Manager secret that 
contains the database credentials\"\n";
        ;

        if (args.length != 7) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbClusterGroupName = args[0];
        String dbParameterGroupFamily = args[1];
        String dbInstanceClusterIdentifier = args[2];
        String dbInstanceIdentifier = args[3];
        String dbName = args[4];
        String dbSnapshotIdentifier = args[5];
        String secretName = args[6];

        // Retrieve the database credentials using AWS Secrets Manager.
        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)), 
User.class);
        String username = user.getUsername();
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the 
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient, 
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier, 
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier, 
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }

    public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
            throws InterruptedException {
        try {
            boolean isDataDel = false;
            boolean didFind;
            String instanceARN;

            // Make sure that the database has been deleted.
            while (!isDataDel) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
```

```
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
 DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


 rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
 dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
 DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
 dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
 DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
 rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
 response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitForSnapshotReady(RdsClient rdsClient, String
 dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
 DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
 rdsClient.describeDBClusterSnapshots(snapshotsRequest);
```

```
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
```

```
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest = 
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response = 
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection 
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest = 
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
```

```
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
```

```
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
```

```
                .dbClusterParameterGroupName(dbParameterGroupFamily)
                .engine("aurora-mysql")
                .masterUsername(userName)
                .masterUserPassword(password)
                .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                .dbParameterGroupFamily(dbParameterGroupFamily)
                .engine("aurora-mysql")
                .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
```

```
    public static void modifyDBClusterParas(RdsClient rdsClient, String
 dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
            ModifyDbClusterParameterGroupRequest groupRequest =
 ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
 rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
 response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameters(RdsClient rdsClient, String
 dbCLusterGroupName, int flag) {
        try {
            DescribeDbClusterParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
                dbParameterGroupsRequest =
 DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest =
 DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
```

```
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
```

```
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();
```

```
            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances

- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
```

```
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
 User::class.java)
    val username = user.username
    val userPassword = user.password
```

```
    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()

    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
 dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
 engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
 dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)
```

```
    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)

    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
```

```
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
```

```
    }
}

suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
```

```
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
```

```
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "aurora-mysql"
            maxRecords = 20
        }
    var instanceClass = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbClustersRequest {
            dbClusterIdentifier = dbClusterIdentifierVal
        }
```

```
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
```

```
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
```

```
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
```

```
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}

suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            engine = "aurora-mysql"
            defaultOnly = true
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
```

```
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper

    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
engine and
        create a DB cluster parameter group that is compatible with a selected
engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
parameter group.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group
```

```
    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
```

```
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [
                ver["EngineVersionDescription"] for ver in engine_versions
            ]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
```

```
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"]
                    for opt in inst_opts
                }
            )
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
```

```
                f"Creating a database instance. This typically takes several
 minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name,
                cluster_name,
                cluster["Engine"],
                inst_opts[inst_index]["DBInstanceClass"],
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
 how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
```

```
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
 few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
 group.
        Before the DB cluster parameter group can be deleted, all associated DB
 instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
```

```
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
parameter_group)
        wait(5)
```

```
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Aurora 작업을 관리하기 위해 시나리오가 호출하는 함수를 정의합니다.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
 parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
        :param parameter_group_family: The family that is used as the basis of
 the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
```

```
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
```

```
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
```

```
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why: 
 %s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return cluster


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
```

```
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
 that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
```

```
        :param cluster_id: The ID of the DB cluster where the DB instance is
 created.
        :param db_engine: The database engine of a database to create in the DB
 instance.
                          This must be compatible with the configured parameter
 group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
 instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
 list of
                                       engine versions to those that are
 compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
```

```
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
        :return: The list of DB instance options that can be used to create a
 compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
```

```
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS코드 예시 리포지토리에서 전체 예시를 찾고 설정 및 실행하는 방법을 배워보세요.

Aurora 시나리오의 시나리오별 함수가 들어 있는 라이브러리입니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
```

```
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
```

```
            context: Some(MetadataError::from(err)),
        }
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}
```

```
pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // snippet-start:[rust.aurora.get_engines.usage]
    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
 ScenarioError> {
        let describe_db_engine_versions =
 self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };
```

```
        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }
    // snippet-end:[rust.aurora.get_engines.usage]

    // snippet-start:[rust.aurora.get_instance_classes.usage]
    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
```

```
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }
    // snippet-end:[rust.aurora.get_instance_classes.usage]

    // snippet-start:[rust.aurora.set_engine.usage]
    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
```

```
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.set_engine.usage]

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
 Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    // snippet-start:[rust.aurora.get_cluster.usage]
    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }
```

```
        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }
    // snippet-end:[rust.aurora.get_cluster.usage]

    // snippet-start:[rust.aurora.cluster_parameters.usage]
    // Get the parameter group. rds.DescribeDbClusterParameterGroups
    // Get parameters in the group. This is a long list so you will have to
 paginate. Find the auto_increment_offset and auto_increment_increment parameters
 (by ParameterName). rds.DescribeDbClusterParameters
    // Parse the ParameterName, Description, and AllowedValues values and display
 them.
    pub async fn cluster_parameters(&self) ->
 Result<Vec<AuroraScenarioParameter>, ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
 {DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
 FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }
    // snippet-end:[rust.aurora.cluster_parameters.usage]
```

```
    // snippet-start:[rust.aurora.update_auto_increment.usage]
    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.update_auto_increment.usage]

    // snippet-start:[rust.aurora.start_cluster_and_instance.usage]
    // Get a list of allowed engine versions.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
 family used to create your parameter group in step 2>)
```

```
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
```

```
            return Err(ScenarioError::with("Created DB Cluster missing
 Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
```

```
            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }
```

```
            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }
    // snippet-end:[rust.aurora.start_cluster_and_instance.usage]

    // snippet-start:[rust.aurora.snapshot.usage]
    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
    pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
        let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
        let snapshot = self
            .rds
            .snapshot_cluster(id, format!("{id}_{name}").as_str())
            .await;
        match snapshot {
            Ok(output) => match output.db_cluster_snapshot {
                Some(snapshot) => Ok(snapshot),
                None => Err(ScenarioError::with("Missing Snapshot")),
            },
            Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
        }
    }
    // snippet-end:[rust.aurora.snapshot.usage]

    // snippet-start:[rust.aurora.clean_up.usage]
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
```

```
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
 self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
 self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
```

```
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
```

```
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
 {status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
```

```
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
    // snippet-end:[rust.aurora.clean_up.usage]
}

#[cfg(test)]
pub mod tests;
```

RDS Client 래퍼 주변의 오토모크를 사용하여 라이브러리를 테스트합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use crate::rds::MockRdsImpl;

use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
```

```
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

// snippet-start:[rust.aurora.set_engine.test]
#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
```

```
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
// snippet-end:[rust.aurora.set_engine.test]

// snippet-start:[rust.aurora.get_engines.test]
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
```

```
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

```
// snippet-end:[rust.aurora.get_engines.test]

// snippet-start:[rust.aurora.get_instance_classes.test]
#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
```

```
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(
 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
// snippet-end:[rust.aurora.get_instance_classes.test]

// snippet-start:[rust.aurora.get_cluster.test]
```

```
#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
```

```
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}
// snippet-end:[rust.aurora.get_cluster.test]

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

// snippet-start:[rust.aurora.cluster_parameters.test]
#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
```

```
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
// snippet-end:[rust.aurora.cluster_parameters.test]

// snippet-start:[rust.aurora.update_auto_increment.test]
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(
```

```
ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
// snippet-end:[rust.aurora.update_auto_increment.test]

// snippet-start:[rust.aurora.start_cluster_and_instance.test]
#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
```

```
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
```

```
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
```

```
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
  SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

  .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

  .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.start_cluster_and_instance.test]

// snippet-start:[rust.aurora.clean_up.test]
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.clean_up.test]

// snippet-start:[rust.aurora.snapshot.test]
#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
```

```
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
// snippet-end:[rust.aurora.snapshot.test]
```

시나리오를 처음부터 끝까지 실행하는 바이너리입니다. 사용자가 일부 결정을 내릴 수 있도록 인콰이어러를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }
```

```
    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
```

```
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
```

```
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings,
 "auto_increment_increment", 3);

    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
 increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get
 just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
 'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(
            "Failed to get username, using default",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
    }
    let username = username.unwrap();

    let password = inquire::Text::new("Password for the database (minimum 8
 characters)")
        .with_validator(|i: &str| {
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
```

```
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
```

```
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
```

```
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
```

```
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

테스트를 위한 오토모킹(automocking)을 허용하는 Amazon RDS 서비스를 둘러싼 래퍼입니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
```

```
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }

    // snippet-start:[rust.aurora.describe_db_engine_versions.wrapper]
    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
```

```
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_engine_versions.wrapper]

    // snippet-start:[rust.aurora.describe_orderable_db_instance_options.wrapper]
    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_orderable_db_instance_options.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_parameter_group.wrapper]
    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
```

```
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.describe_db_clusters.wrapper]
    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_clusters.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_parameters.wrapper]
    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
 SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_parameters.wrapper]

    // snippet-start:[rust.aurora.modify_db_cluster_parameter_group.wrapper]
    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
 SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
```

```
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
    // snippet-end:[rust.aurora.modify_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster.wrapper]
    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster.wrapper]

    // snippet-start:[rust.aurora.create_db_instance.wrapper]
    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
```

```
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_instance.wrapper]

    // snippet-start:[rust.aurora.describe_db_instance.wrapper]
    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_instance.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_snapshot.wrapper]
    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
 SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_snapshot.wrapper]

    // snippet-start:[rust.aurora.describe_db_instances.wrapper]
    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }
    // snippet-end:[rust.aurora.describe_db_instances.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_endpoints.wrapper]
```

```
    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
 SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_endpoints.wrapper]

    // snippet-start:[rust.aurora.delete_db_instance.wrapper]
    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_instance.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster.wrapper]
    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
    pub async fn delete_db_cluster_parameter_group(
        &self,
```

```
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
}
```

이 시나리오에 사용된 종속 항목이 있는 Cargo.toml입니다.

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용한 Aurora용 교차 서비스 예제

다음 샘플 애플리케이션에서는 AWS SDK를 사용하여 Aurora를 다른 AWS 서비스와 결합합니다. 각 예시에는 애플리케이션을 설정하고 실행하는 방법에 대한 지침을 찾을 수 있는 GitHub 링크가 포함되어 있습니다.

예시

- 대출 라이브러리 REST API 생성
- Aurora 서버리스 작업 항목 트래커 만들기

대출 라이브러리 REST API 생성

다음 코드 예제에서는 Amazon Aurora 데이터베이스가 지원하는 REST API를 사용하여 고객이 도서를 빌리고 반납할 수 있는 대출 라이브러리를 생성하는 방법을 보여줍니다.

Python

SDK for Python(Boto3)

Amazon Relational Database Service(Amazon RDS) API 및 AWS Chalice와 함께 AWS SDK for Python (Boto3)를 사용하여 Amazon Aurora 데이터베이스에서 지원하는 REST API를 생성하는 방법을 보여줍니다. 웹 서비스는 완전히 서버리스이며 고객이 책을 빌리고 반납할 수 있는 간단한 대출 라이브러리를 나타냅니다. 다음 작업을 수행하는 방법에 대해 알아보십시오.

- 서버리스 Aurora 데이터베이스 클러스터를 생성하고 관리합니다.
- AWS Secrets Manager를 사용하여 데이터베이스 자격 증명을 관리합니다.
- Amazon RDS를 사용하여 데이터를 데이터베이스 내부 및 외부로 이동하는 데이터 스토리지 계층을 구현합니다.
- AWS Chalice를 사용하여 서버리스 REST API를 Amazon API Gateway 및 AWS Lambda에 배포합니다.
- 요청 패키지를 사용하여 웹 서비스에 요청을 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예제에서 사용되는 서비스

- API Gateway
- Aurora
- Lambda
- Secrets Manager

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Aurora 서비스 작업 항목 트래커 만들기

다음 코드 예제는 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 사용하여 보고서를 보내는 웹 애플리케이션 생성 방법을 보여줍니다.

.NET

SDK for .NET

AWS SDK for .NET을 사용하여 Amazon Aurora 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 통해 보고서를 이메일로 보내는 웹 애플리케이션 생성 방법을 보여줍니다. 이 예제에서는 RESTful .NET 백엔드와의 상호 작용을 위해 React.js로 빌드된 프런트엔드를 사용합니다.

- React 웹 애플리케이션을 AWS 서비스와 통합합니다.
- Aurora 테이블의 항목을 나열, 추가, 업데이트 및 삭제합니다.
- Amazon SES를 사용하여 필터링된 작업 항목에 대한 이메일 보고서를 보냅니다.
- 포함된 AWS CloudFormation 스크립트를 통해 예제 리소스를 배포하고 관리합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

C++

SDK for C++

Amazon Aurora Serverless 데이터베이스에 저장된 작업 항목을 추적하고 보고하는 웹 애플리케이션 생성 방법을 보여줍니다.

Amazon Aurora Serverless 데이터를 쿼리하고 React 애플리케이션에서 사용하도록 C++ REST API를 설정하는 방법에 대한 지침과 전체 소스 코드는 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스

- Amazon SES

Java

SDK for Java 2.x

Amazon RDS 데이터베이스에 저장된 작업 항목을 추적하고 보고하는 웹 애플리케이션 생성 방법을 보여줍니다.

Amazon Aurora Serverless 데이터를 쿼리하고 React 애플리케이션에서 사용하도록 Spring REST API를 설정하는 방법에 대한 지침과 전체 소스 코드는 GitHub에서 전체 예제를 참조하십시오.

JDBC API를 사용한 예제를 설정하고 실행하는 방법에 대한 전체 소스 코드와 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript(v3)을 사용하여 Amazon Aurora 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 통해 보고서를 이메일로 보내는 웹 애플리케이션 생성 방법을 보여줍니다. 이 예제에서는 Express Node.js 백엔드와의 상호 작용을 위해 React.js로 빌드된 프런트엔드를 사용합니다.

- React.js 웹 애플리케이션을 AWS 서비스와 통합합니다.
- Aurora 테이블의 항목을 나열, 추가 및 업데이트합니다.
- Amazon SES를 사용하여 필터링된 작업 항목에 대한 이메일 보고서를 보냅니다.
- 포함된 AWS CloudFormation 스크립트를 통해 예제 리소스를 배포하고 관리합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

Kotlin

SDK for Kotlin

Amazon RDS 데이터베이스에 저장된 작업 항목을 추적하고 보고하는 웹 애플리케이션 생성 방법을 보여줍니다.

Amazon Aurora Serverless 데이터를 쿼리하고 React 애플리케이션에서 사용하도록 Spring REST API를 설정하는 방법에 대한 지침과 전체 소스 코드는 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

PHP

SDK for PHP

AWS SDK for PHP을 사용하여 Amazon RDS 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 통해 보고서를 이메일로 보내는 웹 애플리케이션 생성 방법을 보여줍니다. 이 예제에서는 RESTful PHP 백엔드와의 상호 작용을 위해 React.js로 빌드된 프런트엔드를 사용합니다.

- React.js 웹 애플리케이션을 AWS 서비스와 통합합니다.
- Amazon RDS 테이블의 항목을 나열, 추가, 업데이트 및 삭제합니다.
- Amazon SES를 사용하여 필터링된 작업 항목에 대한 이메일 보고서를 보냅니다.
- 포함된 AWS CloudFormation 스크립트를 통해 예제 리소스를 배포하고 관리합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3)을 사용하여 Amazon Aurora Serverless 데이터베이스에서 작업 항목을 추적하고 Amazon Simple Email Service(Amazon SES)를 통해 보고서를 이메일로 보내는 REST 서비스 생성 방법을 보여 줍니다. 이 예제는 Flask 웹 프레임워크를 사용하여 HTTP 라우팅을 처리하고 React 웹 페이지와 통합하여 완전한 기능을 갖춘 웹 애플리케이션을 제공합니다.

- AWS 서비스와 통합되는 Flask REST 서비스를 구축합니다.
- Aurora Serverless 데이터베이스에 저장된 작업 항목을 읽고, 쓰고, 업데이트합니다.
- 데이터베이스 보안 인증 정보가 포함된 AWS Secrets Manager 암호를 생성하고 이를 사용하여 데이터베이스에 대한 호출을 인증합니다.
- Amazon SES를 사용하여 작업 항목에 대한 이메일 보고서를 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Aurora
- Amazon RDS
- Amazon RDS 데이터 서비스
- Amazon SES

AWS SDK 개발자 가이드 및 코드 예시의 전체 목록은 AWS SDK와 함께 이 서비스 사용 단원을 참조하세요. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon Aurora 모범 사례

데이터의 사용 또는 Amazon Aurora DB 클러스터로의 데이터 마이그레이션에 대한 일반적인 모범 사례 및 옵션에 대한 정보를 확인할 수 있습니다.

Amazon Aurora에 대한 모범 사례 중 일부는 특정 데이터베이스 엔진으로 국한됩니다. 데이터베이스 엔진에 특정한 Aurora 모범 사례에 대한 자세한 정보는 다음을 참조하십시오.

데이터베이스 엔진	모범 사례
Amazon Aurora MySQL	Amazon Aurora MySQL 모범 사례섹션을 참조하십시오.
Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL 모범 사례섹션을 참조하십시오.

> **Note**
>
> Aurora에 대한 일반적인 권장 사항은 Amazon Aurora의 권장 사항 단원을 참조하십시오.

주제

Amazon Aurora에 대한 기본 운영 지침

다음은 Amazon Aurora로 작업할 때 모든 사용자가 따라야 하는 기본 운영 지침입니다. Amazon RDS 서비스 수준 계약에 다음 지침을 따르도록 명시되어 있습니다.

- 메모리, CPU 및 스토리지 사용을 모니터링합니다. Amazon CloudWatch에서 사용 패턴이 변경되거나 사용자가 배포 용량에 도달했을 때 알림을 받도록 설정할 수 있으므로 이렇게 하면 시스템 성능 및 가용성을 유지할 수 있습니다.

- 클라이언트 애플리케이션이 DB 인스턴스의 DNS(Domain Name Service) 데이터를 캐시하는 경우 TTL(Time-to-Live) 값을 30초 미만으로 설정합니다. 장애 조치 후에 DB 인스턴스의 기본 IP 주소가 변경될 수 있습니다. 따라서 DNS 데이터를 오랜 시간 동안 캐시하면 애플리케이션이 더 이상 서비스되지 않는 IP 주소에 연결하려 할 경우 연결 오류로 이어질 수 있습니다. 연결에서 리더 엔드포인트를 사용하고 읽기 전용 복제본 인스턴스 중 하나가 유지 관리 중이거나 삭제된 경우에도 여러 읽기 전용 복제본이 있는 Aurora DB 클러스터에 연결 실패가 발생할 수 있습니다.
- DB 클러스터의 장애 조치를 테스트하여 사용 사례 절차에 소요되는 시간을 파악하십시오. 장애 조치 테스트를 통해 DB 클러스터에 액세스하는 애플리케이션이 장애 조치 후 자동으로 새 DB 클러스터에 연결할 수 있도록 할 수 있습니다.

DB 인스턴스 RAM 권장 사항

성능을 최적화하려면 작업 세트가 거의 완전히 메모리에 상주하도록 RAM을 충분히 할당하십시오. 작업 세트가 거의 전부 메모리에 있는지 여부를 판별하려면 Amazon CloudWatch에서 다음 지표를 조사하십시오.

- `VolumeReadIOPS` – 이 지표는 5분 간격으로 보고되는 클러스터 볼륨에서 읽기 I/O 작업의 평균 수를 측정합니다. `VolumeReadIOPS`의 값은 작고 안정적이어야 합니다. 경우에 따라 읽기 I/O가 급증하거나 평소보다 높을 수도 있습니다. 그런 경우 DB 클러스터의 DB 인스턴스를 조사하여 어떤 DB 인스턴스가 I/O 증가를 야기하고 있는지 확인합니다.

> **Tip**
>
> Aurora MySQL 클러스터가 병렬 쿼리를 사용하는 경우 `VolumeReadIOPS` 값이 증가할 수 있습니다. 병렬 쿼리는 버퍼 풀을 사용하지 않습니다. 따라서 쿼리는 빠르지만 이렇게 최적화된 프로세싱은 읽기 작업 및 관련 비용을 증가시킬 수 있습니다.

- `BufferCacheHitRatio` – 이 지표는 DB 클러스터에서 DB 인스턴스의 버퍼 캐시가 제공하는 요청 백분율을 측정합니다. 이 지표는 메모리에서 제공되는 데이터의 양에 대한 통찰력을 제공합니다.

  적중률이 높다면 DB 인스턴스에 사용 가능한 메모리가 충분하다는 뜻입니다. 적중률이 낮다면 이 DB 인스턴스에 대한 쿼리가 디스크로 자주 이동한다는 뜻입니다. 워크로드를 조사하여 이 동작의 원인이 되는 쿼리를 확인하십시오.

워크로드를 조사한 후에 더 많은 메모리가 필요하다면 DB 인스턴스 클래스를 RAM이 더 큰 클래스로 확장하는 것을 고려합니다. 이렇게 한 후 앞에서 언급한 지표를 조사하고 필요에 따라 계속 확장할 수

있습니다. Aurora 클러스터가 40TB보다 큰 경우 db.t2, db.t3 또는 db.t4g 인스턴스 클래스를 사용하지 마세요.

자세한 내용은 Amazon Aurora에 대한 Amazon CloudWatch 지표 단원을 참조하십시오.

AWS 데이터베이스 드라이버

애플리케이션 연결에 AWS 드라이버 제품군을 사용하는 것이 좋습니다. 더 빠른 전환 및 장애 조치 시간, AWS Secrets Manager, AWS Identity and Access Management(IAM) 및 페더레이션 ID를 사용한 인증을 지원하도록 설계된 드라이버입니다. AWS 드라이버는 DB 클러스터 상태 모니터링과 클러스터 토폴로지 파악을 통해 새 라이터를 결정합니다. 이 접근 방식은 전환 및 장애 조치 시간을 오픈 소스 드라이버의 경우 수십 초였던 것에 비해 10초 미만으로 단축합니다.

새로운 서비스 기능이 도입됨에 따라 AWS 드라이버 제품군의 목표는 이러한 서비스 기능에 대한 지원을 기본 제공하는 것입니다.

자세한 내용은 AWS 드라이버를 사용하여 Aurora DB 클러스터에 연결 단원을 참조하십시오.

Amazon Aurora 모니터링

Amazon Aurora는 모니터링을 통해 Aurora DB 클러스터의 상태와 성능을 평가할 수 있는 여러 가지 지표와 인사이트를 제공합니다. AWS Management Console, AWS CLI 및 CloudWatch API와 같은 다양한 도구를 사용하여 Aurora 지표를 볼 수 있습니다. 성능 개선 도우미 대시보드에서 결합된 성능 개선 도우미 및 CloudWatch 지표를 확인하고 DB 인스턴스를 모니터링할 수 있습니다. 이 모니터링 보기를 사용하려면 DB 인스턴스에서 성능 개선 도우미가 켜져 있어야 합니다. 이 모니터링에 대한 자세한 내용은 성능 개선 도우미 대시보드와 결합된 지표 보기 섹션을 참조하세요.

특정 기간에 대한 성과 분석 보고서를 생성하고 식별된 인사이트와 문제 해결을 위한 권장 사항을 볼 수 있습니다. 자세한 내용은 성능 개선 도우미에서 성과 분석 보고서 생성 섹션을 참조하세요.

DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업

프로덕션 DB 클러스터에 파라미터 그룹 변경 사항을 적용하기 전에 테스트 DB 클러스터에서 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 변경 사항을 시험해 보는 것이 좋습니다. DB 엔진 파라미터를 잘못 설정하면 성능 저하 및 시스템 불안정 등의 의도하지 않은 부작용이 있을 수 있습니다.

DB 엔진 파라미터를 수정할 때 항상 주의를 기울이고 DB 파라미터 그룹을 수정하기 전에 DB 클러스터를 백업하십시오. DB 클러스터 백업에 대한 자세한 정보는 Amazon Aurora DB 클러스터 백업 및 복구 단원을 참조하십시오.

Amazon Aurora 모범 사례 비디오

YouTube의 AWS Online Tech Talks 채널에는 보다 안전하고 가용성이 높은 Amazon Aurora DB 클러스터를 생성 및 구성하는 모범 사례에 대한 비디오 프레젠테이션이 포함되어 있습니다. 고가용성을 위한 Amazon Aurora 모범 사례를 참조하십시오.

Amazon Aurora에서 개념 증명 수행

다음은 Aurora에 대해 개념 증명을 설정하고 실행하는 방법에 대한 설명입니다. 개념 증명은 Aurora가 애플리케이션에 적합한지 확인하기 위해 수행하는 조사입니다. 개념 증명을 통해 고객님의 고유한 데이터베이스 애플리케이션의 컨텍스트에서 Aurora 기능을 이해할 수 있고 Aurora가 현재 데이터베이스 환경과 어떻게 대비되는지 알 수 있습니다. 또한 데이터 이동, SQL 코드 포트, 성능 튜닝, 현재 관리 절차 변경에 필요한 작업의 수준을 알 수 있습니다.

이 주제에서는 아래에 나열된 것과 같이 개념 증명을 실행하는 데 수반되는 높은 수준의 절차 및 의사 결정의 개요와 단계별 요점이 정리되어 있습니다. 자세한 지침을 얻으려면 특정 주제에 대한 전체 문서로 연결되는 링크를 따라가면 됩니다.

Aurora 개념 증명 개요

Amazon Aurora에 대한 개념 증명을 수행할 때는 기존 데이터 및 SQL 애플리케이션을 Aurora로 포팅하기 위해 필요한 것이 무엇인지 알아야 합니다. 프로덕션 환경을 대표하는 다량의 데이터 및 활동을 사용하여 규모에 따라 Aurora의 여러 가지 중요 속성을 시험합니다. 이 작업의 목표는 이전 데이터베이스 인프라에 맞지 않게 만드는 과제와 Aurora의 장점이 서로 잘 부합하는지에 대한 확신을 얻는 것입니다. 개념 증명을 마치는 시점에 더 큰 규모의 성능 벤치마킹과 애플리케이션 테스트를 수행할 확고한 계획이 수립됩니다. 이 시점에서 프로덕션 배포까지의 과정 중 가장 큰 작업 항목에 대해 이해하게 됩니다.

모범 사례에 대한 다음 조언을 통해 벤치마킹 중에 문제를 일으키는 일반적인 실수를 피할 수 있습니다. 그러나 이 주제에서는 벤치마크 수행 및 성능 튜닝 수행의 단계별 과정은 다루지 않습니다. 그러한 절차는 워크로드와 고객님이 사용하는 Aurora 기능에 따라 달라집니다. 자세한 내용은 Aurora DB 클러스터의 성능 및 확장 관리, Amazon Aurora MySQL 성능 개선 사항, Amazon Aurora PostgreSQL의 성능 및 규모 조정, 성능 개선 도우미를 통한 Amazon Aurora 모니터링와 같은 성능 관련 문서를 참조하십시오.

이 주제에서 제공하는 정보는 조직이 코드를 작성하고 스키마를 설계하며 MySQL 및 PostgreSQL 오픈 소스 데이터베이스 엔진을 지원하느 애플리케이션에 주로 적용됩니다. 애플리케이션 프레임워크에서 생성하는 상업용 애플리케이션 또는 코드를 테스트하는 경우, 모든 지침을 적용할 수 있는 유연성을 갖지 못할 수 있습니다. 그러한 경우 AWS 담당자와 함께 사용하는 애플리케이션 유형에 대한 Aurora 모범 사례 또는 사례 연구가 있는지 확인하세요.

1. 목표 식별

Aurora를 개념 증명의 일부로 평가할 때 무엇을 측정할 것이며 시험의 성공을 어떤 방식으로 평가할 것인지 선택해야 합니다.

애플리케이션의 모든 기능이 Aurora와 호환되는지 반드시 확인해야 합니다. Aurora 메이저 버전은 해당 MySQL 및 PostgreSQL 메이저 버전과도 유선 호환되므로 이러한 엔진을 위해 개발된 애플리케이션은 대부분 Aurora와도 호환됩니다. 하지만 그럼에도 개별 애플리케이션에 대해 호환성을 검증해야 합니다.

예를 들어 Aurora 클러스터를 설정할 때 선택하는 구성 중 일부는 특정 데이터베이스 기능을 사용할 수 있는지 또는 사용해야 하는지에 대해 영향을 미칩니다. 프로비저닝되었다라는 표현을 쓰는 가장 범용성이 큰 Aurora 클러스터로 시작할 수 있습니다. 그런 다음 서버리스 또는 병렬 쿼리와 같은 특수 구성이 고객님의 워크로드에 이점을 제공하는지 여부를 판단할 수 있습니다.

다음 질문을 사용하면 고객님의 목표를 식별하고 계량화하는 데 도움이 됩니다.

- Aurora는 워크로드의 모든 기능적 사용 사례를 지원합니까?
- 어떤 데이터세트 크기 또는 로드 수준을 원하십니까? 그러한 수준으로 규모를 조정할 수 있습니까?
- 고객님의 구체적인 쿼리 처리량 또는 지연 요구 사항은 무엇입니까? 이 요건을 달성할 수 있습니까?
- 워크로드에 대한 계획된 또는 계획되지 않은 가동 중단을 수용할 수 있는 최소 시간은 얼마입니까? 이 목표를 달성할 수 있습니까?
- 운영 효율성 유지에 필수적인 지표는 무엇입니까? 이 지표를 정확하게 모니터링할 수 있습니까?
- Aurora는 비용 절감, 배포 증가 또는 프로비저닝 속도와 같은 특정 비즈니스 목표를 지원합니까? 이러한 목표를 계량화할 수 있는 방법이 있습니까?
- 워크로드에 대한 모든 보안 및 규정 준수 요구 사항을 충족할 수 있습니까?

약간의 시간을 들여 Aurora 데이터베이스 엔진 및 플랫폼 기능에 대한 지식을 쌓고 서비스 설명서를 검토하십시오. 원하는 성과를 얻는 데 도움이 될 수 있는 모든 기능을 메모해 두십시오. 이러한 기능 중 한 가지는 AWS 데이터베이스 블로그 게시글 통합 워크로드를 위해 MySQL과 호환되는 Amazon Aurora를 계획하고 최적화하는 방법에서 설명하는 워크로드 통합입니다. 또 다른 한 가지는 Amazon Aurora 사용 설명서의 Aurora 복제본을 사용하는 Amazon Aurora Auto Scaling에 설명된 수요 기반 확장입니다. 그밖에 성능 향상 또는 단순화된 데이터베이스 연산이 있습니다.

2. 워크로드 특성에 대한 이해

의도한 사용 사례의 맥락에서 Aurora를 평가하세요. Aurora는 온라인 트랜잭션 처리(OLTP) 워크로드에 좋은 선택입니다. 또한 별도의 데이터 웨어하우스 클러스터를 프로비저닝하지 않고도 실시간 OLTP 데이터를 보유한 클러스터에서 보고서를 실행할 수 있습니다. 다음 특성을 확인하여 사용 사례가 이 범주에 해당하는지 알아낼 수 있습니다.

- 동시 클라이언트가 수십, 수백 또는 수천에 달하는 높은 동시성
- 지연 시간이 밀리초에서 몇 초로 짧은 대량의 쿼리
- 짧은 실시간 트랜잭션
- 인덱스 기반 조회 기능을 갖춘 고도로 선택적인 쿼리 패턴
- HTAP의 경우, Aurora 병렬 쿼리를 이용할 수 있는 분석적 쿼리

데이터베이스 선택에 영향을 미치는 주된 요인 중 한 가지는 데이터 속도입니다. 빠른 속도에는 매우 자주 삽입되고 업데이트되는 데이터가 수반됩니다. 이러한 시스템에는 데이터베이스에 대한 수천 건의 연결과 수십만 건의 동시 쿼리 읽기 및 쓰기가 발생할 수 있습니다. 고속 시스템에서 발생하는 쿼리는 비교적 적은 수의 행에 영향을 미치는 것이 보통이며, 일반적으로 동일 행에 있는 여러 열에 액세스합니다.

Aurora는 고속 데이터 처리를 위해 설계되었습니다. 워크로드에 따라 단일 r4.16xlarge DB 인스턴스가 포함된 Aurora 클러스터는 초당 600,000건 이상의 `SELECT` 문을 처리할 수 있습니다. 이러한 클러스터는 워크로드에 따라 초당 200,000개의 `INSERT`, `UPDATE` 및 `DELETE` 문을 처리할 수 있습니다. Aurora는 행 스토어 데이터베이스로, 대량의 처리량 및 고도로 병렬화된 OLTP 워크로드에 이상적입니다.

또한 Aurora는 OLTP 워크로드를 처리하는 동일한 클러스터에서 보고 쿼리를 실행할 수도 있습니다. Aurora는 최대 15개의 [복제본](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Replication.html)을 지원하며, 각 복제본은 기본 인스턴스에서 평균 10~20밀리초 이내에 지원됩니다. 분석가는 데이터를 별도 데이터 웨어하우스 클러스터에 복사하지 않고도 실시간으로 OLTP 데이터를 쿼리할 수 있습니다. 병렬 쿼리 기능을 사용하는 Aurora 클러스터를 통해 처리 필터링 대부분과 집계 작업을 엄청나게 분산된 Aurora 스토리지 하위 시스템으로 오프로드할 수 있습니다.

이 계획 단계를 사용하여 Aurora, 기타 AWS 서비스, AWS Management Console 및 AWS CLI의 기능을 숙지하세요. 또한 이러한 것들이 고객님이 개념 증명에 사용할 계획인 기타 도구와 어떻게 연동되는지 확인하십시오.

3. AWS Management Console 또는 AWS CLI를 이용한 실습

다음 단계로 AWS Management Console 또는 AWS CLI를 이용한 실습을 통해 이러한 도구와 Aurora를 숙지하세요.

AWS Management Console을 이용한 실습

Aurora 데이터베이스 클러스터를 이용한 다음과 같은 초기 활동은 주로 AWS Management Console 환경을 숙지하고 Aurora 클러스터 설정 및 수정을 연습할 수 있도록 하기 위한 것입니다. Amazon RDS에서 MySQL 호환 및 PostgreSQL 호환 데이터베이스 엔진을 사용하는 경우, Aurora를 사용할 때 이 지식을 활용할 수 있습니다.

Aurora 공유 스토리지 모델과 복제, 스냅샷과 같은 기능을 이용하여 전체 데이터베이스 클러스터를 고객님이 자유롭게 조작할 수 있는 다른 종류의 객체로 취급할 수 있습니다. 개념 증명 중에 Aurora 클러스터의 용량을 자주 설정, 제거 및 변경할 수 있습니다. 용량, 데이터베이스 설정 및 물리적 데이터 레이아웃에 대한 초기 선택에 락인(lock-in)되지 않습니다.

시작하려면 빈 Aurora 클러스터를 설정하십시오. 초기 실험에 대해 provisioned(프로비저닝된) 용량 유형과 regional(리전) 위치를 선택합니다.

SQL 명령줄 애플리케이션과 같은 클라이언트 프로그램을 사용해 이 클러스터에 연결합니다. 처음에는 클러스터 엔드포인트를 사용해 연결합니다. 이 엔드포인트에 연결하여 데이터 정의 언어(DDL) 문과 추출, 변환, 로드(ETL) 프로세스와 같은 쓰기 연산을 수행합니다. 개념 증명 후반부에는 쿼리 워크로드를 클러스터에 있는 여러 DB 인스턴스에 배포하는 리더 엔드포인트를 사용해 쿼리 집약적인 세션을 연결합니다.

Aurora 복제본을 추가하여 클러스터의 규모를 확장하십시오. 이를 위한 절차는 Amazon Aurora를 사용한 복제 단원을 참조하십시오. AWS 인스턴스 클래스를 변경하여 DB 인스턴스의 규모를 확장 또는 축소합니다. 시스템 용량에 대한 초기 추정치가 부정확한 경우 처음부터 다시 시작하지 않고 나중에 조정할 수 있도록 Aurora가 이러한 종류의 연산을 어떻게 단순화하는지 이해하십시오.

스냅샷을 생성하여 다른 클러스터로 복원합니다.

클러스터 지표를 검토하여 시간 경과에 따른 활동을 보고 지표가 클러스터 내의 DB 인스턴스에 어떻게 적용되는지 알아보십시오.

처음에는 AWS Management Console을 통해 이러한 작업을 수행하는 방법을 숙지하면 도움이 됩니다. Aurora로 할 수 있는 작업이 무엇인지 이해하고 나면 AWS CLI를 사용해 이러한 작업을 자동화하는 단계로 나아갈 수 있습니다. 다음 단원에서는 개념 증명 기간 중 이러한 활동의 절차와 모범 사례에 대한 더 자세한 내용을 보실 수 있습니다.

AWS CLI을 이용한 실습

개념 증명 설정에서도 배포 및 관리 절차를 자동화하는 것이 좋습니다. 이를 위해서는 AWS CLI를 숙지해야 합니다. Amazon RDS에서 MySQL 호환 및 PostgreSQL 호환 데이터베이스 엔진을 사용하는 경우, Aurora를 사용할 때 이 지식을 활용할 수 있습니다.

Aurora에는 일반적으로 클러스터 내에 정렬된 DB 인스턴스 그룹이 수반됩니다. 따라서 많은 연산에는 어떤 DB 인스턴스가 클러스터와 연결되어 있는지 확인하고 모든 인스턴스의 루프에서 관리 연산을 수행하는 작업이 수반됩니다.

예를 들어 Aurora 클러스터 생성과 같은 단계를 자동화한 후 더 규모가 큰 인스턴스 클래스로 확장하거나 추가 DB 인스턴스로 확장할 수 있습니다. 이렇게 하면 개념 증명 중에서 어느 단계이든 반복할 수 있고 다양한 종류 또는 구성의 Aurora 클러스터가 포함된 가정(what-if) 시나리오를 탐색하는 데 도움이 됩니다.

AWS CloudFormation과 같은 인프라 배포 도구의 기능 및 제한 사항에 대해 알아보십시오. 개념 증명 컨텍스트에서 수행하는 활동은 프로덕션 용도로 적합하지 않다는 것을 알게 될 수도 있습니다. 예를 들어 수정을 위한 AWS CloudFormation 동작은 새 인스턴스를 생성하고 데이터를 포함한 현재 인스턴스를 삭제하는 것입니다. 이 동작에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 스택 리소스의 업데이트 동작을 참조하세요.

4. Aurora 클러스터 생성

Aurora에서는 DB 인스턴스를 클러스터에 추가하고 DB 인스턴스를 더 강력한 인스턴스 클래스로 확장하여 가정(what-if) 시나리오를 탐색할 수 있습니다. 또한 구성 설정이 다양한 클러스터를 생성하여 동일한 워크로드를 병렬로 실행할 수 있습니다. Aurora는 DB 클러스터를 설정, 제거 및 구성할 수 있는 유연성이 높습니다. 이를 고려할 때 개념 증명 프로세스 초기 단계에서 이러한 기법을 실습하는 것이 도움이 됩니다. Aurora 클러스터 생성을 위한 일반적인 절차는 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

실현 가능한 경우 다음 설정을 사용해 클러스터를 시작하십시오. 염두에 두고 계신 구체적인 특정 사용 사례가 있는 경우에만 이 단계를 건너뛰십시오. 예를 들어 고객님의 사용 사례에 특수한 종류의 Aurora 클러스터가 필요한 경우 이 단계를 건너뛸 수 있습니다. 또는 데이터베이스 엔진 및 버전의 특정 조합이 필요한 경우 건너뛸 수 있습니다.

- [간편 생성(Easy create)]을 비활성화합니다. 개념 증명의 경우 나중에 동일한 또는 약간 다른 클러스터를 생성할 수 있도록 고객님이 선택하는 모든 설정을 숙지하는 것이 좋습니다.
- 최신 DB 엔진 버전을 사용하세요. 데이터베이스 엔진 및 버전의 이러한 조합은 다른 Aurora 기능과 폭넓게 호환되며 프로덕션 애플리케이션의 고객 사용량이 상당히 큽니다.

  - Aurora MySQL 버전 3.x (MySQL 8.0 호환)
  - Aurora PostgreSQL 버전 15.x 또는 16.x
- 개발/테스트 템플릿을 선택합니다. 이 선택은 개념 증명 활동에 중요하지는 않습니다.
- [DB 인스턴스 클래스(DB instance class)]에서 [메모리 최적화 클래스((Memory optimized classes)]와 [xlarge] 인스턴스 클래스 중 하나를 선택합니다. 나중에 인스턴스 클래스를 확장 또는 축소 조정할 수 있습니다.
- [다중 AZ 배포(Multi-AZ Deployment)]에서 [다른 AZ에 Aurora 복제본 또는 리더 노드 생성(Create an Aurora Replica or Reader node in a different AZ)]을 선택합니다. Aurora가 지닌 가장 유용한 속성 중 다수는 여러 DB 인스턴스로 구성된 클러스터와 관련된 것입니다. 새로운 클러스터에서는 항상 최소 2개의 DB 인스턴스로 시작하는 것이 타당합니다. 두 번째 DB 인스턴스에 대해 다른 가용 영역을 사용하면 다양한 고가용성 시나리오를 테스트하는 데 도움이 됩니다.
- DB 인스턴스의 이름을 선택할 때 일반적인 이름 지정 규칙을 사용하십시오. 클러스터 DB 인스턴스를 “라이터”로 참조해서는 안 됩니다. 서로 다른 DB 인스턴스는 이러한 역할을 필요에 따라 수임하기 때문입니다. `clustername-az-serialnumber`과 같은 것을 사용하는 것이 좋습니다(예: `myprodappdb-a-01`). 이 조각들은 DB 인스턴스와 그 배치를 고유하게 식별합니다.
- Aurora 클러스터에 대해 백업 보존을 ‘높음’으로 설정합니다. 장기 보존 기간을 통해 최대 35일 동안 PITR(특정 시점으로 복구)를 수행할 수 있습니다. DDL 및 데이터 조작 언어(DML) 문과 관련된 테스트를 실행한 후에 데이터베이스를 알려진 상태로 재설정할 수 있습니다. 또한 실수로 데이터를 삭제 또는 변경한 경우 이를 복구할 수 있습니다.
- 클러스터 생성 시 추가 복구, 로깅 및 모니터링 기능을 활성화합니다. 역추적, 성능 개선 도우미, 모니터링 및 로그 내보내기에서 사용 가능한 모든 선택지를 활성화합니다. 이러한 기능을 활성화하면 워크로드에 대한 역추적, 확장 모니터링 또는 성능 개선 도우미와 같은 기능의 적합성을 테스트할 수 있습니다. 또한 개념 증명 중에 성능을 쉽게 조사하고 문제 해결을 수행할 수 있습니다.

5. 스키마 설정

Aurora 클러스터에서 애플리케이션을 위한 데이터베이스, 테이블, 인덱스, 외래 키 및 기타 스키마 객체를 설정하십시오. 다른 MySQL 호환 또는 PostgreSQL 호환 데이터베이스 시스템으로부터 다른 시스템으로 이동하는 경우 이 단계는 단순하고 간결할 것으로 기대하시면 됩니다. 데이터베이스 엔진에 대해서는 동일한 SQL 구문 및 명령줄 또는 고객님이 잘 알고 있는 기타 클라이언트 애플리케이션을 사용하십시오.

클러스터에 SQL 문을 발행하려면 클러스터 엔드포인트를 찾아 그 값을 클라이언트 애플리케이션에 연결 파라미터로 제공하십시오. 클러스터 세부 정보 페이지에 있는 Connectivity(연결성) 탭에서 클러스터 엔드포인트를 찾을 수 있습니다. 이 클러스터 엔드포인트는 Writer(라이터)라고 레이블이 지정된

것입니다. Reader(리더)라는 레이블이 지정된 다른 엔드포인트는 보고서 또는 기타 읽기 전용 쿼리를 실행하는 최종 사용자에게 공급할 수 있는 읽기 전용 연결을 대표합니다. 고객님의 클러스터에 연결하는 것과 관련된 모든 문제에 대한 도움을 받으려면 Amazon Aurora DB 클러스터에 연결 단원을 참조하십시오.

다른 데이터베이스 시스템에서 스키마와 데이터를 포트하는 경우 이 시점에서 일부 스키마 변경이 이루어질 것으로 기대해야 합니다. 이러한 스키마 변경 사항은 Aurora에서 사용할 수 있는 SQL 구문과 역량과 부합해야 합니다. 이 시점에서 특정 열, 제약 조건, 트리거 또는 기타 스키마 객체는 생략할 수 있습니다. 이렇게 하면 특히 이 객체들이 Aurora 호환성을 위해 재작업이 필요하고 개념 증명 관련 목표에는 중요하지 않은 경우 도움이 될 수 있습니다.

기본 엔진이 Aurora의 기본 엔진과 다른 데이터베이스 시스템에서 마이그레이션하는 경우 AWS Schema Conversion Tool(AWS SCT)을 사용해 프로세스를 간소화하는 것을 고려하세요. 자세한 내용은 AWS Schema Conversion Tool 사용 설명서를 참조하세요. 마이그레이션 및 포트 활동에 대한 일반적인 세부 정보는 AWS 백서, Migrating Your Databases to Amazon Aurora(Amazon Aurora로 데이터베이스 마이그레이션)를 참조하세요.

이 단계에서는 스키마 설정과 관련해(예: 인덱싱 전략 또는 분할된 테이블과 같은 기타 테이블 구조) 비효율성이 있는지 평가할 수 있습니다. 이러한 비효율성은 DB 인스턴스가 여러 개이고 워크로드가 큰 클러스터에 애플리케이션을 배포하면 증폭될 수 있습니다. 지금 또는 전체 벤치마크 테스트와 같은 추후 활동 중에 이러한 성능 속성을 미세 조정할 수 있을지 고려하십시오.

6. 데이터 가져오기

개념 증명 중에 이전 데이터베이스 시스템에서 데이터 또는 대표 샘플을 가져오십시오. 실현 가능한 경우 각 테이블에 최소한 일부라도 데이터를 설정하십시오. 이렇게 하면 모든 데이터 유형 및 스키마 기능의 호환성을 테스트하는 데 도움이 됩니다. 기본 Aurora 기능을 사용해본 후에 데이터 양을 늘리십시오. 개념 증명이 끝나가는 시점에 정확한 결론을 도출할 수 있을 만큼 큰 데이터세트로 ETL 도구, 쿼리 및 전체 워크로드를 테스트해야 합니다.

몇 가지 기법을 사용해 물리적 또는 논리적 백업 데이터를 Aurora로 가져올 수 있습니다. 자세한 내용은 개념 증명에 사용 중인 데이터베이스 엔진에 따라 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 또는 데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션 단원을 참조하십시오.

현재 고려 중인 ETL 도구 및 기술을 실험해 보십시오. 고객님의 필요를 가장 잘 충족하는 것이 무엇인지 확인하십시오. 처리량과 유연성을 모두 고려하십시오. 예를 들어 어떤 ETL 도구는 일회성 전송을 수행하고 어떤 ETL 도구는 낡은 시스템에서 Aurora로의 지속적인 복제가 수반됩니다.

MySQL 호환 시스템에서 Aurora MySQL로 마이그레이션하는 경우 기본 데이터 전송 도구를 사용할 수 있습니다. PostgreSQL 호환 시스템에서 Aurora PostgreSQL으로 마이그레이션하는 경우에도 마찬가지입니다. Aurora에서 사용하는 기본 엔진이 아닌 다른 기본 엔진을 사용하는 데이터베이스 시스템에서 마이그레이션하는 경우 AWS Database Migration Service(AWS DMS)로 실험할 수 있습니다. AWS DMS에 대한 자세한 내용은 AWS Database Migration Service 사용 설명서를 참조하세요.

마이그레이션 및 포트 활동에 대한 자세한 내용은 AWS 백서인 Aurora 마이그레이션 핸드북을 참조하세요.

7. SQL 코드 포팅

SQL 및 연결된 애플리케이션을 사용해보려면 다양한 사례에 따라 다양한 수준의 작업이 필요합니다. 특히 작업의 수준은 MySQL 호환 또는 PostgreSQL 호환 시스템 또는 다른 종류의 시스템 중 어느 시스템에서 이동하느냐에 따라 달라집니다.

- RDS for MySQL 또는 RDS for PostgreSQL에서 이동하는 경우 SQL 변경 사항은 Aurora를 이용해 원본 SQL 코드를 사용하고 필요한 변경 사항을 수동으로 통합할 수 있을 정도로 적습니다.
- 이와 마찬가지로 MySQL 또는 PostgreSQL과 호환되는 온프레미스 데이터베이스에서 이동하는 경우 원본 SQL 코드를 사용해보고 변경 사항을 수동으로 통합할 수 있습니다.
- 다른 상업용 데이터베이스에서 이동하는 경우에는 필수 SQL 변경 사항이 더 광범위합니다. 이 경우에는 AWS SCT 사용을 고려하십시오.

이 단계에서는 스키마 설정과 관련해(예: 인덱싱 전략 또는 분할된 테이블과 같은 기타 테이블 구조) 비효율성이 있는지 평가할 수 있습니다. 지금 또는 전체 벤치마크 테스트와 같은 추후 활동 중에 이러한 성능 속성을 미세 조정할 수 있을지 고려하십시오.

애플리케이션에서 데이터베이스 연결 로직을 확인할 수 있습니다. Aurora 분산형 처리를 이용하려면 읽기 및 쓰기 연산에 대해 별도의 연결을 사용하고 쿼리 연산에 대해 비교적 짧은 세션을 사용해야 할 수 있습니다. 연결에 관한 자세한 내용은 9. Aurora에 연결 단원을 참조하십시오.

프로덕션 데이터베이스에서 문제를 해결하기 위해 타협과 거래를 해야 했는지 생각해 보십시오. 개념 증명 일정에 스키마 설계 및 쿼리를 개선할 수 있는 시간을 할애하십시오. 성능, 운영 비용 및 확장성 개선을 쉽게 달성할 수 있는지 여부를 판단하려면 다양한 Aurora 클러스터에서 원본 및 수정 애플리케이션을 나란히 사용해 보십시오.

마이그레이션 및 포트 활동에 대한 자세한 내용은 AWS 백서인 Aurora 마이그레이션 핸드북을 참조하세요.

8. 구성 설정 지정

Aurora 개념 증명 실습의 일부로 데이터베이스 구성 파라미터를 검토할 수도 있습니다. 현재 환경에서 성능 및 확장성을 위해 MySQL 또는 PostgreSQL 구성 설정을 이미 튜닝했을 수도 있습니다. Aurora 스토리지 하위 시스템은 고속 스토리지 하위 시스템을 통해 분산형 클라우드 기반 환경에 맞게 조정 및 튜닝됩니다. 결과적으로 다수의 이전 데이터베이스 엔진 설정은 적용되지 않습니다. 초기 실험은 기본 Aurora 구성 설정으로 수행하는 것이 좋습니다. 성능 및 확장성 병목 현상이 발생하는 경우에만 현재 환경의 설정을 다시 적용하십시오. 관심이 있는 경우 이 주제에 관해 더 깊이 알아보려면 AWS 데이터베이스 블로그의 Aurora 스토리지 엔진 소개를 참조하세요.

Aurora 덕분에 특정 애플리케이션 또는 사용 사례에 최적의 구성 설정을 쉽게 재사용할 수 있습니다. 각 DB 인스턴스에 대해 별도의 구성 파일을 편집하는 대신에 전체 클러스터 또는 특정 DB 인스턴스에 할당하는 파라미터 세트를 관리하십시오. 예를 들어 시간대 설정은 클러스터의 모든 DB 인스턴스에 적용되며 각 DB 인스턴스에 대한 페이지 캐치 크기 설정을 조정할 수 있습니다.

기본 파라미터 세트 중 하나로 시작하고, 미세 조정해야 할 파라미터에만 변경 사항을 적용하십시오. 파라미터 그룹 사용에 대한 자세한 내용은 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 단원을 참조하십시오. Aurora 클러스터에 적용되거나 적용되지 않는 구성 설정에 대해서는 데이터베이스 엔진에 따라 Aurora MySQL 구성 파라미터 또는 Amazon Aurora PostgreSQL parameters를 참조하십시오.

9. Aurora에 연결

초기 스키마 및 데이터 설정을 수행하고 샘플 쿼리를 실행할 시점을 알게 됨에 따라 Aurora 클러스터에서 다양한 엔드포인트에 연결할 수 있습니다. 사용할 엔드포인트는 연산이 `SELECT` 문과 같은 읽기인지, 아니면 `CREATE` 또는 `INSERT` 문과 같은 쓰기인지에 따라 달라집니다. Aurora 클러스터에서 워크로드를 늘리고 Aurora 기능으로 실험을 할 때 애플리케이션이 각 연산을 적절한 엔드포인트에 할당하는 것이 중요합니다.

클러스터 엔드포인트를 쓰기 연산에 사용하면 읽기/쓰기 기능이 있는 클러스터의 DB 인스턴스에 항상 연결됩니다. 기본적으로 Aurora 클러스터에 있는 하나의 DB 인스턴스에만 읽기-쓰기 기능이 있습니다. 이 DB 인스턴스를 기본 인스턴스라고 합니다. 원본 기본 인스턴스를 사용할 수 없게 되면 Aurora는 장애 조치 메커니즘을 활성화하고 다른 DB 인스턴스가 기본 인스턴스로 인계를 받습니다.

이와 마찬가지로 `SELECT` 문을 리더 엔드포인트로 유도함으로써 쿼리를 처리하는 작업을 클러스터 내 DB 인스턴스에 분산할 수 있습니다. 각 리더 연결은 라운드 로빈 DNS 확인을 사용해 다른 DB 인스턴스에 할당됩니다. 읽기 전용 DB Aurora 복제본에서 대부분의 쿼리 작업을 하면 기본 인스턴스에 대한 로드가 줄어들어 DDL 및 DML 문을 처리할 수 있는 여유가 생깁니다.

이러한 엔드포인트를 사용하면 하드 코딩된 호스트 이름에 대한 종속성이 줄어들고 애플리케이션이 DB 인스턴스 장애에서 더 빨리 복구될 수 있습니다.

> **Note**
>
> Aurora에는 고객님이 생성하는 사용자 지정 엔드포인트가 있습니다. 이러한 엔드포인트는 개념 증명 중에는 대개 필요하지 않습니다.

Aurora 복제본은 복제본 지연 시간이 대개 10-20밀리초라 하더라도 이 지연 시간의 적용을 받습니다. 복제 지연 시간을 모니터링하고 데이터 일관성 요구 사항의 범위 내에 있는지 확인할 수 있습니다. 어떤 경우에는 읽기 쿼리에 강력한 읽기 일관성이 필요합니다(쓰기 후 읽기 일관성). 이 경우 해당 쿼리에 리더 엔드포인트가 아닌 클러스터 엔드포인트를 계속 사용할 수 있습니다.

분산형 병렬 실행을 위한 Aurora 기능을 최대한 활용하려면 연결 로직을 변경해야 할 수 있습니다. 목표는 모든 읽기 요청을 기본 인스턴스로 전송하는 일을 방지하는 것입니다. 읽기 전용 Aurora 복제본은 모든 동일 데이터와 함께 대기하면서 SELECT 문을 처리할 준비를 하고 있습니다. 애플리케이션 로직을 코딩하여 각각의 연산 유형에 적절한 엔드포인트를 사용하십시오. 다음과 같은 일반 지침을 따르십시오.

- 모든 데이터베이스 세션에 하드 코딩된 단일 연결 문자열을 사용하지 않도록 하십시오.
- 가능한 경우 DDL 및 DDL 문과 같은 쓰기 연산을 클라이언트 애플리케이션 코드 내 함수에 묶습니다. 이렇게 하면 다양한 종류의 연산에서 특정 연결을 사용하게 할 수 있습니다.
- 쿼리 작업을 위해 별도의 함수를 만듭니다. Aurora는 리더 엔드포인트에 대한 각각의 새 연결을 다른 Aurora 복제본에 할당하여 읽기 집약적인 애플리케이션에 대해 로드 밸런싱을 수행합니다.
- 쿼리 세트를 수반하는 연산의 경우 관련된 각 쿼리 세트가 완료되면 리더 엔드포인트에 대한 연결을 종료한 후 다시 엽니다. 이 기능을 소프트웨어 스택에서 사용할 수 있다면 연결 풀링을 사용하십시오. 서로 다른 연결에 쿼리를 유도하면 Aurora가 클러스터 내 DB 인스턴스에 읽기 워크로드를 분산하는 데 도움이 됩니다.

Aurora의 연결 관리 및 엔드포인트에 대한 일반적인 정보는 [Amazon Aurora DB 클러스터에 연결](#) 단원을 참조하십시오. 이 주제에 관해 자세히 알아보려면 [Aurora MySQL 데이터베이스 관리자용 핸드북 – 연결 관리](#)를 참조하십시오.

10. 워크로드 실행

스키마, 데이터 및 구성 설정을 완료한 후에는 워크로드를 실행하여 클러스터에 대한 실습을 시작할 수 있습니다. 프로덕션 워크로드의 주요 속성을 반영하는 워크로드를 개념 증명에 사용하십시오. 항상 sysbench 또는 TPC-C와 같은 합성 벤치마크보다는 실제 테스트 및 워크로드를 사용해 성능 관련 의사 결정을 내리는 것이 좋습니다.

가능한 한 애플리케이션이 실행될 실제 조건을 복제하십시오. 예를 들어 사용자는 일반적으로 Aurora 클러스터와 동일한 AWS 리전과 동일한 Virtual Private Cloud(VPC)에 있는 Amazon EC2 인스턴스에서 애플리케이션 코드를 실행합니다. 프로덕션 애플리케이션이 여러 가용 영역에 걸쳐 있는 여러 EC2 인스턴스에서 실행되는 경우 이와 동일한 방법으로 개념 증명 환경을 설정하십시오. AWS 리전에 대한 자세한 내용은 Amazon RDS 사용 설명서의 리전 및 가용 영역을 참조하세요. Amazon VPC 서비스에 대한 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC란 무엇인가? 단원을 참조하십시오.

애플리케이션 작업의 기본 기능을 확인하고 나서 Aurora를 통해 데이터에 액세스할 수 있게 되었다면 Aurora 클러스터의 여러 속성을 실습해볼 수 있습니다. 시험해 보려는 기능 중 일부는 로드 밸런싱, 동시 트랜잭션 및 자동 복제와의 동시 연결입니다.

이 시점이 되면 데이터 전송 메커니즘이 익숙할 것이므로 더 큰 비율의 샘플 데이터를 이용해 테스트를 실행할 수 있습니다.

이 단계에서는 메모리 제한 및 연결 제한과 같은 구성 설정을 변경하면 결과가 어떻게 되는지 확인할 수 있습니다. 8. 구성 설정 지정에서 알아본 절차를 다시 살펴보십시오.

스냅샷 생성 및 복원과 같은 메커니즘을 이용해 실험할 수도 있습니다. 예를 들어 다양한 AWS 인스턴스 클래스, AWS 복제본의 수 등을 이용해 클러스터를 생성할 수 있습니다. 그런 다음 각 클러스터에서 스키마와 모든 데이터가 포함된 동일한 스냅샷을 복원할 수 있습니다. 이 주기에 대한 자세한 내용은 DB 클러스터 스냅샷 생성 및 DB 클러스터 스냅샷에서 복원 단원을 참조하십시오.

11. 성능 측정

이 영역의 모범 사례는 적합한 모든 도구와 프로세스를 설정하여 워크로드 연산 중에 비정상적 작동을 신속히 격리하도록 설계되었습니다. 또한 이러한 설정을 통해 해당되는 모든 원인을 안정적으로 식별할 수 있는지 확인할 수 있습니다.

항상 모니터링 탭을 검토하여 클러스터의 현재 상태를 확인하거나 시간 경과에 따른 추세를 살펴볼 수 있습니다. 이 탭은 각 Aurora 클러스터 또는 DB 인스턴스의 콘솔 세부 정보 페이지에 제공됩니다. 이 탭에는 Amazon CloudWatch 모니터링 서비스의 지표가 차트 형태로 표시됩니다. 지표를 이름, DB 인스턴스, 기간을 기준으로 필터링할 수 있습니다.

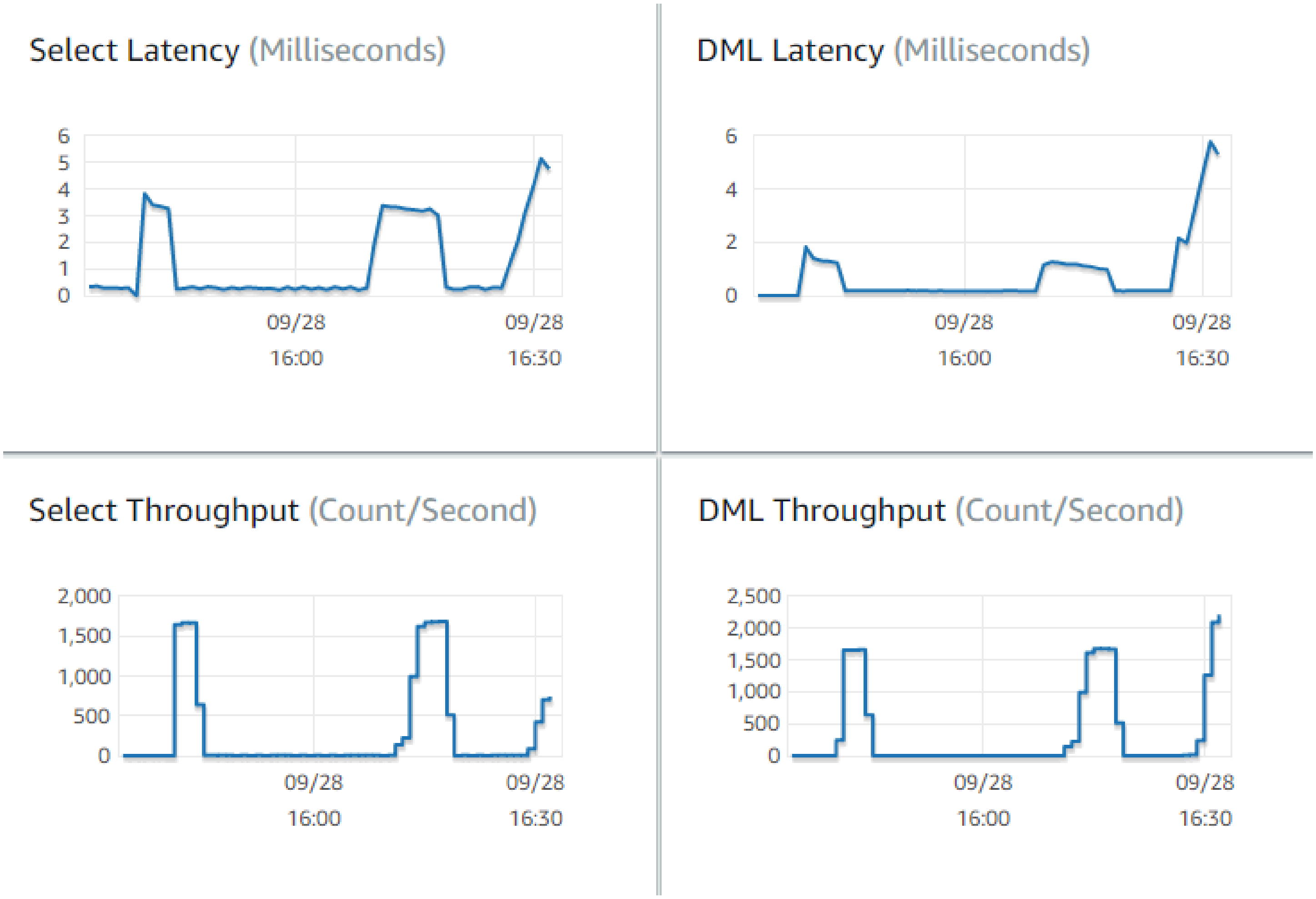
모니터링 탭에서 더 많은 사항을 선택할 수 있게 하려면 클러스터 설정에서 기본 모니터링 및 성능 개선 도우미를 활성화하십시오. 클러스터 설정 시 이를 선택하지 않았다면 이 선택 항목을 활성화할 수도 있습니다.

성능을 측정하려면 전체 Aurora 클러스터에 대한 활동을 보여주는 차트에 거의 의존해야 합니다. Aurora 복제본의 로드 및 응답 횟수가 이와 유사한지 확인할 수 있습니다. 읽기/쓰기 기본 인스턴스와 읽기 전용 Aurora 복제본 간에 작업이 분할되는 방식도 확인할 수 있습니다. DB 인스턴스 간 약간의 불균형 또는 하나의 DB 인스턴스에만 영향을 미치는 문제가 있는 경우 이 특정 인스턴스의 모니터링 탭을 검토할 수 있습니다.

환경 및 실제 워크로드를 설정하여 프로덕션 애플리케이션을 에뮬레이션한 후에는 Aurora가 얼마나 잘 작동하는지 성능을 측정할 수 있습니다. 답해야 할 가장 중요한 질문은 다음과 같습니다.

- Aurora가 초당 몇 건의 쿼리를 처리하고 있습니까? Throughput(처리량) 지표를 검토하여 다양한 연산 유형의 수치를 확인할 수 있습니다.
- Aurora가 특정 쿼리를 처리하는 데 평균적으로 시간이 얼마나 걸립니까? Latency(지연 시간) 지표를 검토하여 다양한 연산 유형의 수치를 확인할 수 있습니다.

처리량 및 지연 시간 지표를 보려면 [Amazon RDS 콘솔](https://console.aws.amazon.com/rds/)에서 해당 Aurora 클러스터의 모니터링 탭을 확인합니다. 다음 스크린샷은 모니터링 탭의 지연 시간 선택, DML 지연 시간, 처리량 선택, DML 처리량 예를 보여줍니다.



가능하다면 이러한 지표의 기준 값을 현재 환경에 설정하십시오. 불가능하다면 프로덕션 애플리케이션과 동등한 워크로드를 실행하여 Aurora 클러스터에 기준을 구성하십시오. 예를 들어 동시 사용자 및 쿼리의 수가 비슷한 Aurora 워크로드를 실행할 수 있습니다. 그런 다음 다양한 인스턴스 클래스, 클러스터 크기, 구성 설정 등으로 실험해 가면서 값이 어떻게 변하는지 관찰합니다.

처리량 숫자가 예상보다 낮으면 워크로드 처리를 위한 데이터베이스 성능에 영향을 미치는 요인을 더 조사하십시오. 이와 마찬가지로 지연 시간 숫자가 예상보다 더 높다면 추가 조사를 실시합니다. 이를 위해서는 DB 서버의 보조 지표(CPU, 메모리 등)를 모니터링해야 합니다. DB 인스턴스 각 지표의 한도에 근접하는지 확인할 수 있습니다. 또한 DB 인스턴스가 동시 쿼리, 더 큰 데이블에 대한 쿼리 등을 처리할 추가 용량이 얼마나 되는지도 확인할 수 있습니다.

Tip

예상 범위를 벗어나는 지표 값을 감지하려면 CloudWatch 경보를 설정하십시오.

이상적인 Aurora 클러스터 크기 및 용량을 평가하는 과정에서 오버-프로비저닝 리소스 없이 최상의 애플리케이션 성능을 달성하는 구성을 찾을 수 있습니다. 한 가지 중요한 요인은 Aurora 클러스터에 있는 DB 인스턴스에 적절한 크기를 찾는 것입니다. 현재 프로덕션 환경과 CPU 및 메모리 용량이 비슷한 인스턴스 크기를 선택하는 것부터 시작하십시오. 선택한 인스턴스 크기에서 워크로드에 대한 처리량 및 지연 시간 수치를 수집합니다. 이어서 인스턴스를 그다음으로 큰 크기로 확장합니다. 처리량 및 지연 시간 수치가 개선되는지 확인합니다. 또한 인스턴스 크기를 줄이고, 지연 시간 및 처리량 수치가 그대로 유지되는지 확인합니다. 이 작업의 목표는 가능한 한 가장 작은 인스턴스에서 최대 처리량을 최소 지연 시간으로 얻는 것입니다.

> **Tip**
>
> 기존 용량이 충분한 Aurora 클러스터 및 연의 크기를 조정하여 갑작스럽고 예측할 수 없는 트래픽 급등을 처리할 수 있게 합니다. 미션 크리티컬 데이터베이스의 경우 최소 20퍼센트의 예비 CPU 및 메모리 용량을 남겨 두십시오.

웜 및 안정적 상태에서 데이터베이스 성능을 측정하기에 충분할 만큼 오래 성능 테스트를 실행합니다. 이러한 안정적 상태에 도달하려면 수 분 또는 몇 시간 동안 워크로드를 실행해야 할 수 있습니다. 실행을 시작할 때 발생하는 약간의 편차는 정상적인 것입니다. 이러한 편차가 발생하는 이유는 각 Aurora 복제본이 자체 처리하는 SELECT 쿼리에 근거하여 캐시를 워밍업하기 때문입니다.

Aurora는 여러 동시 사용자 및 쿼리를 수반하는 트랜잭션 워크로드에서 최상의 성능을 발휘합니다. 충분한 로드를 촉진하여 최적의 성능을 얻을 수 있도록 보장하려면 멀티스레딩을 사용하는 벤치마크를 실행하거나 여러 개의 성능 테스트 인스턴스를 동시에 실행하는 벤치마크를 실행하십시오. 수백 또는 수천 건의 동시 클라이언트 스레드에서 성능을 측정하십시오. 프로덕션 환경에서 예상되는 동시 스레드의 수를 시뮬레이션하십시오. 더 많은 스레드에서 추가 스트레스 테스트를 수행하여 Aurora 확장성을 측정할 수도 있습니다.

12. Aurora 고가용성 실습

주요 Aurora 기능 중 다수는 고가용성을 포함합니다. 이러한 기능으로 자동 복제, 자동 장애 조치, 특정 시점으로 복원을 포함한 자동 백업, 클러스터에 DB 인스턴스를 추가할 수 있는 기능을 들 수 있습니다. 이러한 기능의 안전성과 안정성은 미션 크리티컬 애플리케이션에 중요합니다.

이러한 기능을 평가하려면 특정 사고 방식이 필요합니다. 성능 측정 등 앞서 한 활동에서 모든 것이 제대로 작동할 때 시스템이 어떤 성능을 보이는지 관찰하십시오. 고가용성 테스트 시 최악의 작동에 대해 충분히 생각해야 합니다. 드문 일이라 하더라도 다양한 종류의 장애를 고려해야 합니다. 시스템이 정확하고 빠르게 복구되게 하려면 의도적으로 문제를 도입해야 할 수 있습니다.

> **Tip**
>
> 개념 증명의 경우 동일한 AWS 인스턴스 클래스로 Aurora 클러스터에서 모든 DB 인스턴스를 설정합니다. 이렇게 하면 장애를 시뮬레이션하기 위해 DB 인스턴스를 오프라인으로 전환할 때 성능 및 확장성에 큰 변동이 없이 Aurora 가용성 기능을 시험해 볼 수 있습니다.

각 Aurora 클러스터에서 최소 두 개의 인스턴스를 사용하는 것이 좋습니다. Aurora 클러스터의 DB 인스턴스는 최대 3개의 가용 영역(AZ)에 걸쳐 존재할 수 있습니다. 최초 2개 또는 3개의 DB 인스턴스 각각을 다른 AZ에 배치하십시오. 더 큰 규모의 클러스터를 사용하려면 해당 AWS 리전의 모든 AZ에 DB 인스턴스를 분산하십시오. 이렇게 하면 내결함 능력이 향상됩니다. 한 가지 문제가 전체 AZ에 영향을 미친다 해도 Aurora는 다른 AZ에 있는 DB 인스턴스에 장애 조치를 취할 수 있습니다. 인스턴스가 3개 이상인 클러스터를 실행하는 경우 DB 인스턴스를 세 가지 AZ 모두에 균등하게 분산하십시오.

> **Tip**
>
> Aurora 클러스터용 스토리지는 DB 인스턴스에서 독립되어 있습니다. 각 Aurora 클러스터의 스토리지는 항상 세 가지 AZ에 걸쳐 있습니다.
> 고가용성 기능을 테스트할 때 항상 테스트 클러스터에서 동일한 용량을 지닌 DB 인스턴스를 사용하십시오. 이를 통해 DB 인스턴스 하나가 다른 인스턴스를 인계할 때마다 성능, 지연 시간 등이 예기치 않게 변경되는 것을 방지할 수 있습니다.

장애 조건을 시뮬레이션하여 고가용성 기능을 테스트하는 방법에 대해 알아보려면 오류 삽입 쿼리를 사용하여 Amazon Aurora MySQL 테스트 단원을 참조하십시오.

개념 증명 실습의 한 가지 목표는 DB 인스턴스의 이상적 개수와 이 DB 인스턴스의 최적 인스턴스 클래스를 알아내는 것입니다. 이를 위해서는 고가용성 및 성능에 대한 요구 사항을 균형 있게 조절해야 합니다.

Aurora의 경우 클러스터에 DB 인스턴스가 많을수록 고가용성의 이점이 더 큽니다. 더 많은 DB 인스턴스를 보유하면 읽기 집약적인 애플리케이션의 확장성도 향상됩니다. Aurora는 `SELECT` 쿼리에 대한 여러 연결을 읽기 전용 Aurora 복제본에 배포할 수 있습니다.

한편, DB 인스턴스의 수를 제한하면 기본 노드에서 나오는 복제 트래픽이 줄어듭니다. 복제 트래픽은 전체 성능 및 확장성의 다른 속성인 네트워크 대역폭을 소비합니다. 따라서 쓰기 집약적인 OLTP 애플리케이션에 대해서는 다수의 스몰 DB 인스턴스보다는 더 적은 수의 라지 DB 인스턴스를 보유하는 것이 더 낫습니다.

일반적인 Aurora 클러스터에서는 하나의 DB 인스턴스(기본 인스턴스)가 모든 DDL 및 DML 문을 처리합니다. 기타 DB 인스턴스(Aurora 복제본)는 SELECT 문만 처리합니다. 각 DB 인스턴스가 정확히 동일한 양의 작업을 하는 것은 아니지만 클러스터에 있는 모든 DB 인스턴스에 대해 동일한 인스턴스 클래스를 사용하는 것이 좋습니다. 이로써 장애가 발생하고 Aurora가 읽기 전용 DB 인스턴스 중 하나를 새로운 기본 인스턴스로 승격시키는 경우 기본 인스턴스의 용량은 이전과 동일합니다.

동일 클러스터에서 다양한 용량을 지닌 DB 인스턴스를 사용해야 하는 경우 DB 인스턴스에 대해 장애 조치 티어를 설정하십시오. 이러한 티어로 인해 Aurora 복제본이 장애 조치 메커니즘에 의해 승격되는 순서가 결정됩니다. 다른 것보다 훨씬 더 크거나 작은 DB 인스턴스를 더 낮은 장애 조치 티어에 배치하십시오. 이로써 승격과 관련해 마지막으로 선택됩니다.

특정 시점으로 자동 복원, 수동 스냅샷 및 복원, 클러스터 역추적 등 Aurora 데이터 복구 기능을 연습합니다. 적절한 경우 스냅샷을 다른 AWS 리전에 복사하고 다른 AWS 리전으로 복원하여 DR 시나리오를 모방합니다.

RTO(복원 시간 목표), RPO(복원 시점 목표) 및 지리적 중복성에 대한 조직의 요구 사항을 조사합니다. 대부분의 조직은 재해 복구라는 넓은 범주로 이 항목들을 그룹화합니다. 재해 복구 프로세스의 맥락에서 이 단원에 설명된 Aurora 고가용성 기능을 평가하여 RTO 및 RPO 요구 사항이 충족되게 하십시오.

13. 다음에 수행할 작업

개념 증명 프로세스를 성공적으로 종료하는 시점에 예상 워크로드에 근거하여 Aurora가 적합한 솔루션이라는 것을 확인합니다. 이전 프로세스에서는 Aurora가 실제 운영 환경에서 어떻게 작동하는지 확인하고 성공 기준과 비교 측정하였습니다.

Aurora를 이용해 데이터베이스 환경을 가동하기 시작한 후에는 더 세부적인 평가 단계로 나아감으로써 최종 마이그레이션 및 프로덕션 배포까지 완료할 수 있습니다. 상황에 따라 이러한 기타 단계는 개념 증명 프로세스에 포함될 수도 포함되지 않을 수도 있습니다. 마이그레이션 및 포트 활동에 대한 자세한 내용은 AWS 백서인 Aurora 마이그레이션 핸드북을 참조하세요.

또 다른 후속 단계에서는 워크로드에 타당하고 프로덕션 환경에서 보안 요구 사항을 충족하도록 설계된 보안 구성을 고려해 보십시오. Aurora 클러스터 마스터 사용자 자격 증명에 대한 액세스 권한을 보호하기 위해 어떤 제어를 실시할지 계획하십시오. 데이터베이스 사용자의 역할 및 책임을 정의하여 Aurora 클러스터에 저장된 데이터에 대한 액세스 권한을 제어하십시오. 애플리케이션, 스크립트 및 타사 도구 또는 서비스에 대한 데이터베이스 액세스 요구 사항을 고려하십시오. AWS Secrets Manager 및 AWS Identity and Access Management(IAM) 인증과 같은 AWS 서비스 및 기능에 대해 알아보세요.

이 시점에 이르면 Aurora를 이용한 벤치마크 테스트 실행의 절차 및 모범 사례를 이해해야 합니다. 성능 튜닝을 추가로 수행할 필요가 있다는 것을 알게 될 수도 있습니다. 자세한 내용은 Aurora DB 클러스

터의 성능 및 확장 관리, Amazon Aurora MySQL 성능 개선 사항, Amazon Aurora PostgreSQL의 성능 및 규모 조정 및 성능 개선 도우미를 통한 Amazon Aurora 모니터링 단원을 참조하십시오. 추가 튜닝을 수행하는 경우 개념 증명 중에 수집한 지표를 숙지해야 합니다. 다음 단계를 위해 구성 설정, 데이터베이스 엔진 및 데이터베이스 버전에 대한 선택 사항이 다른 새 클러스터를 생성할 수 있습니다. 또는 특정 사용 사례의 필요에 부합하는 특별한 종류의 Aurora 클러스터를 생성할 수도 있습니다.

예를 들어 하이브리드 트랜잭션/분석적 처리(HTAP) 애플리케이션을 위한 Aurora 병렬 쿼리 클러스터를 탐색할 수 있습니다. 광범위한 지리적 분산이 재해 복구에 중요한 경우 또는 지연 시간을 최소화하고 싶은 경우 Aurora 글로벌 데이터베이스를 탐색할 수 있습니다. 워크로드가 간헐적이거나 개발/테스트 시나리오에서 Aurora를 사용 중이라면 Aurora Serverless 클러스터를 탐색할 수 있습니다.

프로덕션 클러스터는 고용량의 수신 연결을 처리해야 할 수도 있습니다. 이러한 기법에 대해 알아보려면 AWS 백서인 Aurora MySQL 데이터베이스 관리자용 핸드북 – 연결 관리를 참조하세요.

개념 증명을 마친 후 사용 사례가 Aurora에 적합하지 않다고 판단되면 다른 AWS 서비스를 고려합니다.

- 순전히 분석적인 사용 사례의 경우 워크로드에는 OLAP 워크로드에 더 적합한 열 기반 스토리지 형식 및 기타 기능이 도움이 됩니다. 이러한 사용 사례에 대처할 수 있는 AWS 서비스는 다음과 같습니다.
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- Aurora와 이 서비스 중 한 개 이상을 조합하면 많은 워크로드에서 그 이점을 활용할 수 있습니다. 다음 기능을 이용해 이러한 서비스 간에 데이터를 이동할 수 있습니다.
  - AWS Glue
  - AWS DMS
  - Amazon Aurora 사용 설명서 에 설명된 Amazon S3에서 가져오기
  - Amazon Aurora 사용 설명서 에 설명된 Amazon S3로 내보내기
  - 그밖에 널리 사용되는 다수의 ETL 도구

Amazon Aurora의 보안

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 가장 보안에 민감한 조직의 요구 사항에 부합하도록 빌드된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS과 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS클라우드에서 AWS서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사원은 정기적으로 AWS규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Aurora(Aurora)에 적용되는 규정 준수 프로그램에 대해 알아보려면 AWS services in scope by compliance program을 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS서비스에 의해 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon Aurora 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 항목에서는 보안 및 규정 준수 목표를 충족하도록 Amazon Aurora를 구성하는 방법을 보여 줍니다. 또한 Amazon Aurora 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

DB 클러스터에서 Amazon Aurora 리소스 및 데이터베이스에 대한 액세스를 관리할 수 있습니다. 액세스 관리에 사용하는 방법은 사용자가 Amazon Aurora를 사용하여 수행해야 하는 작업 유형에 따라 다릅니다.

- 네트워크 액세스 제어를 최대한 강화할 목적으로 Amazon VPC 서비스에 따라 DB 클러스터를가상 프라이빗 클라우드(VPC)에서 실행합니다. DB 클러스터를 VPC에서 만드는 방법에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 섹션을 참조하세요.
- Amazon Aurora 리소스를 관리할 수 있는 사용자를 결정하는 권한을 할당하려면 AWS Identity and Access Management(IAM) 정책을 사용합니다. 예를 들면, IAM을 사용하여 DB 클러스터, 태그 리소스를 생성, 설명, 수정, 삭제하거나 보안 그룹을 수정할 수 있는 사용자를 결정할 수 있습니다.

  IAM 정책 예제는 Amazon Aurora 자격 증명 기반 정책 예 섹션을 참조하세요.
- 보안 그룹을 사용하여 어떤 IP 주소 또는 Amazon EC2 인스턴스가 DB 클러스터에 있는 데이터베이스에 연결할 수 있는지 제어합니다. DB 클러스터를 처음 생성하면, 방화벽이 연결된 보안 그룹에서 지정한 규칙을 통한 액세스 이외의 데이터베이스 액세스를 차단합니다.

- Aurora MySQL 또는 Aurora PostgreSQL을 실행하는 DB 클러스터와 함께 Secure Socket Layer(SSL) 또는 Transport Layer Security(TLS) 연결을 사용합니다. DB 클러스터에서 SSL/TLS를 사용하는 것에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.
- Amazon Aurora 암호화를 사용하여 DB 클러스터 및 저장 중인 스냅샷을 보호합니다. Amazon Aurora 암호화는 DB 클러스터를 호스팅하는 서버의 데이터를 업계 표준 AES-256 암호화 알고리즘을 사용하여 암호화합니다. 자세한 정보는 Amazon Aurora 리소스 암호화을 참조하세요.
- DB 엔진의 보안 기능을 사용하여 DB 클러스터에 있는 데이터베이스에 누가 로그인할 수 있는지 제어합니다. 이러한 보안 기능은 데이터베이스가 마치 로컬 네트워크에 있는 것처럼 실행됩니다.

  Aurora MySQL 사용 시 보안에 대한 자세한 내용은 Amazon Aurora MySQL를 사용한 보안 섹션을 참조하세요. Aurora PostgreSQL 사용 시 보안에 대한 자세한 내용은 Amazon Aurora PostgreSQL를 사용한 보안 섹션을 참조하세요.

Aurora는 관리형 데이터베이스 서비스인 Amazon Realtional Database Service(Amazon RDS)의 일부입니다. Amazon RDS는 클라우드에서 관계형 데이터베이스를 더 쉽게 설치, 운영 및 크기 조정할 수 있는 웹 서비스입니다. Amazon RDS에 익숙하지 않은 경우 Amazon RDS 사용 설명서를 참조하십시오.

Aurora에는 고성능 스토리지 하위시스템이 포함됩니다. MySQL 및 PostgreSQL과 호환되는 데이터베이스 엔진은 빠른 분산형 스토리지를 활용하도록 사용자 지정됩니다. Aurora는 또한 데이터베이스 구성 및 관리의 가장 어려운 측면 중 하나인 데이터베이스 클러스터링 및 복제를 자동화하고 표준화합니다.

Amazon RDS와 Aurora에서 모두 프로그래밍 방식으로 RDS API에 액세스할 뿐만 아니라 AWS CLI를 사용해 RDS API에 대화식으로 액세스할 수도 있습니다. 일부 RDS API 작업과 AWS CLI 명령은 Amazon RDS 및 Aurora에 모두 적용되는 반면 Amazon RDS 또는 Aurora에만 적용되는 작업이나 명령도 있습니다. RDS API 작업에 대한 자세한 내용은 Amazon RDS API 참조 단원을 참조하십시오. AWS CLI에 대한 자세한 내용은 Amazon RDS용 AWS Command Line Interface 참조를 참조하세요.

> **Note**
>
> 사용 사례에 따라 보안을 구성해야 합니다. 프로세스를 Amazon Aurora에서 관리하는 경우에는 보안 액세스를 구성할 필요 없습니다. 이러한 프로세스로는 백업 생성, 자동 장애 조치 등이 있습니다.

Amazon Aurora 리소스에 대한 액세스 관리와 DB 클러스터의 데이터베이스에 대한 액세스 관리에 대해서는 아래 주제를 참조하세요.

주제

Amazon Aurora을 사용한 데이터베이스 인증

Amazon Aurora는 데이터베이스 사용자를 인증하는 여러 가지 방법을 지원합니다.

암호 인증은 기본적으로 모든 DB 클러스터에 사용할 수 있습니다. Aurora MySQL 및 Aurora PostgreSQL의 경우, 같은 DB 클러스터에 대해 IAM 데이터베이스 인증과 Kerberos 인증을 둘 중 하나 또는 둘 다 추가할 수도 있습니다.

암호, Kerberos 및 IAM 데이터베이스 인증은 데이터베이스에 대해 서로 다른 인증 방법을 사용합니다. 따라서 특정 사용자는 하나의 인증 방법만 사용하여 데이터베이스에 로그인할 수 있습니다.

PostgreSQL의 경우, 특정 데이터베이스 사용자에 대해 다음 역할 설정 중 하나만 사용하세요.

- IAM 데이터베이스 인증을 사용하려면 `rds_iam` 역할을 사용자에게 할당합니다.
- Kerberos 인증을 사용하려면 `rds_ad` 역할을 사용자에게 할당합니다.
- 암호 인증을 사용하려면 `rds_iam` 또는 `rds_ad` 역할을 사용자에게 할당하지 않습니다.

`rds_iam` 및 `rds_ad` 역할을 직접적으로 또는 중첩된 액세스 권한 부여를 통해 간접적으로 PostgreSQL 데이터베이스 사용자에게 둘 다 할당하지 않도록 합니다. `rds_iam` 역할이 마스터 사용자에게 추가되면, IAM 인증이 암호 인증보다 우선하므로 마스터 사용자가 IAM 사용자로 로그인해야 합니다.

> **Important**
>
> 애플리케이션에서 직접 마스터 사용자를 사용하지 않는 것이 좋습니다. 대신에 애플리케이션에 필요한 최소 권한으로 생성한 데이터베이스 사용자를 사용하는 모범 사례를 준수하십시오.

주제

- 암호 인증
- IAM 데이터베이스 인증
- Kerberos 인증

암호 인증

암호 인증을 통해 데이터베이스는 모든 사용자 계정 관리를 수행합니다. `CREATE USER`와 같은 DB 엔진에서 암호를 지정하는 데 필요한 적절한 절이 있는 SQL 문을 사용하여 사용자를 생성합니다. 예를 들어 MySQL에서 명령문은 `CREATE USER` *##* `IDENTIFIED BY` *##*이지만 PostgreSQL에서 명령문은 `CREATE USER` *##* `WITH PASSWORD` *##*입니다.

암호 인증을 통해 데이터베이스는 사용자 계정을 제어하고 인증합니다. DB 엔진에 강력한 암호 관리 기능이 있는 경우 보안을 강화할 수 있습니다. 소규모 사용자 커뮤니티가 있는 경우 암호 인증을 사용하여 데이터베이스 인증을 쉽게 관리할 수 있습니다. 이 경우 일반 텍스트 암호가 생성되므로 AWS Secrets Manager과 통합하면 보안이 강화될 수 있습니다.

Secrets Manager를 Amazon Aurora와 함께 사용하는 것에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서에서 Creating a basic secret 및 Rotating secrets for supported Amazon RDS databases를 참조하세요. 사용자 지정 애플리케이션에서 프로그래밍 방식으로 비밀을 검색하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 비밀 값 검색을 참조하세요.

IAM 데이터베이스 인증

AWS Identity and Access Management(IAM) 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다. 이러한 인증 방식은 DB 클러스터에 연결할 때 암호를 사용할 필요가 없습니다. 대신에 인증 토큰을 사용합니다.

특정 DB 엔진의 가용성에 대한 정보를 포함하여 IAM 데이터베이스 인증에 대한 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요.

Kerberos 인증

Amazon Aurora 에서는 Kerberos 및 Microsoft Active Directory를 사용한 데이터베이스 사용자의 외부 인증을 지원합니다. Kerberos는 티켓과 대칭 키 암호화를 사용하여 네트워크를 통해 암호를 전송할 필요가 없는 네트워크 인증 프로토콜입니다. Kerberos는 Active Directory에 내장되어 있으며 데이터베이스와 같은 네트워크 리소스에 대해 사용자를 인증하도록 설계되었습니다.

Kerberos 및 Active Directory에 대한 Amazon Aurora 의 지원은 데이터베이스 사용자에게 Single Sign-On 및 중앙 집중식 인증의 이점을 제공합니다. 사용자 자격 증명을 Active Directory에 보관할 수 있습니다. Active Directory는 여러 DB 클러스터에 대한 자격 증명을 보관하고 관리할 수 있는 중앙 집중식 공간을 제공합니다.

자체 관리형 Active Directory의 자격 증명을 사용하려면 DB클러스터가 조인된 Microsoft Active Directory의 AWS Directory Service에 대한 신뢰 관계를 설정해야 합니다.

Aurora PostgreSQL 및 Aurora MySQL은 포리스트 전체 인증 또는 선택적 인증을 통해 단방향 및 양방향 포리스트 신뢰 관계를 지원합니다.

일부 시나리오에서는 외부 신뢰 관계를 통해 Kerberos 인증을 구성할 수 있습니다. 이를 위해서는 자체 관리형 Active Directory에 추가 설정이 필요합니다. 여기에는 Kerberos 포리스트 검색 순서가 포함되지만, 이에 국한되지는 않습니다.

Aurora는 Aurora MySQL 및 Aurora PostgreSQL DB 클러스터에 대한 Kerberos 인증을 지원합니다. 자세한 내용은 Aurora MySQL에 Kerberos 인증 사용 및 Aurora PostgreSQL과 함께 Kerberos 인증 사용를 참조하세요.

Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리

Amazon Aurora는 Secrets Manager와 통합되어 DB 클러스터의 마스터 사용자 암호를 관리합니다.

주제

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. 버전 및 Secrets Manager와 Amazon Aurora통합의 리전 가용성에 대한 자세한 내용은 Secrets Manager 통합을 지원하는 리전 및 Aurora DB 엔진을 참조하세요.

Secrets Manager와 Amazon Aurora 통합에 대한 제한 사항

다음 기능에서는 Secrets Manager를 사용한 마스터 사용자 암호 관리가 지원되지 않습니다.

- Amazon RDS 블루/그린 배포
- Aurora Global Database의 일부인 DB 클러스터
- Aurora Serverless v1 DB 클러스터
- Aurora 읽기 전용 복제본

AWS Secrets Manager를 통한 마스터 사용자 암호 관리 개요

AWS Secrets Manager를 사용하면 데이터베이스 암호를 포함한 하드 코딩된 보안 인증 정보를 Secrets Manager에서 프로그래밍 방식으로 비밀을 검색하게 하는 API 호출로 바꿀 수 있습니다. Secrets Manager 사용에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서를 참조하세요.

Secrets Manager에 데이터베이스 암호를 저장하면 AWS 계정에서 요금을 부과합니다. 요금에 대한 자세한 내용은 AWS Secrets Manager 요금을 참조하십시오.

다음 작업 중 하나를 수행할 때 Aurora가 Secrets Manager에서 Amazon Aurora DB 클러스터의 마스터 사용자 암호를 관리하도록 지정할 수 있습니다.

- DB 클러스터 만들기
- DB 클러스터를 수정하는 방법
- Amazon S3에서 DB 클러스터 복원(Aurora MySQL만 해당)

Aurora가 Secrets Manager에서 마스터 사용자 암호를 관리하도록 지정하면 Aurora가 암호를 생성하여 Secrets Manager에 저장합니다. 암호와 직접 상호 작용하여 마스터 사용자의 보안 인증 정보를 검색할 수 있습니다. 고객 관리 키를 지정하여 암호를 암호화하거나 Secrets Manager에서 제공하는 KMS 키를 사용할 수도 있습니다.

Aurora는 비밀 설정을 관리하고 기본적으로 7일마다 비밀을 교체합니다. 교체 일정 같은 일부 설정을 수정할 수 있습니다. Secrets Manager에서 암호를 관리하는 DB 클러스터를 삭제하면 암호 및 관련 메타데이터도 삭제됩니다.

비밀의 보안 인증 정보를 사용하여 DB 클러스터에 연결하려면 Secrets Manager에서 비밀을 검색하면 됩니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager에서 비밀 검색 및 AWS Secrets Manager 비밀에 있는 보안 인증 정보를 사용하여 SQL 데이터베이스에 연결을 참조하세요.

Secrets Manager를 통한 마스터 사용자 암호 관리의 이점

Secrets Manager를 사용하여 Aurora 마스터 사용자 암호를 관리하면 다음과 같은 이점이 있습니다.

- Aurora가 데이터베이스 보안 인증 정보를 자동으로 생성합니다.
- Aurora가 데이터베이스 보안 인증 정보를 AWS Secrets Manager에 자동으로 저장하고 관리합니다.
- 애플리케이션을 변경할 필요 없이 Aurora가 정기적으로 데이터베이스 보안 인증 정보를 교체합니다.
- Secrets Manager가 사용자 액세스 및 일반 텍스트 보기로부터 데이터베이스 보안 인증 정보를 보호합니다.
- Secrets Manager를 사용하면 데이터베이스 연결을 위한 비밀의 데이터베이스 보안 인증 정보를 검색할 수 있습니다.

- Secrets Manager에서 IAM을 사용하여 비밀의 데이터베이스 보안 인증 정보에 대한 액세스를 세밀하게 제어할 수 있습니다.
- 필요에 따라 다른 KMS 키를 사용하여 데이터베이스 암호화와 보안 인증 정보 암호화를 분리할 수 있습니다.
- 데이터베이스 보안 인증 정보를 수동으로 관리하고 교체하지 않아도 됩니다.
- AWS CloudTrail 및 Amazon CloudWatch를 사용하여 데이터베이스 보안 인증 정보를 쉽게 모니터링할 수 있습니다.

Secrets Manager의 이점에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서를 참조하세요.

Secrets Manager 통합에 필요한 권한

사용자는 Secrets Manager 통합과 관련된 작업을 수행하는 데 필요한 권한이 있어야 합니다. 사용자에게 필요한 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 부여하는 IAM 정책을 생성할 수 있습니다. 그런 다음 해당 권한이 필요한 IAM 권한 세트 또는 역할에 이러한 정책을 연결할 수 있습니다. 자세한 내용은 Amazon Aurora의 자격 증명 및 액세스 관리 섹션을 참조하세요.

생성, 수정 또는 복원 작업의 경우 Aurora가 Secrets Manager에서 마스터 사용자 암호를 관리하도록 지정하는 사용자는 다음 작업을 수행할 권한이 있어야 합니다.

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

`MasterUserSecretKmsKeyId`에 대한 고객 관리형 키에 액세스하고 `aws/secretsmanager`를 설명하려면 `kms:DescribeKey` 권한이 필요합니다.

생성, 수정 또는 복원 작업의 경우 Secrets Manager에서 비밀을 암호화하는 고객 관리형 키를 지정하는 사용자는 다음 작업을 수행할 권한이 있어야 합니다.

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

수정 작업의 경우 Secrets Manager에서 마스터 사용자 암호를 교체하는 사용자는 다음 작업을 수행할 권한이 있어야 합니다.

- `secretsmanager:RotateSecret`

AWS Secrets Manager 마스터 사용자 암호의 Aurora 관리 적용

IAM 조건 키를 사용하여 AWS Secrets Manager에서 마스터 사용자 암호의 Aurora 관리를 적용할 수 있습니다. 다음 정책은 Secrets Manager에서 Aurora가 마스터 사용자 암호를 관리하지 않는 한 사용자가 DB 인스턴스 또는 DB 클러스터를 생성 또는 복원하거나 하는 것을 허용하지 않습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> 이 정책은 생성 시 AWS Secrets Manager에서 암호 관리를 적용합니다. 그러나 클러스터를 수정하여 여전히 Secrets Manager 통합을 비활성화하고 마스터 암호를 수동으로 설정할 수 있습니다.
>
> 이를 방지하려면 정책의 Action 블록에 `rds:ModifyDBInstance`, `rds:ModifyDBCluster`를 포함하세요. 이렇게 하면 사용자가 Secrets Manager 통합이 활성화되지 않은 기존 클러스터에 추가 수정 사항을 적용할 수 없습니다.

IAM 정책에서 조건 키 사용에 관한 자세한 내용은 Aurora의 정책 조건 키 및 정책 예: 조건 키 사용을 참조하세요.

Secrets Manager를 사용하여 DB 클러스터의 마스터 사용자 암호 관리

다음 작업을 수행할 때 Secrets Manager에서 마스터 사용자 암호의 Aurora 관리를 구성할 수 있습니다.

- Amazon Aurora DB 클러스터 생성
- Amazon Aurora DB 클러스터 수정
- 외부 MySQL 데이터베이스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션

콘솔, AWS CLI 또는 RDS API를 사용하여 이러한 작업을 수행할 수 있습니다.

콘솔

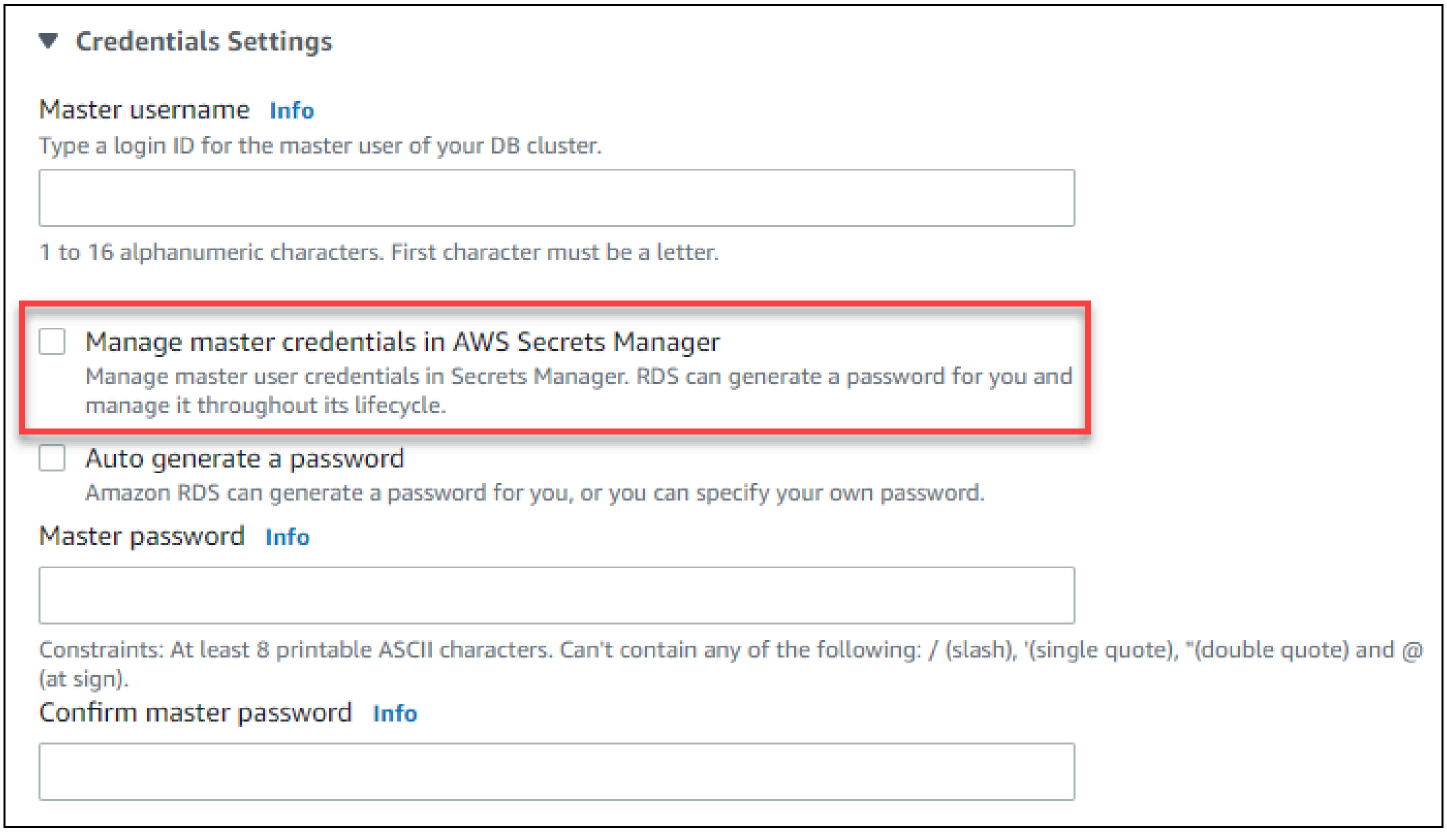
다음 지침에 따라 RDS 콘솔을 사용하여 DB 클러스터를 생성하거나 수정합니다.

- DB 클러스터 생성
- DB 클러스터에서 DB 인스턴스 수정

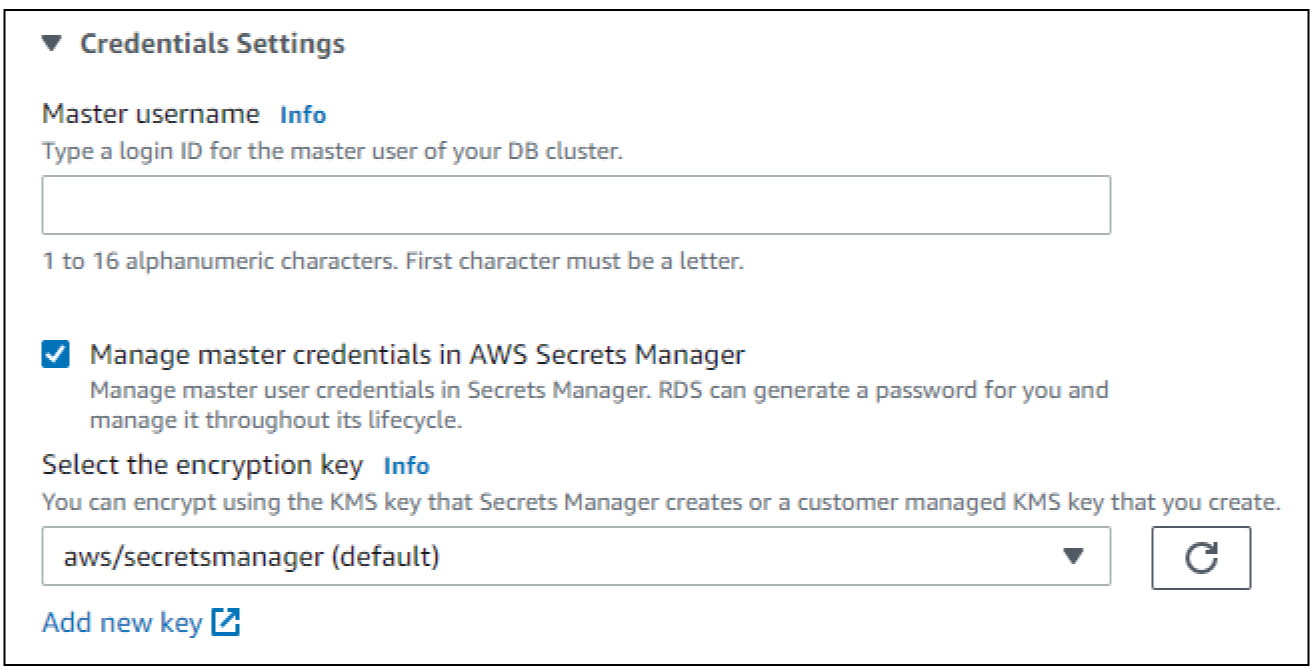
  RDS 콘솔에서 DB 인스턴스를 수정하여 전체 DB 클러스터의 마스터 사용자 암호 관리 설정을 지정할 수 있습니다.
- Amazon S3 버킷에서 Amazon Aurora MySQL DB 클러스터 복원

RDS 콘솔을 사용하여 이러한 작업 중 하나를 수행하는 경우 Secrets Manager에서 Aurora가 마스터 사용자 암호를 관리하도록 지정할 수 있습니다. DB 클러스터를 생성 또는 복원할 때 이를 수행하려면 Credential settings(보안 인증 정보 설정)에서 Manage master credentials in AWS Secrets Manager를 선택합니다. DB 클러스터를 수정할 경우 Settings(설정)에서 Manage master credentials in AWS Secrets Manager를 선택합니다.

다음 이미지는 DB 클러스터를 생성 또는 복원할 때의 Manage master credentials in AWS Secrets Manager 설정의 예입니다.



이 옵션을 선택하면 Aurora가 Secrets Manager에서 마스터 사용자 암호를 생성하고 수명 주기 동안 이를 관리합니다.



Secrets Manager가 제공하는 KMS 키 또는 사용자가 생성한 고객 관리 키를 사용하여 비밀을 암호화하도록 선택할 수 있습니다. Aurora가 DB 클러스터의 데이터베이스 보안 인증 정보를 관리한 후에는 비밀을 암호화하는 데 사용되는 KMS 키를 변경할 수 없습니다.

요구 사항에 맞는 다른 설정을 선택할 수 있습니다.

DB 클러스터를 생성할 때 사용 가능한 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정을 참조하세요. DB 클러스터를 수정할 때 사용 가능한 설정에 대한 자세한 내용은 Amazon Aurora에 대한 설정을 참조하세요.

AWS CLI

Aurora가 Secrets Manager에서 마스터 사용자 암호를 관리하도록 지정하려면 다음 명령 중 하나에서 `--manage-master-user-password` 옵션을 지정하세요.

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

이러한 명령에서 `--manage-master-user-password` 옵션을 지정하면 Aurora가 Secrets Manager에서 마스터 사용자 암호를 생성하고 해당 수명 주기 동안 이를 관리합니다.

비밀을 암호화하려면 고객 관리형 키를 지정하거나 Secrets Manager에서 제공하는 기본 KMS 키를 사용하면 됩니다. 고객 관리형 키를 지정하려면 `--master-user-secret-kms-key-id` 옵션을 사용하세요. AWS KMS 키 식별자는 KMS 키의 키 ARN, 키 ID, 별칭 ARN 또는 별칭 이름입니다. 다른 AWS 계정에서 KMS 키를 사용하려면 키 ARN 또는 별칭 ARN을 지정하세요. Aurora가 DB 클러스터의 데이터베이스 보안 인증 정보를 관리한 후에는 비밀을 암호화하는 데 사용되는 KMS 키를 변경할 수 없습니다.

요구 사항에 맞는 다른 설정을 선택할 수 있습니다.

DB 클러스터를 생성할 때 사용 가능한 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정을 참조하세요. DB 클러스터를 수정할 때 사용 가능한 설정에 대한 자세한 내용은 Amazon Aurora에 대한 설정을 참조하세요.

이 예에서는 DB 클러스터를 생성하고 Aurora가 Secrets Manager에서 암호를 관리하도록 지정합니다. 비밀은 Secrets Manager에서 제공하는 KMS 키를 사용하여 암호화됩니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --master-username admin \
    --manage-master-user-password
```

Windows의 경우:

```
aws rds create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql ^
    --engine-version 8.0 ^
    --master-username admin ^
    --manage-master-user-password
```

RDS API

Aurora가 Secrets Manager에서 마스터 사용자 암호를 관리하도록 지정하려면 다음 작업 중 하나에서 `ManageMasterUserPassword` 파라미터를 `true`로 설정하세요.

- CreateDBCluster
- ModifyDBCluster
- RestoreDBClusterFromS3

이러한 명령에서 `ManageMasterUserPassword` 파라미터를 `true`로 설정하면 Aurora가 Secrets Manager에서 마스터 사용자 암호를 생성하고 해당 수명 주기 동안 이를 관리합니다.

비밀을 암호화하려면 고객 관리형 키를 지정하거나 Secrets Manager에서 제공하는 기본 KMS 키를 사용하면 됩니다. `MasterUserSecretKmsKeyId` 파라미터를 사용하여 고객 관리형 키를 지정합니다. AWS KMS 키 식별자는 KMS 키의 키 ARN, 키 ID, 별칭 ARN 또는 별칭 이름입니다. 다른 AWS 계정에서 KMS 키를 사용하려면 키 ARN 또는 별칭 ARN을 지정하세요. Aurora가 DB 클러스터의 데이터베이스 보안 인증 정보를 관리한 후에는 비밀을 암호화하는 데 사용되는 KMS 키를 변경할 수 없습니다.

DB 클러스터의 마스터 사용자 암호 비밀 교체

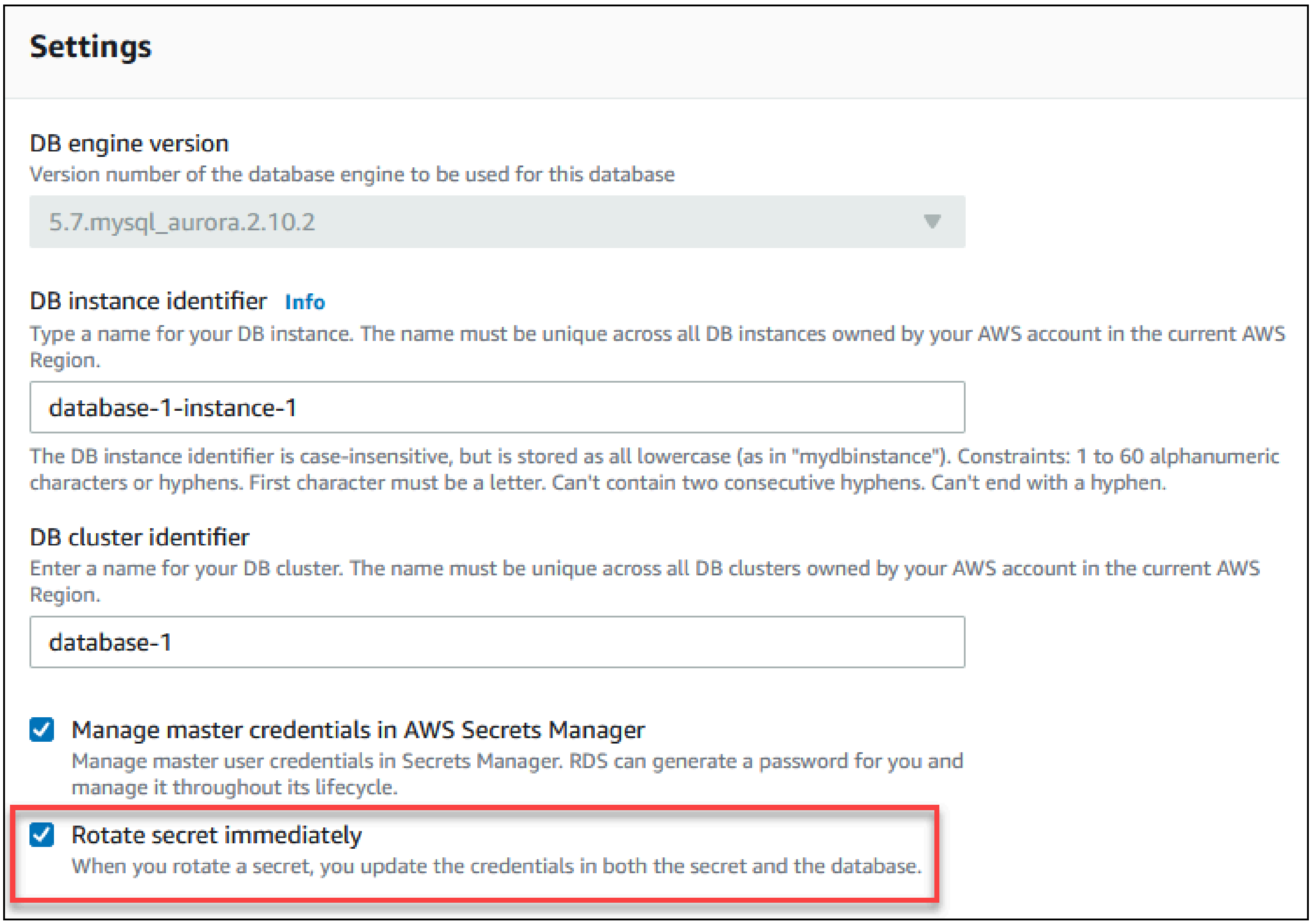
Aurora가 마스터 사용자 암호 비밀을 교체하면 Secrets Manager는 기존 비밀의 새 비밀 버전을 생성합니다. 새 버전의 비밀에는 새 마스터 사용자 암호가 포함됩니다. Aurora는 새 비밀 버전의 암호와 일치하도록 DB 클러스터의 마스터 사용자 암호를 변경합니다.

예약된 교체를 기다리지 않고 비밀을 즉시 교체할 수 있습니다. Secrets Manager에서 마스터 사용자 암호 비밀을 교체하려면 DB 클러스터를 수정하세요. DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정을 참조하세요.

RDS 콘솔, AWS CLI 또는 RDS API를 사용하여 마스터 사용자 암호 비밀을 즉시 교체할 수 있습니다. 새 암호는 항상 28자이며 하나 이상의 대문자와 소문자, 하나의 숫자, 하나의 구두점을 포함합니다.

콘솔

RDS 콘솔을 사용하여 마스터 사용자 암호 비밀을 교체하려면 DB 클러스터를 수정하고 Settings(설정)에서 Rotate secret immediately(암호 즉시 교체)를 선택합니다.



RDS 콘솔을 사용하여 DB 클러스터를 수정하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정의 지침에 따르세요. 확인 페이지에서 Apply immediately(즉시 적용)를 선택해야 합니다.

AWS CLI

AWS CLI를 사용하여 마스터 사용자 암호를 교체하려면 modify-db-cluster 명령을 사용하고 `--rotate-master-user-password` 옵션을 지정합니다. 마스터 암호를 교체할 때 `--apply-immediately` 옵션을 지정해야 합니다.

이 예에서는 마스터 사용자 암호 비밀을 교체합니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

Windows의 경우:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

ModifyDBCluster 작업을 사용하고 `RotateMasterUserPassword` 파라미터를 `true`로 설정하여 마스터 사용자 암호 비밀을 교체할 수 있습니다. 마스터 암호를 교체할 때 `ApplyImmediately` 파라미터를 `true`로 설정해야 합니다.

DB 클러스터의 비밀에 대한 세부 정보 보기

콘솔(https://console.aws.amazon.com/secretsmanager/) 또는 AWS CLI(get-secret-value Secrets Manager 명령)을 사용하여 비밀을 검색할 수 있습니다.

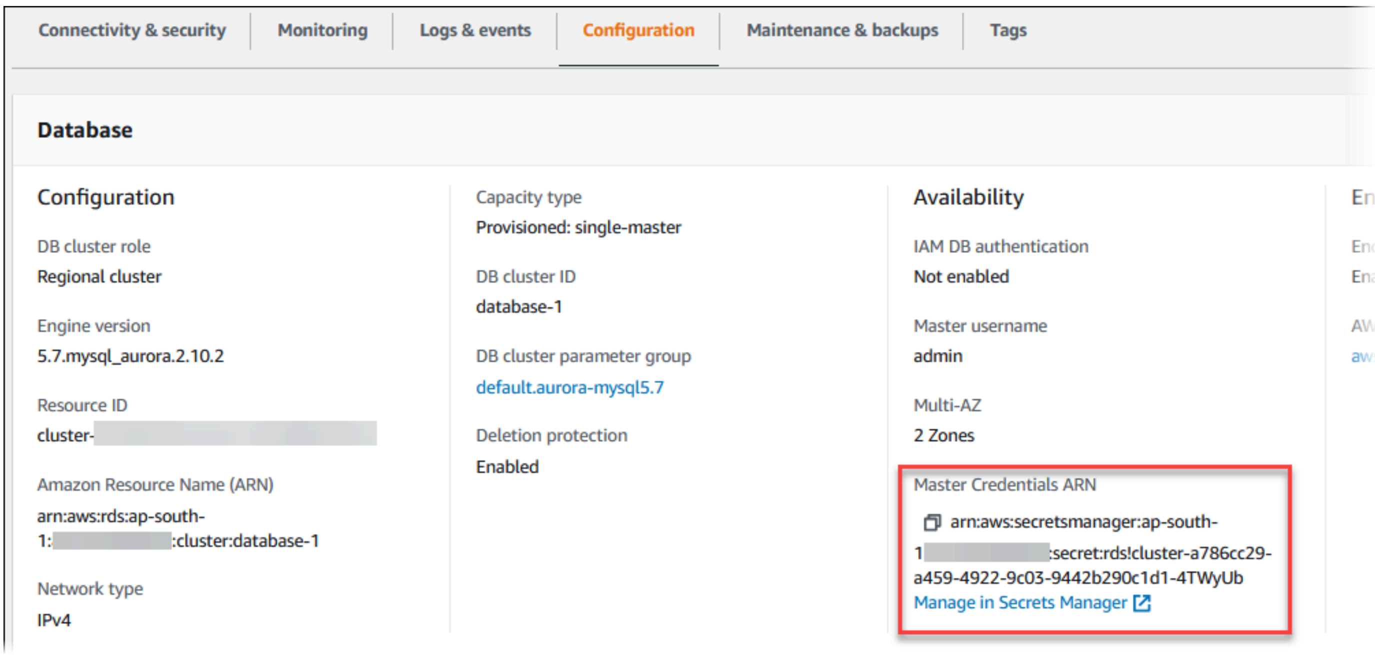
RDS 콘솔, AWS CLI 또는 RDS API를 사용하여 Secrets Manager에서 Aurora가 관리하는 비밀의 Amazon 리소스 이름(ARN)을 찾을 수 있습니다.

콘솔

Secrets Manager에서 Aurora가 관리하는 비밀에 대한 세부 정보 보기

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 그런 다음 세부 정보를 표시할 DB 클러스터 이름을 선택합니다.
4. Configuration(구성) 탭을 선택합니다.

   마스터 보안 인증 정보 ARN에서 비밀 ARN을 볼 수 있습니다.

   Manage in Secrets Manager(Secrets Manager에서 관리) 링크를 클릭하면 Secrets Manager 콘솔에서 비밀을 보고 관리할 수 있습니다.

AWS CLI

RDS AWS CLI describe-db-clusters 명령을 사용하면 Secrets Manager에서 Aurora가 관리하는 비밀에 대한 다음 정보를 찾을 수 있습니다.

- `SecretArn` - 비밀의 ARN
- `SecretStatus` - 비밀의 상태

  가능한 상태 값에는 다음이 포함됩니다.

  - `creating` - 비밀이 생성 중입니다.
  - `active` - 비밀의 일반적 사용 및 교체가 가능합니다.
  - `rotating` - 비밀이 교체 중입니다.
  - `impaired` - 비밀을 데이터베이스 보안 인증 정보에 액세스하는 데 사용할 수 있지만 교체할 수는 없습니다. 예를 들어 권한이 변경되어 RDS가 더 이상 비밀 또는 비밀의 KMS 키에 액세스할 수 없는 경우 비밀이 이 상태가 될 수 있습니다.

    암호가 이 상태인 경우 상태를 초래한 조건을 수정할 수 있습니다. 상태를 초래한 조건을 수정하면 다음 교체까지 상태가 `impaired`로 유지됩니다. 또는 DB 클러스터를 수정하여 데이터베이스 보안 인증 정보 자동 관리를 해제한 다음 DB 클러스터를 다시 수정하여 데이터베이스 보안 인증 정보 자동 관리를 켤 수 있습니다. DB 클러스터를 수정하려면 modify-db-cluster 명령에서 `--manage-master-user-password` 옵션을 사용하세요.
- `KmsKeyId` - 비밀을 암호화하는 데 사용된 KMS 키의 ARN입니다.

특정 DB 클러스터의 출력을 표시하려면 `--db-cluster-identifier` 옵션을 지정합니다. 이 예제는 DB 클러스터가 사용하는 비밀의 출력을 보여 줍니다.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

다음 샘플은 비밀 출력을 보여 줍니다.

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

보안 ARN이 있으면 get-secret-value Secrets Manager CLI 명령을 사용하여 비밀에 대한 세부 정보를 볼 수 있습니다.

이 예제는 이전 샘플 출력의 비밀에 대한 세부 정보를 보여 줍니다.

Example

대상 LinuxmacOS, 또는Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

Windows의 경우:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

DescribedBClusters RDS 작업을 사용하고 `DBClusterIdentifier` 파라미터를 DB 클러스터 식별자로 설정하여 Secrets Manager에서 Aurora가 관리하는 비밀의 ARN, 상태, KMS 키를 볼 수 있습니다. 비밀에 대한 세부 정보가 출력에 포함됩니다.

비밀 ARN이 있으면 GetSecretValue Secrets Manager 작업을 사용하여 비밀에 대한 세부 정보를 볼 수 있습니다.

Amazon RDS의 데이터 보호

AWS 공동 책임 모델은 Amazon Relational Database Service에서 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS은 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정 자격 증명을 보호하고 AWS IAM Identity Center 또는 AWS Identity and Access Management(IAM)를 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정하세요. AWS 활동 캡처에 CloudTrail 추적을 사용하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.

- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용하세요.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-3 검증된 암호화 모듈이 필요한 경우, FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Amazon RDS 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버로 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함해서는 안 됩니다.

주제

암호화를 사용하여 데이터 보호

데이터베이스 리소스에 대한 암호화를 활성화할 수 있습니다. 또한 DB 클러스터에 대한 연결도 암호화가 가능합니다.

주제

Amazon Aurora 리소스 암호화

Amazon Aurora는 Amazon Aurora DB 클러스터를 암호화할 수 있습니다. 유휴 시 암호화되는 데이터로는 DB 클러스터에 대한 기본 스토리지, 자동 백업 파일, 읽기 전용 복제본 및 스냅샷이 포함됩니다.

Amazon Aurora 암호화된 DB 클러스터는 Amazon Aurora DB 클러스터를 호스팅하는 서버의 데이터를 업계 표준 AES-256 암호화 알고리즘을 사용하여 암호화합니다. 데이터가 암호화된 이후 Amazon Aurora가 성능에 미치는 영향을 최소화한 상태에서 데이터 액세스 및 암호 해독의 인증을 투명하게 처리합니다. 암호화를 사용하도록 데이터베이스 클라이언트 애플리케이션을 수정하지 않아도 됩니다.

> **Note**
>
> 암호화/비암호화 DB 클러스터의 경우에는 AWS 리전 간 복제에서도 원본과 읽기 전용 복제본 사이에 전송되는 데이터가 암호화됩니다.

주제

Amazon Aurora 리소스 암호화 개요

Amazon Aurora 암호화된 DB 클러스터는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지해 추가 계층의 데이터 보호를 제공합니다. 클라우드에 배포된 애플리케이션의 데이터 보호를 강화하고 저장된 데이터 암호화를 위한 규정 준수 요구 사항을 만족하기 위해 Amazon Aurora 암호화를 사용할 수 있습니다. Amazon Aurora 암호화된 DB 클러스터의 경우 모든 DB 인스턴스, 로그, 백업 및 스냅샷이 암호화됩니다. 암호화의 가용성 및 제한에 대한 자세한 내용은 Amazon Aurora 암호화 가용성 및 Amazon Aurora 암호화된 DB 클러스터의 제한 섹션을 참조하세요.

Amazon Aurora는 AWS Key Management Service 키를 사용하여 이러한 리소스를 암호화합니다. AWS KMS는 안전하고 가용성이 높은 하드웨어와 소프트웨어를 결합하여 클라우드에 맞게 규모가 조정된 키 관리 시스템을 제공합니다. AWS 관리형 키을(를) 사용하거나 고객 관리형 키를 생성할 수 있습니다.

암호화된 DB 클러스터를 생성할 때 Amazon Aurora에 사용할 고객 관리형 키 또는 AWS 관리형 키을(를) 선택하여 DB 클러스터를 암호화할 수 있습니다. 고객 관리형 키의 키 식별자를 지정하지 않으면 Amazon Aurora는 새 DB 클러스터에 AWS 관리형 키을(를) 사용합니다. Amazon Aurora는 AWS 계정

에 대해 Amazon Aurora용 AWS 관리형 키을(를) 생성합니다. AWS 계정에 각 AWS 리전의 Amazon Aurora에 대한 다른 AWS 관리형 키이(가) 있습니다.

Amazon Aurora 리소스 암호화 및 복호화에 사용되는 고객 관리형 키를 관리하려면 AWS Key Management Service(AWS KMS)를 사용합니다.

AWS KMS를 사용하면 고객 관리형 키를 생성하고 이 키의 사용을 제어하는 정책을 정의할 수 있습니다. AWS KMS는 CloudTrail을 지원하므로 KMS 키 사용을 감사하여 고객 관리형 키가 적절하게 사용되고 있는지 확인할 수 있습니다. 고객 관리형 키는 Amazon Aurora를 비롯해 Amazon S3, Amazon EBS, Amazon Redshift 등 지원되는 AWS 서비스에서 사용할 수 있습니다. AWS KMS와 통합되는 서비스 목록은 AWS 서비스 통합을 참조하세요. KMS 키 사용에 대한 몇 가지 고려 사항:

- 암호화된 DB 인스턴스를 생성한 후에는 해당 DB 인스턴스에서 사용된 KMS 키를 변경할 수 없습니다. 따라서 암호화된 DB 인스턴스를 생성하기 전에 KMS 키 요구 사항을 결정해야 합니다.

  DB 클러스터의 암호화 키를 변경해야 하는 경우 클러스터의 수동 스냅샷을 생성하고 스냅샷을 복사하는 동안 암호화를 활성화합니다. 자세한 내용은 re:Post 지식 센터 문서를 참조하세요.
- 암호화된 스냅샷을 복사하는 경우 소스 스냅샷을 암호화하는 데 사용된 것과 다른 KMS 키를 사용하여 대상 스냅샷을 암호화할 수 있습니다.
- 스냅샷을 공유한 AWS 계정의 AWS 관리형 키을(를) 사용하여 암호화된 스냅샷은 공유할 수 없습니다.
- DB 클러스터의 각 DB 인스턴스는 DB 클러스터와 동일한 KMS 키를 사용하여 암호화됩니다.
- Amazon Aurora 암호화된 클러스터의 읽기 전용 복제본을 암호화할 수도 있습니다.

> ⚠ Important
>
> Amazon Aurora는 KMS 키를 비활성화하면 DB 클러스터의 KMS 키에 대한 액세스 권한을 잃을 수 있습니다. 이 경우 암호화된 DB 클러스터는 `inaccessible-encryption-credentials-recoverable` 상태입니다. DB 클러스터는 7일 동안 이 상태로 유지되며, 이 기간 동안 인스턴스가 중지됩니다. 이 기간 동안 DB 클러스터에 대한 API 호출은 성공하지 못할 수 있습니다. DB 클러스터를 복구하려면 KMS 키를 활성화하고 이 DB 클러스터를 다시 시작하세요. AWS Management Console, AWS CLI 또는 RDS API에서 KMS 키를 활성화합니다. AWS CLI 명령 start-db-cluster 또는 AWS Management Console을 사용하여 DB 클러스터를 다시 시작합니다.
>
> `inaccessible-encryption-credentials-recoverable` 상태는 중지할 수 있는 DB 클러스터에만 적용됩니다. 리전 간 읽기 전용 복제본이 있는 클러스터와 같이 중지할 수 없는 인

스턴스에는이 복구 가능 상태를 적용할 수 없습니다. 자세한 내용은 the section called “제한 사항” 섹션을 참조하세요.

7일 이내에 DB 클러스터가 복구되지 않으면 터미널 `inaccessible-encryption-credentials` 상태가 됩니다. 이 상태에서는 DB 클러스터를 더 이상 사용할 수 없으며 백업에서 DB 클러스터를 복원할 수만 있습니다. 데이터베이스에서 암호화된 데이터가 손실되지 않도록 보호하려면 암호화된 DB 클러스터에 대해 항상 백업을 활성화하는 것이 좋습니다.

DB 클러스터를 생성하는 동안 Aurora는 호출 주체가 KMS 키에 대한 액세스 권한이 있는지 확인하고 DB 클러스터의 전체 수명 기간 동안 사용하는 KMS 키에서 부여를 생성합니다. MS 키에 대한 호출 주체의 액세스 권한을 취소해도 실행 중인 데이터베이스에는 영향을 미치지 않습니다. 스냅샷을 다른 계정으로 복사하는 등 계정 간 시나리오에서 KMS 키를 사용하는 경우, KMS 키를 다른 계정과 공유해야 합니다. 다른 KMS 키를 지정하지 않고 스냅샷에서 DB 클러스터를 만들면 새 클러스터는 소스 계정의 KMS 키를 사용합니다. DB 클러스터를 생성한 후 키에 대한 액세스 권한을 취소해도 클러스터에는 영향을 미치지 않습니다. 그러나 키를 비활성화하면 해당 키로 암호화된 모든 DB 클러스터에 영향을 미칩니다. 이를 방지하려면 스냅샷 복사 작업 중에 다른 키를 지정하세요.

KMS 키에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS KMS keys 섹션 및 AWS KMS key 관리 섹션을 참조하세요.

Amazon Aurora DB 클러스터 암호화

새로운 DB 클러스터를 암호화하려면 콘솔에서 암호화 활성화(Enable encryption)를 선택합니다. DB 클러스터 생성에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.

create-db-cluster AWS CLI 명령을 사용하여 암호화된 DB 클러스터를 생성할 경우 `--storage-encrypted` 파라미터를 설정합니다. CreateDBCluster API 작업을 사용할 경우 `StorageEncrypted` 파라미터를 true로 설정하십시오.

암호화된 DB 클러스터를 생성한 후에는 해당 DB 클러스터에서 사용된 KMS 키를 변경할 수 없습니다. 따라서 암호화된 DB 클러스터를 생성하기 전에 KMS 키 요구 사항을 결정해야 합니다.

AWS CLI `create-db-cluster` 명령을 사용하여 고객 관리형 키로 암호화된 DB 클러스터를 생성하는 경우 `--kms-key-id` 파라미터를 KMS 키의 키 식별자로 설정합니다. Amazon RDS API `CreateDBInstance` 작업을 사용하는 경우 `KmsKeyId` 파라미터를 KMS 키의 키 식별자로 설정합니다. 다른 AWS 계정에서 고객 관리형 키를 사용하려면 키 ARN 또는 별칭 ARN을 지정합니다.

DB 클러스터에 대해 암호화가 켜져 있는지 확인

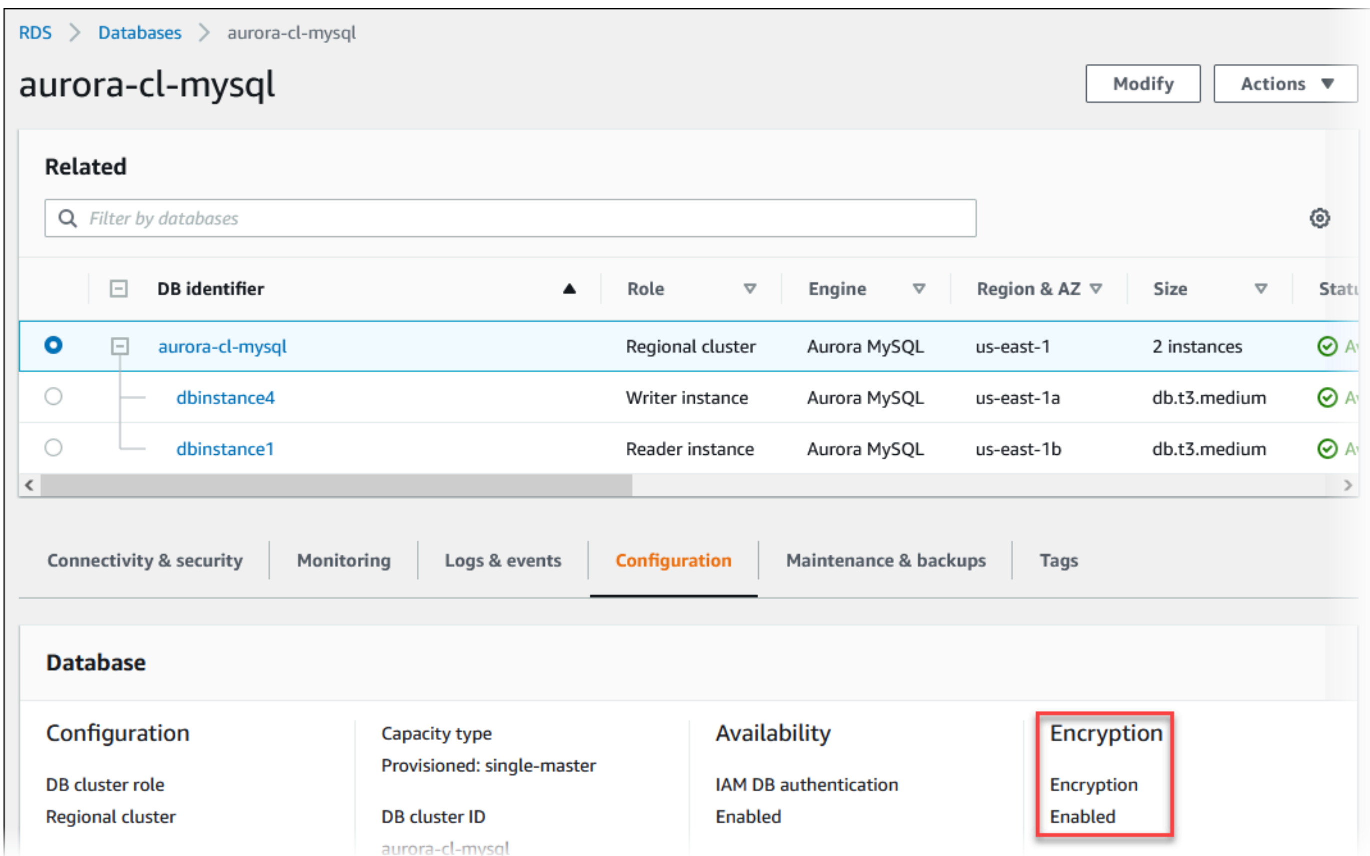
AWS Management Console, AWS CLI 또는 RDS API를 사용하여 DB 클러스터에 대해 저장 데이터 암호화가 켜져 있는지 확인합니다.

콘솔

DB 클러스터에 대해 저장 데이터 암호화가 켜져 있는지 확인하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 세부 정보를 보기 위해 확인하려는 DB 클러스터의 이름을 선택합니다.
4. 구성(Configuration) 탭을 선택하고 암호화(Encryption) 값을 확인합니다.

   그러면 활성화(Enabled) 또는 비활성화(Not enabled) 중 하나가 표시됩니다.



AWS CLI

AWS CLI를 통해 DB 클러스터에 대해 저장 데이터 암호화가 켜져 있는지 확인하려면 다음 옵션을 사용하여 describe-db-clusters 명령을 호출합니다.

- `--db-cluster-identifier` – DB 클러스터의 이름입니다.

다음 예에서는 쿼리를 사용하여 `mydb` DB 클러스터의 저장 데이터 암호화에 대해 `TRUE` 또는 `FALSE` 중 하나를 반환합니다.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

Amazon RDS API를 통해 DB 클러스터에 대해 저장 데이터 암호화가 켜져 있는지 확인하려면 다음 파라미터를 사용하여 DescribeDBClusters를 호출합니다.

- `DBClusterIdentifier` – DB 클러스터의 이름입니다.

Amazon Aurora 암호화 가용성

Amazon Aurora 암호화는 현재 모든 데이터베이스 엔진과 스토리지 유형에 사용할 수 있습니다.

> **Note**
> Amazon Aurora 암호화는 db.t2.micro DB 인스턴스 클래스에서는 사용 가능하지 않습니다.

전송 중 암호화

물리적 계층에서의 암호화

AWS 글로벌 네트워크를 통해 AWS 리전으로 이동하는 모든 데이터는 AWS 보안 시설에서 나가기 전에 물리적 계층에서 자동으로 암호화됩니다. AZ 간에 전송되는 모든 트래픽은 암호화됩니다. 이 섹션에 나열된 암호화 계층을 비롯한 추가 암호화 계층은 추가적인 보호 기능을 제공할 수 있습니다.

Amazon VPC 피어링 및 전송 게이트웨이 교차 리전 피어링에서 제공하는 암호화

Amazon VPC 및 Transit Gateway 피어링을 사용하는 모든 리전 간 트래픽은 리전을 종료하면 자동으로 대량 암호화됩니다. AWS 보안 시설에서 나가기 전에 모든 트래픽에 대한 추가 암호화 계층이 물리적 계층에서 자동으로 제공됩니다.

인스턴스 간 암호화

AWS는 모든 유형의 DB 인스턴스 간 보안 프라이빗 연결을 제공합니다. 또한 일부 인스턴스 유형은 기본 Nitro 시스템 하드웨어의 오프로드 기능을 사용하여 인스턴스 간 전송 중 트래픽을 자동으로 암호화합니다. 이 암호화는 256비트 암호화와 함께 관련 데이터로 인증된 암호화(AEAD) 알고리즘을 사용합니다. 네트워크 성능에는 영향을 미치지 않습니다. 인스턴스 간에 이러한 전송 중 트래픽 암호화를 추가로 지원하려면 다음 요구 사항을 충족해야 합니다.

- 이러한 인스턴스는 다음 인스턴스 유형을 사용합니다.
  - 범용: M6i, M6id, M6in, M6idn, M7g
  - 메모리 최적화: R6i, R6id, R6in, R6idn, R7g, X2idn, X2iedn, X2iezn
- 인스턴스가 동일한 AWS 리전에 있습니다.
- 인스턴스가 동일한 VPC 또는 피어링된 VPC에 있으며, 트래픽이 로드 밸런서나 전송 게이트웨이 같은 가상 네트워크 디바이스 또는 서비스를 통과하지 않습니다.

Amazon Aurora 암호화된 DB 클러스터의 제한

Amazon Aurora 암호화된 DB 클러스터에는 다음과 같은 제한이 있습니다.

- 암호화된 DB 클러스터의 암호화를 비활성화할 수 없습니다.
- 암호화되지 않은 DB 클러스터의 암호화된 스냅샷은 생성할 수 없습니다.
- 암호화된 DB 클러스터의 스냅샷은 DB 클러스터와 동일한 KMS 키를 사용하여 암호화해야 합니다.
- 암호화되지 않은 DB 클러스터를 암호화된 DB 클러스터로 변환할 수 없습니다. 하지만 암호화되지 않은 스냅샷을 암호화된 Aurora DB 클러스터로 복원할 수 있습니다. 암호화되지 않은 스냅샷에서 복원할 때 KMS 키를 지정하면 가능합니다.
- 암호화되지 않은 Aurora DB 클러스터에서 암호화된 Aurora 복제본을 생성할 수 없습니다. 암호화된 Aurora DB 클러스터에서 암호화되지 않은 Aurora 복제본을 생성할 수 없습니다.
- 암호화된 스냅샷을 한 AWS 리전에서 다른 리전으로 복사하려면 대상 AWS 리전에 KMS 키를 지정해야 합니다. 이는 KMS 키가 생성된 AWS 리전에만 해당하기 때문입니다.

소스 스냅샷은 복사 프로세스 전체에서 암호화를 유지합니다. Amazon Aurora는 봉투 암호화를 사용하여 복사 프로세스 중에 데이터를 보호합니다. 봉투 암호화에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 봉투 암호화를 참조하세요.

- 암호화된 DB 클러스터의 암호화를 해제할 수 없습니다. 하지만 암호화된 DB 클러스터에서 데이터를 내보내고 암호화되지 않은 DB 클러스터로 해당 데이터를 가져올 수 있습니다.

AWS KMS key 관리

Amazon Aurora는 AWS Key Management Service(AWS KMS)와 자동으로 통합하여 키 관리를 수행합니다. Amazon Aurora는 봉투 암호화를 사용합니다. 봉투 암호화에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 봉투 암호화를 참조하세요.

두 가지 유형의 AWS KMS 키를 사용하여 DB 클러스터를 암호화할 수 있습니다.

- KMS 키를 완전히 제어하기 위해서는 고객 관리형 키를 생성해야 합니다. 고객 관리형 키에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 고객 관리형 키를 참조하세요.
- AWS 관리형 키 은(는) AWS KMS와 통합된 AWS 서비스가 고객의 계정에서 고객 대신 생성, 관리 및 사용하는 KMS 키입니다. 기본적으로 RDS AWS 관리형 키(`aws/rds`)는 암호화에 사용됩니다. RDS AWS 관리형 키는 관리, 교체 또는 삭제할 수 없습니다. AWS 관리형 키에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 AWS 관리형 키를 참조하십시오.

Amazon Aurora 암호화된 DB 클러스터에 사용되는 KMS 키를 관리하려면 AWS KMS 콘솔, AWS CLI 또는 AWS KMS API에서 AWS Key Management Service(AWS KMS)를 사용합니다. AWS 관리형 또는 고객 관리형 키로 수행한 모든 작업의 감사 로그를 보려면 AWS CloudTrail을 사용합니다. 키 교체에 대한 자세한 내용은 AWS KMS 키 교체를 참조하세요.

고객 관리형 키의 사용 권한 부여

Aurora가 암호화 작업에 고객 관리형 키를 사용하는 경우 Aurora 리소스를 만들거나 변경하는 사용자를 대신해 작업합니다.

고객 관리형 키를 사용하여 Aurora 리소스를 만들려면 고객 관리형 키에서 다음 작업을 직접적으로 호출할 수 있는 권한이 사용자에게 있어야 합니다.

- `kms:CreateGrant`
- `kms:DescribeKey`

키 정책에서 허용하는 경우 키 정책 또는 IAM 정책에서 이러한 필수 권한을 지정할 수 있습니다.

> **Important**
>
> Amazon RDS와 같은 관리형 서비스에서 AWS KMS 키 정책의 모든 리소스(*)에 대해 명시적 거부 문을 사용하는 경우 리소스 소유 계정을 허용하는 조건을 지정해야 합니다. 거부 규칙에 IAM 사용자에 대한 예외가 포함되어 있더라도 이 조건이 없다면 작업이 실패할 수 있습니다.

> **Tip**
>
> 최소 권한의 원칙을 따르려면 `kms:CreateGrant`에 대한 전체 액세스 권한을 허용하지 마세요. 대신에 kms:ViaService 조건 키를 사용하여 AWS 서비스에서 사용자를 대신하여 권한이 부여되는 경우에만 사용자가 KMS 키에 대한 권한을 부여할 수 있도록 허용합니다.

다양한 방법으로 IAM 정책을 더 엄격하게 설정할 수 있습니다. 예를 들어 Aurora에서 생성된 요청에 대해서만 고객 관리형 키를 사용할 수 있도록 허용하고 싶다면 `rds.<region>.amazonaws.com` 값과 함께 kms:ViaService 조건 키를 사용합니다. 또한 암호화 작업에 대한 고객 관리형 키 사용 조건으로 Amazon RDS 암호화 컨텍스트의 키 또는 값을 사용할 수도 있습니다.

자세한 내용을 알아보려면 AWS Key Management Service 개발자 안내서의 다른 계정의 사용자가 KMS 키를 사용하도록 허용 및 AWS KMS의 키 정책을 참조하세요.

Amazon RDS 암호화 컨텍스트

Aurora가 사용자의 KMS 키를 사용하거나 Amazon EBS가 Aurora를 대신하여 KMS 키를 사용하는 경우, 서비스가 암호화 컨텍스트를 지정합니다. 암호화 컨텍스트는 AWS KMS가 데이터 무결성을 보장하기 위해 사용하는 추가 인증 데이터(AAD)입니다. 암호화 작업에 대해 암호화 컨텍스트가 지정되면 서비스가 암호화 해제 작업에 대해 동일한 암호화 컨텍스트를 지정해야 합니다. 그렇지 않으면 암호화 해제가 실패합니다. 암호화 컨텍스트는 AWS CloudTrail 로그에도 기록되어, 해당 KMS 키가 사용된 이유를 이해하는 데 도움을 줍니다. CloudTrail 로그에 CMK 사용을 설명하는 여러 항목이 포함될 수 있지만, 각 로그 항목의 암호화 컨텍스트는 특히 해당 KMS 키를 사용한 이유를 파악하는 데 도움이 될 수 있습니다.

최소한, 다음 JSON 형식 예에서 보듯이 Aurora는 항상 암호화 컨텍스트에 DB 클러스터 ID를 사용합니다.

```
{ "aws:rds:dbc-id": "cluster-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

이 암호화 컨텍스트는 KMS 키가 사용된 DB 클러스터를 식별하는 데 도움이 될 수 있습니다.

KMS 키가 특정 DB 클러스터와 특정 Amazon EBS 볼륨에 사용되는 경우, 다음 JSON 형식 예에서 보듯이 암호화 컨텍스트에 DB 클러스터 ID와 Amazon EBS 볼륨 ID가 모두 사용됩니다.

```
{
  "aws:rds:dbc-id": "cluster-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화

애플리케이션에서 SSL(Secure Socket Layer) 또는 TLS(전송 계층 보안)를 사용하여 Aurora MySQL 또는 Aurora PostgreSQL을 실행하는 DB 클러스터에 대한 연결을 암호화할 수 있습니다.

SSL/TLS 연결은 클라이언트와 DB 클러스터 사이에 전송되는 데이터를 암호화하여 하나의 보안 계층을 제공합니다. 필요에 따라 SSL/TLS 연결에서는 데이터베이스에 설치된 서버 인증서를 검증하여 서버 ID 확인을 수행할 수 있습니다. 서버 ID 확인을 요구하려면 다음의 일반적인 절차를 따르세요.

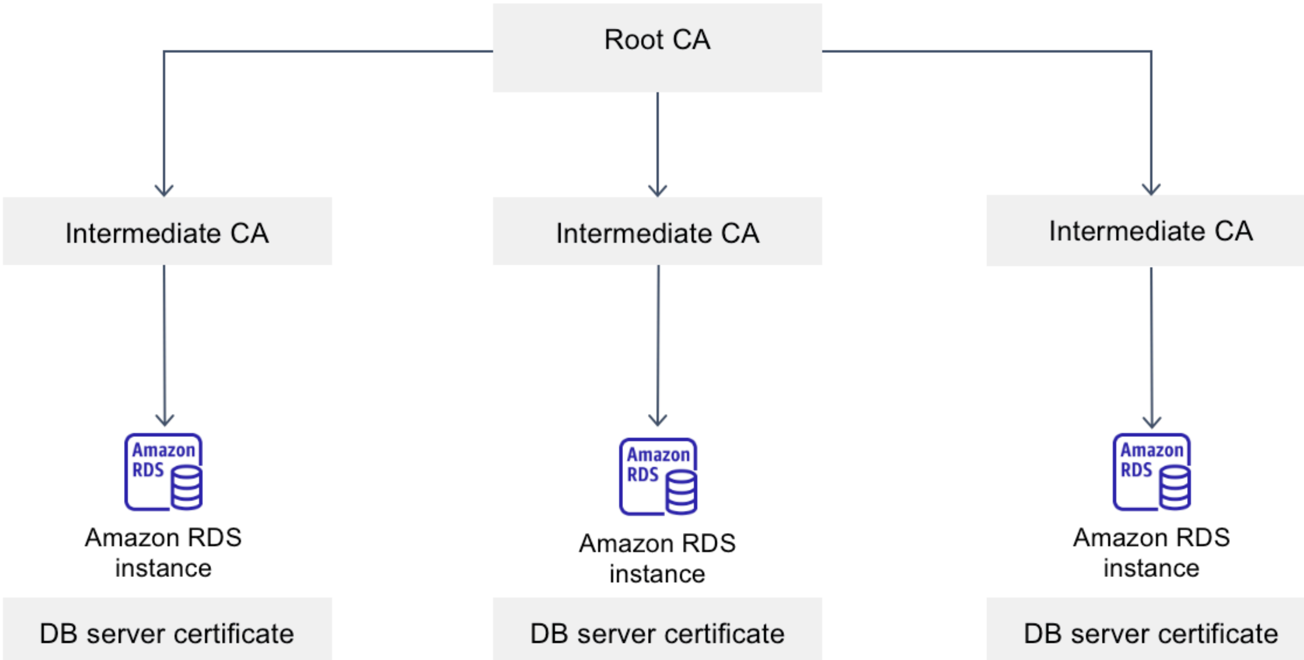
1. 데이터베이스의 DB 서버 인증서에 서명하는 인증 기관(CA)을 선택합니다. 인증 기관에 관한 자세한 내용은 인증 기관 단원을 참조하세요.
2. 데이터베이스에 연결할 때 사용할 인증서 번들을 다운로드합니다. 인증서 번들을 다운로드하려면 AWS 리전별 인증서 번들 단원을 참조하세요.

> **Note**
> 모든 인증서는 SSL/TLS 연결을 통한 다운로드에만 사용 가능합니다.

3. SSL/TLS 연결을 구현하는 DB 엔진 프로세스를 사용하여 데이터베이스에 연결합니다. 각 DB 엔진에는 SSL/TLS를 구현하기 위한 고유한 프로세스가 있습니다. 데이터베이스에서 SSL/TLS를 구현하는 방법을 알아보려면 DB 엔진에 해당하는 아래 링크로 이동하세요.
   - Amazon Aurora MySQL를 사용한 보안
   - Amazon Aurora PostgreSQL를 사용한 보안

인증 기관

인증 기관(CA)은 인증서 체인의 맨 위에 있는 루트 CA를 식별하는 인증서입니다. CA는 각 DB 인스턴스에 설치된 DB 서버 인증서에 서명합니다. DB 서버 인증서는 DB 인스턴스를 신뢰할 수 있는 서버로 식별합니다.



Amazon RDS는 데이터베이스의 DB 서버 인증서에 서명할 수 있는 다음 CA를 제공합니다.

인증 기관(CA)	설명	일반 이름(CN)
rds-ca-2019	RSA 2048 프라이빗 키 알고리즘과 SHA256 서명 알고리즘을 갖춘 인증 기관을 사용합니다. 이 CA는 2024년에 만료되며 자동 서버 인증서 교체를 지원하지 않습니다. 이 CA를 사용 중이고 동일한 표준을 유지하려면 rds-ca-rsa2048-g1 CA로 전환하는 것이 좋습니다.	Amazon RDS *region-identifier* CA 2019
rds-ca-rsa2048-g1	대부분의 AWS 리전에서 RSA 2048 프라이빗 키 알고리즘과 SHA256 서명 알고리즘을 갖춘 인증 기관을 사용합니다. AWS GovCloud (US) Regions에서 이 CA는 RSA 2048 프라이빗 키 알고리즘과 SHA384 서명 알고리즘을 갖춘 인증 기관을 사용합니다.	Amazon RDS *region-identifier* RSA2048 G1

인증 기관(CA)	설명	일반 이름(CN)
	이 CA의 유효 기간은 rds-ca-2019 CA보다 깁니다. 이 CA는 자동 서버 인증서 교체를 지원합니다.	
rds-ca-rsa4096-g1	RSA 4096 프라이빗 키 알고리즘과 SHA384 서명 알고리즘을 갖춘 인증 기관을 사용합니다. 이 CA는 자동 서버 인증서 교체를 지원합니다.	Amazon RDS *region-identifier* RSA4096 G1
rds-ca-ecc384-g1	ECC 384 프라이빗 키 알고리즘과 SHA384 서명 알고리즘을 갖춘 인증 기관을 사용합니다. 이 CA는 자동 서버 인증서 교체를 지원합니다.	Amazon RDS *region-identifier* ECC384 G1

> **Note**
>
> AWS CLI를 사용하는 경우 describe-certificates를 사용하여 위에 나열된 인증 기관의 유효성을 확인할 수 있습니다.

이러한 CA 인증서는 지역 및 글로벌 인증서 번들에 포함되어 있습니다. rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1 CA를 데이터베이스에 사용하면 RDS가 데이터베이스에서 DB 서버 인증서를 관리합니다. RDS는 DB 서버 인증서가 만료되기 전에 자동으로 교체합니다.

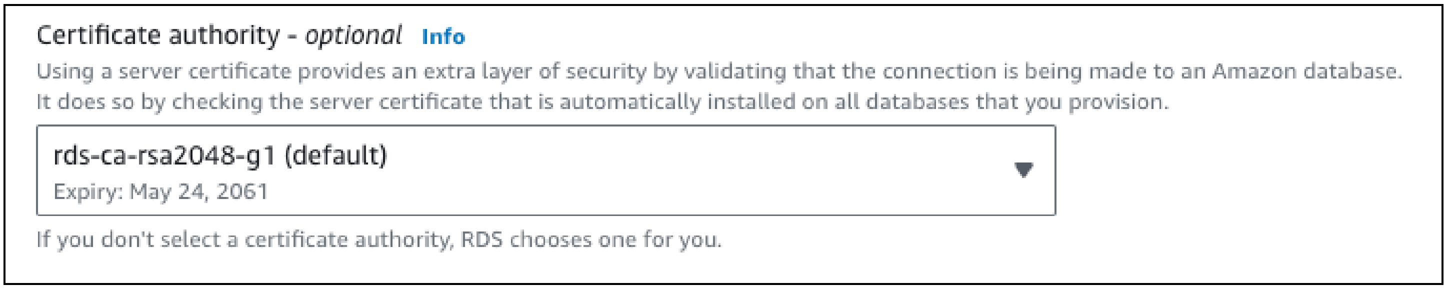
데이터베이스의 CA 설정

다음 작업을 수행할 때 데이터베이스의 CA를 설정할 수 있습니다.

- Aurora DB 클러스터 생성 - AWS CLI 또는 RDS API를 사용하여 DB 클러스터에서 첫 번째 DB 인스턴스를 생성할 때 Aurora 클러스터의 DB 인스턴스에 대한 CA를 설정할 수 있습니다. 현재는 RDS 콘솔을 사용하여 DB 클러스터를 설정할 때 DB 클러스터의 DB 인스턴스에 대한 CA를 설정할 수 없습니다. 지침은 Amazon Aurora DB 클러스터 생성 단원을 참조하십시오.
- DB 인스턴스 수정 - DB 클러스터의 DB 인스턴스를 수정하여 해당 CA를 설정할 수 있습니다. 지침은 DB 클러스터에서 DB 인스턴스 수정 단원을 참조하십시오.

> **Note**
>
> 기본 CA는 rds-ca-rsa2048-g1로 설정되어 있습니다. [modify-certificates](#) 명령을 사용하여 AWS 계정에 기본 CA를 재정의할 수 있습니다.

사용 가능한 CA는 DB 엔진 및 DB 엔진 버전에 따라 다릅니다. AWS Management Console을 사용하는 경우 다음 이미지에 표시된 것처럼 인증 기관 설정을 사용하여 CA를 선택할 수 있습니다.



콘솔에는 DB 엔진 및 DB 엔진 버전에 사용할 수 있는 CA만 표시됩니다. AWS CLI를 사용하는 경우 [create-db-instance](#) 또는 [modify-db-instance](#) 명령을 사용하여 DB 인스턴스의 CA를 설정할 수 있습니다.

AWS CLI를 사용하는 경우 [describe-certificate](#) 명령을 사용하여 계정에 사용할 수 있는 CA를 확인할 수 있습니다. 이 명령은 출력의 `ValidTill`에 각 CA의 만료 날짜도 표시합니다. [describe-db-engine-version](#) 명령을 사용하여 특정 DB 엔진 및 DB 엔진 버전에 사용할 수 있는 CA를 찾을 수 있습니다.

다음 예제는 기본 RDS for PostgreSQL DB 엔진 버전에 사용할 수 있는 CA를 보여 줍니다.

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

다음과 같은 출력이 표시됩니다. 사용 가능한 CA는 `SupportedCACertificateIdentifiers`에 나열되어 있습니다. 출력은 `SupportsCertificateRotationWithoutRestart`에서 DB 엔진 버전이 다시 시작하지 않고 인증서를 교체하는 기능을 지원하는지 여부도 표시합니다.

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
```

```
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-2019",
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

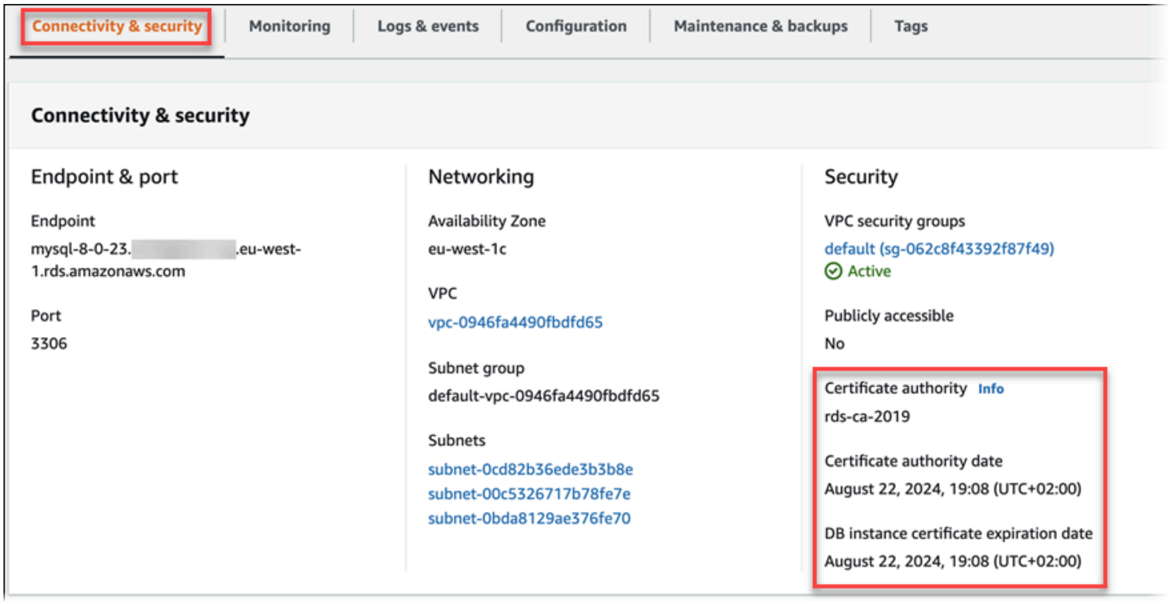
DB 서버 인증서 유효 기간

DB 서버 인증서의 유효 기간은 DB 엔진 및 DB 엔진 버전에 따라 다릅니다. DB 엔진 버전에서 재시작 없이 인증서를 교체하는 기능을 지원하는 경우 DB 서버 인증서의 유효 기간은 1년입니다. 그렇지 않으면 유효 기간은 3년입니다.

DB 서버 인증서 교체에 대한 자세한 내용은 자동 서버 인증서 교체 단원을 참조하세요.

DB 인스턴스의 CA 보기

콘솔의 연결 및 보안 탭에서 다음 이미지와 같이 데이터베이스의 CA에 대한 세부 정보를 볼 수 있습니다.



AWS CLI를 사용 중인 경우 describe-db-instances 명령을 사용하여 DB 인스턴스의 CA에 대한 세부 정보를 확인할 수 있습니다.

Aurora용 인증서 번들 다운로드

SSL 또는 TLS로 데이터베이스에 연결하는 경우 데이터베이스 인스턴스에 Amazon RDS의 신뢰 인증서가 필요합니다. 다음 표에서 적절한 링크를 선택하여 데이터베이스를 호스팅하는 AWS 리전에 해당하는 번들을 다운로드하세요.

AWS 리전별 인증서 번들

모든 AWS 리전 리전과 GovCloud(US) 리전의 인증서 번들에는 다음과 같은 인증서가 포함되어 있습니다.

- `rds-ca-2019` 중간 및 루트 인증서.
- `rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1` 및 `rds-ca-ecc384-g1` 루트 CA 인증서. 애플리케이션 트러스트 스토어에는 루트 CA 인증서만 등록하면 됩니다.

> **Note**
>
> Amazon RDS Proxy 및 Aurora Serverless v1 사용에서는 AWS Certificate Manager(ACM)의 인증서를 합니다. RDS 프록시를 사용하는 경우 Amazon RDS 인증서를 다운로드하거나 RDS 프록시 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다. 자세한 내용은 RDS Proxy에서 TLS/SSL 사용 단원을 참조하십시오.

Aurora Serverless v1을 사용하는 경우 Amazon RDS 인증서를 다운로드할 필요가 없습니다. 자세한 내용은 Aurora Serverless v1에서 TLS/SSL 사용 단원을 참조하십시오.

AWS 리전에 대한 인증서 번들을 다운로드하려면 다음 표에서 해당 데이터베이스를 호스팅하는 AWS 리전에 대한 링크를 선택하세요.

AWS 리전	인증서 번들(PEM)	인증서 번들(PKCS7)
모든 상업용 AWS 리전	global-bundle.pem	global-bundle.p7b
미국 동부(버지니아 북부)	us-east-1-bundle.pem	us-east-1-bundle.p7b
미국 동부(오하이오)	us-east-2-bundle.pem	us-east-2-bundle.p7b
미국 서부(캘리포니아 북부)	us-west-1-bundle.pem	us-west-1-bundle.p7b
미국 서부(오리건)	us-west-2-bundle.pem	us-west-2-bundle.p7b
Africa (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b
Asia Pacific (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
아시아 태평양(하이데라바드)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
아시아 태평양(자카르타)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
아시아 태평양(멜버른)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
아시아 태평양(뭄바이)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
Asia Pacific (Osaka)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
아시아 태평양(도쿄)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
Asia Pacific (Seoul)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
아시아 태평양(싱가포르)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
아시아 태평양(시드니)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b

AWS 리전	인증서 번들(PEM)	인증서 번들(PKCS7)
Canada (Central)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
캐나다 서부(캘거리)	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
유럽(프랑크푸르트)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
유럽(아일랜드)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
Europe (London)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
Europe (Milan)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
Europe (Paris)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
유럽(스페인)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
유럽(스톡홀름)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
유럽(취리히)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
이스라엘(텔아비브)	il-central-1-bundle.pem	il-central-1-bundle.p7b
Middle East (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
중동(UAE)	me-central-1-bundle.pem	me-central-1-bundle.p7b
남아메리카(상파울루)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b
모든 AWS GovCloud (US) Region	global-bundle.pem	global-bundle.p7b
AWS GovCloud(미국 동부)	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud(미국 서부)	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

CA 인증서의 내용 보기

CA 인증서 번들의 내용을 확인하려면 다음 명령을 사용합니다.

```
keytool -printcert -v -file global-bundle.pem
```

SSL/TLS 인증서 교체

Amazon RDS 인증 기관 인증서 rds-ca-2019는 2024년 8월에 만료됩니다. RDS DB 인스턴스 에 연결하기 위해 인증서 확인과 함께 보안 소켓 계층(SSL) 또는 Transport Layer Security(TLS)를 사용하거나 사용할 계획이라면 새 CA 인증서인 rds-ca-rsa2048-g1, rds-ca-rsa4096-g1 또는 rds-ca-ecc384-g1 중 하나를 사용하는 것을 고려하세요. 현재 인증서 확인과 함께 SSL/TLS를 사용하지 않는 경우에도 CA 인증서가 만료되었을 수 있으며, 인증서 확인과 함께 SSL/TLS를 사용하여 RDS 데이터베이스에 연결하려는 경우 새 CA 인증서로 업데이트해야 합니다.

Amazon RDS는 AWS 보안 모범 사례로 새 CA 인증서를 제공합니다. 새 인증서 및 지원되는 AWS 리전에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.

데이터베이스의 CA 인증서를 업데이트하려면 다음 방법을 사용하세요.

- DB 인스턴스를 수정하여 CA 인증서 업데이트
- 유지 관리를 적용하여 CA 인증서 업데이트

DB 인스턴스에서 새로운 CA 인증서를 사용하도록 업데이트하기 전에 RDS 데이터베이스에 연결하는 클라이언트 또는 애플리케이션을 업데이트해야 합니다.

인증서 교체 고려 사항

인증서를 교체하기 전에 다음 상황을 고려하세요.

- Amazon RDS Proxy 및 Aurora Serverless v1 사용에서는 AWS Certificate Manager(ACM)의 인증서를 합니다. RDS 프록시를 사용하는 경우 SSL/TLS 인증서를 교체할 때 RDS 프록시 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다. 자세한 정보는 RDS Proxy에서 TLS/SSL 사용을 참조하세요.
- Aurora Serverless v1을 사용하는 경우 Amazon RDS 인증서를 다운로드할 필요가 없습니다. 자세한 정보는 Aurora Serverless v1에서 TLS/SSL 사용을 참조하세요.
- 2020년 7월 28일 이전에 생성되었거나 rds-ca-2019 인증서로 업데이트된 DB 인스턴스가 있는 Go 버전 1.15 애플리케이션을 사용하는 경우 인증서를 다시 업데이트해야 합니다. 엔진에 따라 rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1 로 인증서를 업데이트하세요.

  새 CA 인증서 식별자를 사용하여 을 사용합니다. `describe-db-engine-versions` 명령을 사용하여 특정 DB 엔진 및 DB 엔진 버전에 사용할 수 있는 CA를 찾을 수 있습니다.

2020년 7월 28일 이후에 데이터베이스를 생성하거나 인증서를 업데이트한 경우에는 아무 조치도 필요하지 않습니다. 자세한 내용은 Go GitHub 문제 #39568를 참조하세요.

DB 인스턴스를 수정하여 CA 인증서 업데이트

다음 예제에서는 rds-ca-2019에서 rds-ca-rsa2048-g1로 CA 인증서를 업데이트합니다. 다른 인증서를 선택할 수 있습니다. 자세한 내용은 인증 기관 섹션을 참조하세요.

애플리케이션 트러스트 스토어를 업데이트하여 CA 인증서 업데이트와 관련된 가동 중단 시간을 줄이세요. CA 인증서 교체와 관련된 다시 시작에 대한 자세한 내용은 자동 서버 인증서 교체 섹션을 참조하세요.

DB 인스턴스를 수정하여 CA 인증서를 업데이트하는 방법

1. SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화에 설명된 대로 새 SSL/TLS 인증서를 다운로드합니다.
2. 새 SSL/TLS 인증서를 사용하도록 애플리케이션을 업데이트합니다.

   새 SSL/TLS 인증서를 위해 애플리케이션을 업데이트하는 방법은 애플리케이션에 따라 다릅니다. 애플리케이션 개발자와 함께 애플리케이션의 SSL/TLS 인증서를 업데이트하십시오.

   SSL/TLS 연결을 확인하고 각 DB 엔진의 애플리케이션을 업데이트하는 방법에 대한 자세한 내용은 다음 주제를 참조하십시오.

   - 새 TLS 인증서를 사용하여 Aurora MySQL DB 클러스터에 연결할 애플리케이션 업데이트
   - 새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션 업데이트

   Linux 운영 체제의 트러스트 스토어를 업데이트하는 샘플 스크립트는 트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트 섹션을 참조하세요.

   > **Note**
   >
   > 인증서 번들에는 이전 및 신규 CA의 인증서가 들어 있으므로 애플리케이션을 안전하게 업그레이드하고 전환 기간에 연결성을 유지할 수 있습니다. AWS Database Migration Service를 사용하여 데이터베이스를 DB 클러스터로 마이그레이션하는 경우, 연결이 끊기지 않고 마이그레이션이 진행되도록 인증서 번들을 사용하는 것이 좋습니다.

3. DB 인스턴스를 수정하여 CA를 rds-ca-2019에서 ca-rsa2048-g1로 변경합니다. CA 인증서를 업데이트하기 위해 데이터베이스를 다시 시작해야 하는지 확인하려면 describe-db-engine-versions 명령을 사용하여 `SupportsCertificateRotationWithoutRestart` 플래그를 확인합니다.

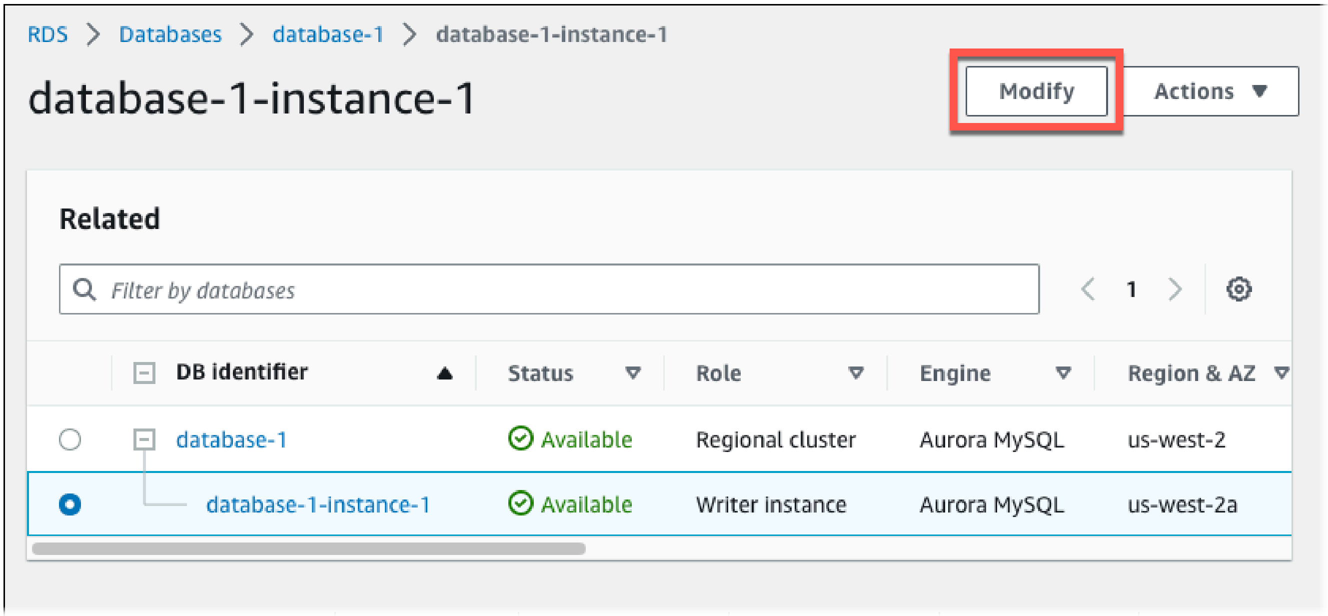
> **Note**
>
> CA 인증서를 업데이트하도록 수정한 후 Babelfish 클러스터를 재부팅합니다.

> **Important**
>
> 인증서 만료 후 연결 문제가 발생하는 경우 콘솔에서 즉시 적용을 지정하거나 `--apply-immediately`를 통해 AWS CLI 옵션을 지정하여 즉시 적용 옵션을 사용합니다. 기본적으로 이 작업은 다음 유지 관리 기간 중에 실행되도록 예약되어 있습니다.
>
> 기본 RDS CA와 다른 클러스터 CA에 재정의를 설정하려면 modify-certificates CLI 명령을 사용합니다.

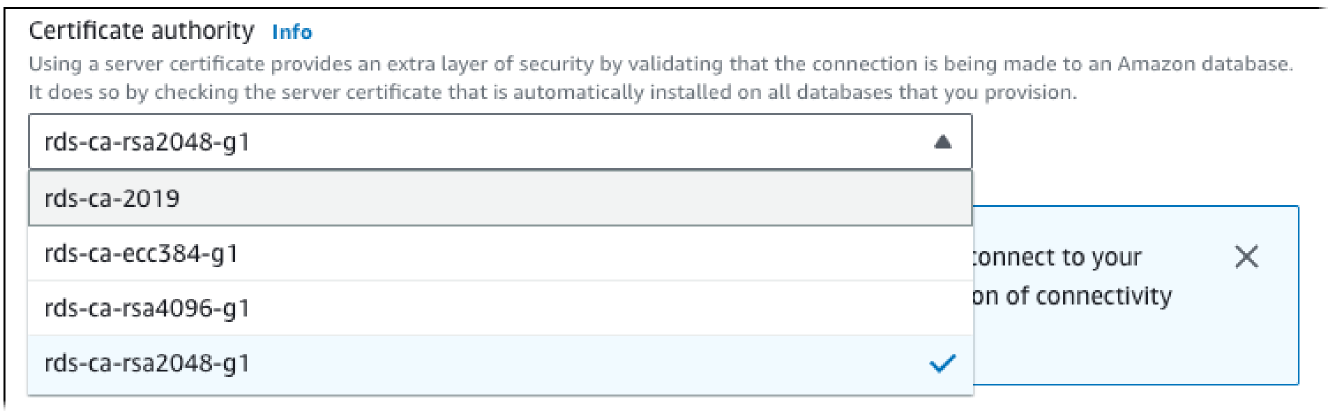
AWS Management Console 또는 AWS CLI를 사용하여 DB 인스턴스 또는 다중 AZ DB 클러스터의 CA 인증서를 에서 rds-ca-rsa2048-g1로 변경할 수 있습니다.

Console

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 수정하려는 DB 인스턴스를 선택합니다.
3. 수정을 선택합니다.



4. 연결 섹션에서 rds-ca-rsa2048-g1을 선택합니다.



5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.
7. 확인 페이지에서 변경 내용을 검토합니다. 내용이 정확할 경우 DB 인스턴스 수정 을 선택하여 변경 사항을 저장합니다.

> **Important**
>
> 이 작업을 예약하는 경우 클라이언트 측 트러스트 스토어를 미리 업데이트했는지 확인하십시오.

또는 뒤로를 선택하여 변경 내용을 편집하거나 취소를 선택하여 변경 내용을 취소합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스의 CA를 rds-ca-2019에서 rds-ca-rsa2048-g1로 변경하려면 [modify-db-instance](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-instance.html) 또는 [modify-db-cluster](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html) 명령을 직접 호출합니다. DB 인스턴스 식별자 및 `--ca-certificate-identifier` 옵션을 지정합니다.

`--apply-immediately` 파라미터를 사용하여 업데이트를 즉시 적용합니다. 기본적으로 이 작업은 다음 유지 관리 기간 중에 실행되도록 예약되어 있습니다.

> **Important**
>
> 이 작업을 예약하는 경우 클라이언트 측 트러스트 스토어를 미리 업데이트했는지 확인하십시오.

Example

다음 예시에서는 CA 인증서를 `rds-ca-rsa2048-g1`로 설정하여 `mydbinstance`를 수정합니다.

대상 LinuxmacOS, 또는Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

Windows의 경우:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> 인스턴스 재부팅이 필요한 경우 [modify-db-instance](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-instance.html) CLI 명령을 사용하여 `--no-certificate-rotation-restart` 옵션을 지정할 수 있습니다.

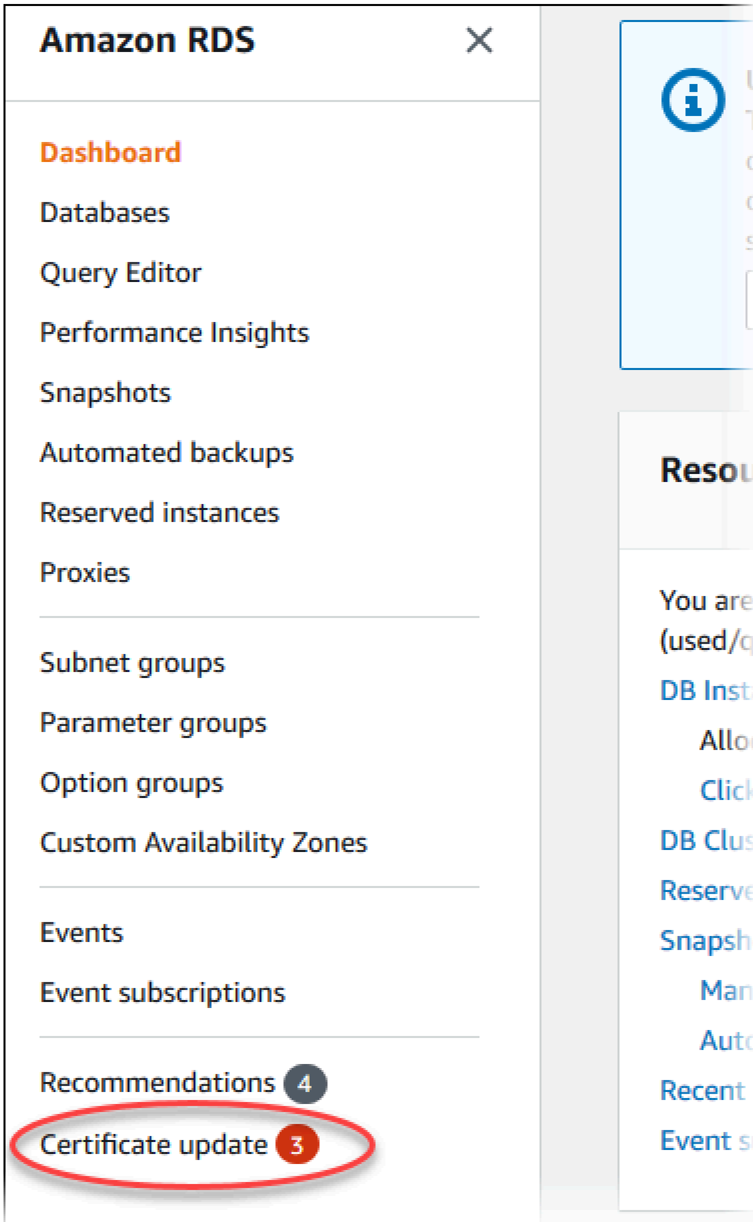
유지 관리를 적용하여 CA 인증서 업데이트

유지 관리를 적용하여 CA 인증서를 업데이트하려면 다음 단계를 수행합니다.

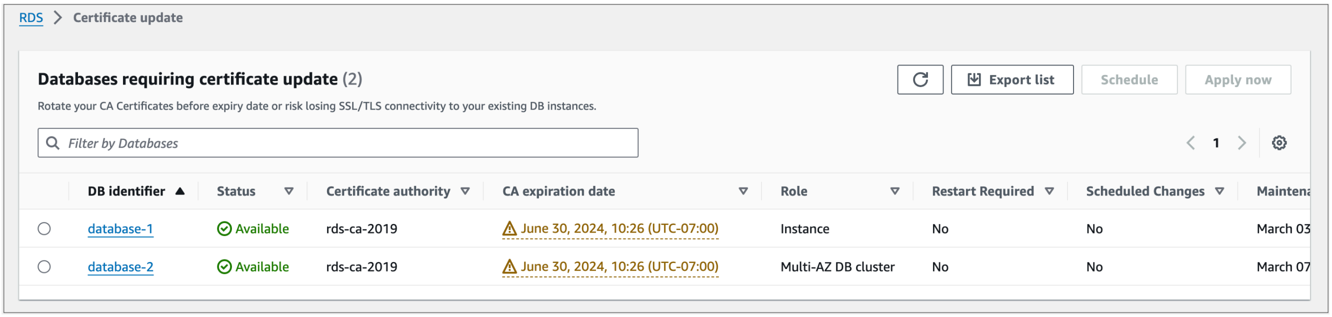
Console

유지 관리를 적용하여 CA 인증서를 업데이트하는 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 인증서 업데이트를 선택합니다.



인증서 업데이트가 필요한 데이터베이스 페이지가 표시됩니다.



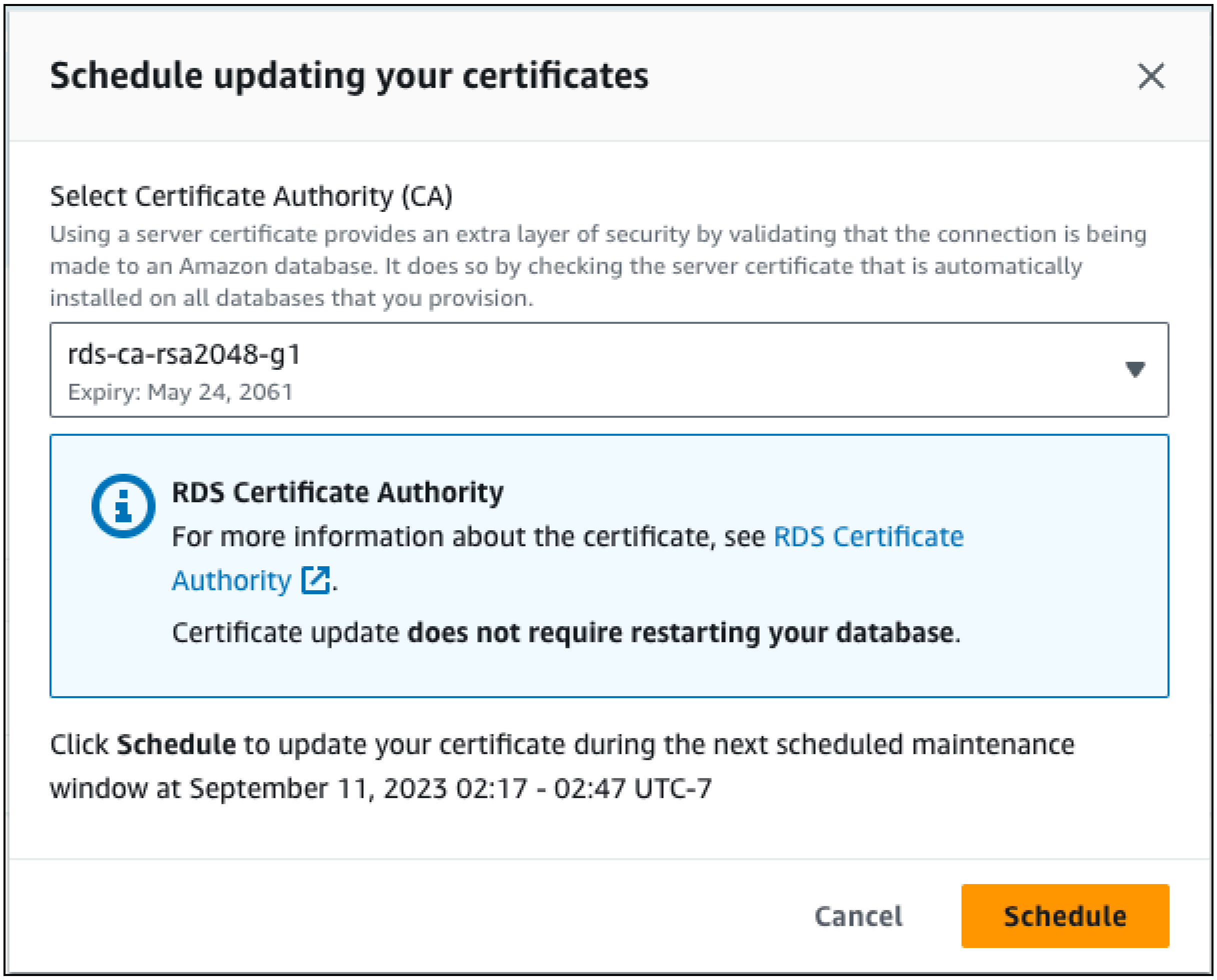
> **Note**
>
> 이 페이지에는 현재 AWS 리전의 DB 인스턴스만 표시됩니다. 둘 이상의 AWS 리전에 데이터베이스가 있는 경우 각 AWS 리전에서 이 페이지를 확인하여 이전 SSL/TLS 인증서가 있는 모든 DB 인스턴스를 확인합니다.

3. 업데이트할 DB 인스턴스를 선택합니다.

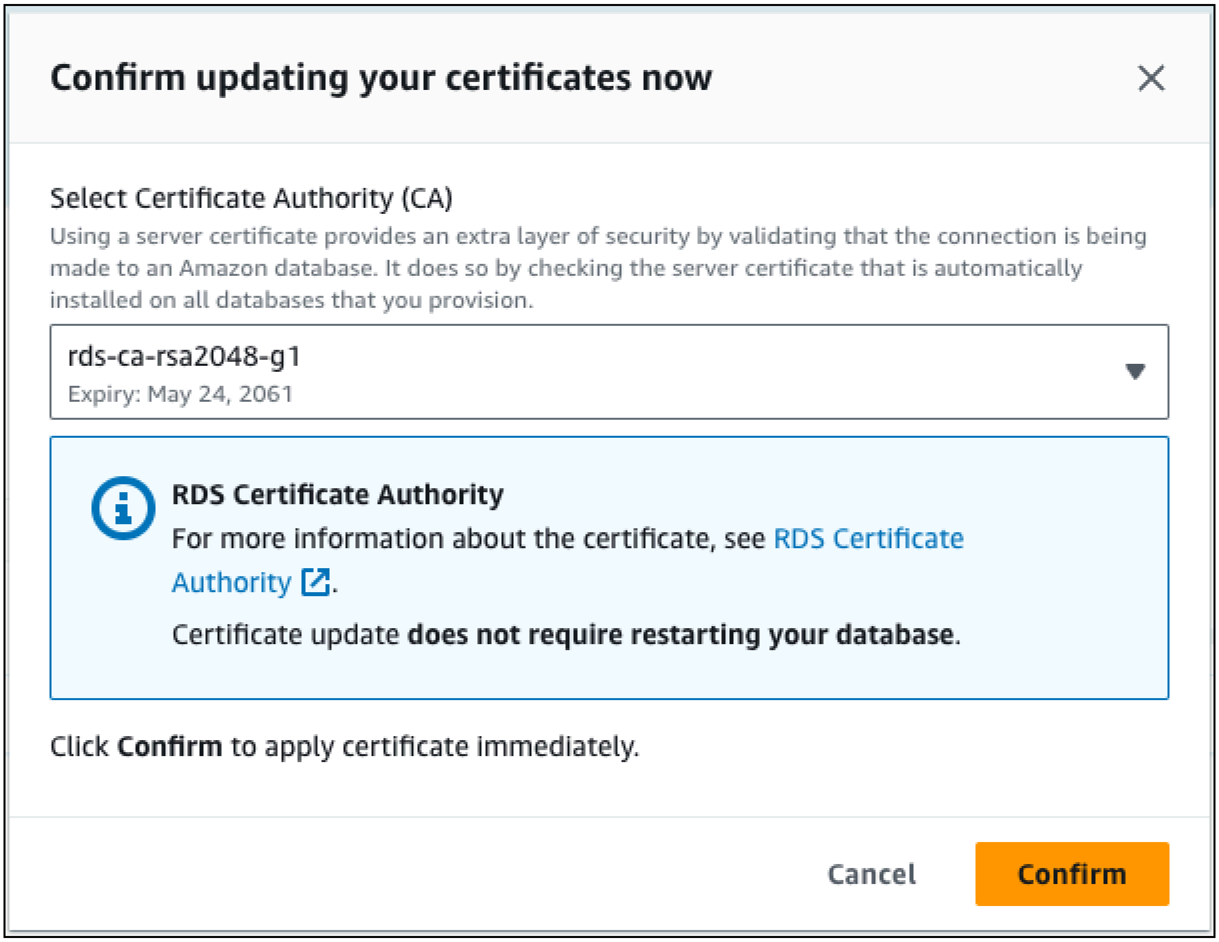
   일정을 선택하여 다음 유지 관리 기간에 따른 인증서 교체를 예약할 수 있습니다. 지금 적용을 선택하여 즉시 교체를 적용합니다.

> **Important**
>
> 인증서 만료 후 연결 문제가 발생하면 지금 적용 옵션을 사용합니다.

4. a. 일정을 선택하면 CA 인증서 교체를 확인하라는 메시지가 표시됩니다. 이 메시지에는 예약된 업데이트 기간도 표시됩니다.



b. 지금 적용을 선택하면 CA 인증서 교체를 확인하라는 메시지가 표시됩니다.



> **Important**
>
> 데이터베이스에서 CA 인증서 교체를 예약하기 전에 SSL/TLS 및 서버 인증서를 사용하여 연결하는 모든 클라이언트 애플리케이션을 업데이트합니다. 이러한 업데이트는 DB 엔진에만 적용됩니다. 이러한 클라이언트 애플리케이션을 업데이트한 후 CA 인증서 교체를 확인할 수 있습니다.

계속하려면 확인란을 선택한 다음 확인선택합니다.

5. 업데이트할 각 DB 인스턴스에 대해 3단계와 4단계를 반복합니다.

자동 서버 인증서 교체

루트 CA에서 자동 서버 인증서 교체를 지원하는 경우 RDS는 DB 서버 인증서의 교체를 자동으로 처리합니다. RDS는 자동 교체에 동일한 루트 CA를 사용하므로 사용자가 새 CA 번들을 다운로드할 필요가 없습니다. 인증 기관 섹션을 참조하세요.

DB 서버 인증서의 교체 및 유효 기간은 다음의 DB 엔진에 따라 다릅니다.

- DB 엔진이 재시작 없는 교체를 지원하는 경우 RDS는 사용자가 별도의 조치를 취하지 않아도 DB 서버 인증서를 자동으로 교체합니다. RDS는 사용자가 선호하는 유지 관리 기간 내에 DB 서버 인증서의 유효 기간이 반쯤 남은 시점에서 DB 서버 인증서 교체를 시도합니다. 새 DB 서버 인증서는 12개월 동안 유효합니다.
- DB 엔진에서 재시작 없이는 교체할 수 없는 경우, RDS는 DB 서버 인증서가 만료되기 최소 6개월 전에 유지 관리 이벤트에 대해 사용자에게 알려줍니다. 새 DB 서버 인증서는 36개월 동안 유효합니다.

describe-db-engine-versions 명령을 사용하고 `SupportsCertificateRotationWithoutRestart` 플래그를 검사하여 DB 엔진 버전이 재시작 없이 인증서를 교체하는 기능을 지원하는지 파악합니다. 자세한 정보는 데이터베이스의 CA 설정을 참조하세요.

트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트

다음은 인증서 번들을 트러스트 스토어로 가져오는 샘플 셸 스크립트입니다.

각 샘플 셸 스크립트는 Java Devopment Kit(JDK)의 일부인 keytool을 사용합니다. Docker 설치에 대한 자세한 내용은 Docker 설치 안내서를 참조하세요.

Linux

다음은 Linux 운영 체제에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 셸 스크립트입니다.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
```

```
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

다음은 macOS에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 셸 스크립트입니다.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done
```

```
rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Amazon RDS에서 SSL을 사용하는 방법에 대한 모범 사례를 자세히 알아보려면 Amazon RDS for Oracle로의 성공적인 SSL 연결을 위한 모범 사례를 참조하세요.

인터네트워크 트래픽 개인 정보

Amazon Aurora와 온프레미스 애플리케이션 간의 연결을 비롯해 Amazon Aurora 및 동일한 AWS 리전의 다른 AWS 리소스 간의 연결이 보호됩니다.

서비스와 온프레미스 클라이언트 및 애플리케이션 간의 트래픽

프라이빗 네트워크와 AWS 사이에 두 연결 옵션이 있습니다.

- AWS Site-to-Site VPN 연결. 자세한 내용은 AWS Site-to-Site VPN란 무엇입니까?를 참조하십시오.
- AWS Direct Connect 연결. 자세한 내용은 AWS Direct Connect란 무엇인가요?를 참조하세요.

AWS에서 게시한 API 작업을 사용하면 네트워크를 통해 Amazon Aurora에 액세스할 수 있습니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

Amazon Aurora의 자격 증명 및 액세스 관리

AWS Identity and Access Management(IAM)은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 Amazon RDS 리소스를 사용할 수 있도록 인증(로그인)되고 권한이 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

고객

AWS Identity and Access Management(IAM)를 사용하는 방법은 Amazon Aurora에서 수행하는 작업에 따라 달라집니다.

서비스 사용자 – Aurora 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 Aurora 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. Aurora의 기능에 액세스할 수 없는 경우 Amazon Aurora 자격 증명 및 액세스 문제 해결 단원을 참조하십시오.

서비스 관리자 – 회사에서 Aurora 리소스를 책임지고 있는 경우 Aurora에 대한 전체 액세스를 가지고 있을 것입니다. 서비스 관리자는 직원이 액세스해야 하는 Aurora 기능과 리소스를 결정합니다. 그런 다음 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 Aurora에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Aurora에서 IAM을 사용하는 방법 단원을 참조하십시오.

관리자 – 관리자는 Aurora에 대한 액세스 권한을 관리하는 정책을 작성하는 방법에 대해 자세히 알아보는 것이 좋습니다. IAM에서 사용할 수 있는 예제 Aurora 자격 증명 기반 정책 예제를 보려면 Amazon Aurora 자격 증명 기반 정책 예 단원을 참조하십시오.

자격 증명을 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로, 또는 IAM 역할을 수임하여 인증(AWS에 로그인)받아야 합니다.

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증이 페더레이션 ID의 예제입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 연동을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하세요.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우, 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용 AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS은(는) 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의 AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

AWS 계정을(를) 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 ID는 AWS 계정루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스에 액세스하도록 요구합니다.

페더레이션 ID는 엔터프라이즈 사용자 디렉터리, 웹 ID 공급자, AWS Directory Service, Identity Center 디렉터리의 사용자 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스에 액세스하는 모든 사용자입니다. 페더레이션 ID는 AWS 계정에 액세스할 때 역할을 수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 AWS 계정 및 애플리케이션에서 사용하기 위해 고유한 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다.

IAM 데이터베이스 인증은 Aurora에서 유효합니다. IAM을 사용한 DB 클러스터 인증에 대한 자세한 내용은 IAM 데이터베이스 인증 섹션을 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 ID입니다. 이 역할은 사용자와 비슷하지만, 특정 개인과 연결되지 않습니다. 역할 전환하여 AWS Management Console에서 IAM 역할을 임시로 수임할 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하세요.

임시 자격 증명이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 임시 사용자 권한 - 사용자는 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스를 사용하면 (역할을 프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.
- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 기능을 사용합니다. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션 - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
- 서비스 연결 역할 – 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

IAM 역할을 사용할지 여부를 알아보려면 IAM 사용 설명서의 사용자 대신 IAM 역할을 생성해야 하는 경우를 참조하십시오.

정책을 사용하여 액세스 관리

정책을 생성하고 IAM 자격 증명 또는 AWS 리소스에 연결하여 AWS에서 액세스를 제어합니다. 정책은 자격 증명이나 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 엔터티(루트 사용자, 사용자 또는 IAM 역할)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS 리소스에 액세스할 수 있는 사람과 해당 리소스에 대해 수행할 수 있는 작업을 지정할 수 있습니다. 모든 IAM 엔터티(권한 세트 또는 역할)는 처음에는 권한이 없습니다. 다시 말해, 기본적으로 사용자는 아무 작업도 수행할 수 없으며, 자신의 암호를 변경할 수도 없습니다. 사용자에게 태스크를 수행할 권한을 부여하기 위해 관리자는 사용자에게 권한 정책을 연결해야 합니다. 또한 관리자는 의도한 권한을 가지고 있는 그룹에 사용자를 추가할 수 있습니다. 관리자가 그룹에 권한을 부여하면 그룹의 모든 사용자가 해당 권한을 받습니다.

IAM 정책은 태스크를 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

자격 증명 기반 정책은 권한 세트나 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 권한 세트 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 권한 세트 및 역할에 추가할 수 있는 독립형 정책입니다. 관리형 정책에는 AWS관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

Amazon Aurora와 관련된 AWS 관리형 정책에 대한 자세한 내용은 Amazon RDS에 대한 AWS 관리형 정책 섹션을 참조하세요.

기타 정책 타입

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(권한 세트 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 엔터티에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 권한 세트 또는 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 IAM 사용 설명서의 IAM 개체에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 기업이 소유하는 여러 개의 AWS 계정을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 특성을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자을 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하십시오.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 권한 세트 또는 역할의 자격 증명 기반 정책과 세션 정책이 만나는 지점입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Aurora에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon Aurora에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 Aurora에서 사용할 수 있는지를 이해해야 합니다.

다음 표에는 Amazon Aurora에서 사용할 수 있는 IAM 기능이 나열되어 있습니다.

IAM 기능	Amazon Aurora 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	No
속성 기반 액세스 제어(ABAC)(정책 태그)	예
임시 보안 인증	예
전달 액세스 세션	예
서비스 역할	예
서비스 연결 역할	예

Amazon Aurora 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서에서 IAM으로 작업하는 AWS 서비스를 참조하세요.

주제

- Aurora 자격 증명 기반 정책
- Aurora 내의 리소스 기반 정책
- Aurora의 정책 작업
- Aurora의 정책 리소스
- Aurora의 정책 조건 키
- Aurora의 액세스 제어 목록(ACL)
- Aurora 태그가 있는 정책의 속성 기반 액세스 제어(ABAC)
- Aurora에서 임시 자격 증명 사용
- Aurora를 위한 전달 액세스 세션
- Aurora에 대한 서비스 역할
- Aurora에 대한 서비스 연결 역할

Aurora 자격 증명 기반 정책

ID 기반 정책 지원: 예.

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 보안 인증 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 보안 주체를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Aurora 자격 증명 기반 정책 예

Aurora 자격 증명 기반 정책 예제를 보려면 Amazon Aurora 자격 증명 기반 정책 예 단원을 참조하십시오.

Aurora 내의 리소스 기반 정책

리소스 기반 정책 지원: 아니요.

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자

는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 서로 다른 AWS 계정에 있는 경우, 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에도 리소스 액세스 권한을 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 크로스 계정 리소스 액세스를 참조하세요.

Aurora의 정책 작업

정책 작업 지원: 예.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWSAPI 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Aurora의 정책 작업은 작업 앞에 다음 접두사를 사용합니다. `rds:`. 예를 들어 Amazon RDS `DescribeDBInstances` API 작업으로 DB 인스턴스를 설명할 수 있는 권한을 부여하려면 해당 정책에 `rds:DescribeDBInstances` 작업을 포함합니다. 정책 문에는 `Action` 또는 `NotAction` 요소가 반드시 추가되어야 합니다. Aurora는 이 서비스로 수행할 수 있는 태스크를 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "rds:action1",
      "rds:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "rds:Describe*"
```

Aurora 작업 목록을 보려면 서비스 권한 부여 참조에서 Amazon RDS에서 정의한 작업을 참조하세요.

Aurora의 정책 리소스

정책 리소스 지원: 예.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

DB 인스턴스 리소스는 다음과 같은 Amazon 리소스 이름(ARN)을 갖습니다.

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름(ARN) 및 AWS 서비스 네임스페이스 단원을 참조하세요.

예를 들어, 문에서 `dbtest` DB 인스턴스를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

특정 계정에 속하는 모든 DB 인스턴스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

리소스를 생성하기 위한 작업과 같은 일부 RDS API 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우 와일드카드(*)를 사용하면 됩니다.

```
"Resource": "*"
```

다양한 Amazon RDS API 작업에는 여러 리소스가 관여합니다. 예를 들어 `CreateDBInstance`는 DB 인스턴스를 생성합니다. DB 인스턴스를 생성할 때는 사용자가 특정 보안 그룹과 파라미터 그룹을 사용해야 한다고 지정할 수 있습니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [
      "resource1",
      "resource2"
```

Aurora 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 권한 부여 참조에서 Amazon RDS에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon RDS에서 정의한 작업을 참조하세요.

Aurora의 정책 조건 키

서비스별 정책 조건 키 지원: 예.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS는 논리적 AND 작업을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우, AWS는 논리적 OR 작업을 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Aurora에서는 자체 조건 키 집합을 정의하고 일부 전역 조건 키 사용도 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

모든 RDS API 작업은 `aws:RequestedRegion` 조건 키를 지원합니다.

Aurora 조건 키 목록을 보려면 서비스 권한 부여 참조에서 Amazon RDS에서 정의한 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon RDS에서 정의한 작업을 참조하세요.

Aurora의 액세스 제어 목록(ACL)

액세스 제어 목록(ACL) 지원; 아니요.

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Aurora 태그가 있는 정책의 속성 기반 액세스 제어(ABAC)

정책의 ABAC(속성 기반 액세스 제어) 태그 지원: 예.

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 엔티티(사용자 또는 역할) 및 많은 AWS리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

Aurora 리소스 태그 지정에 대한 자세한 내용은 조건 지정: 사용자 지정 태그 사용 단원을 참조하십시오. 리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 두 가지 값이 있는 특정 태그를 사용하는 리소스 작업에 대한 권한 부여에서 확인할 수 있습니다.

Aurora에서 임시 자격 증명 사용

임시 자격 증명 지원: 예.

일부 AWS 서비스는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 보안 인증으로 작동하는 AWS 서비스를 비롯한 추가 정보는 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

사용자 이름과 암호를 제외한 다른 방법을 사용하여 AWS Management Console에 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어, 회사의 Single Sign-On(SSO) 링크를 사용하여 AWS에 액세스하면 해당 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 만들 수 있습니다 그런 다음 이러한 임시 자격 증명을 사용하여 AWS에 액세스할 수 있습니다. AWS에서는 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성할 것을 권장합니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

Aurora를 위한 전달 액세스 세션

전달 액세스 세션 지원: 예.

IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 위탁자로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Aurora에 대한 서비스 역할

서비스 역할 지원: 예.

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 Aurora 기능이 중단될 수 있습니다. Aurora에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

Aurora에 대한 서비스 연결 역할

서비스 연결 역할 지원: 예.

서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

Aurora 서비스 연결 역할을 사용하는 방법에 대한 자세한 내용은 Amazon Aurora에 서비스 연결 역할 사용 섹션을 참조하세요.

Amazon Aurora 자격 증명 기반 정책 예

기본적으로 권한 세트 및 역할은 Aurora 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI 또는 AWS API를 사용해 태스크를 수행할 수 없습니다. 관리자는 필요한 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 권한 세트와 역할에 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 권한 세트 또는 역할에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon RDS 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS CloudFormation와 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 – AWS 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우, 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Aurora 콘솔 사용

Amazon Aurora 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한을 통해 AWS 계정에서 Amazon Aurora 리소스에 대한 세부 정보를 나열하고 볼 수 있어야 합니다. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 직접 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

해당 엔터티가 Aurora 콘솔을 여전히 사용할 수 있도록 하려면 AWS 관리형 정책도 엔터티에 연결합니다.

```
AmazonRDSReadOnlyAccess
```

자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

콘솔 사용에 필요한 권한

콘솔에서 작업하려면 최소한의 권한이 사용자에게 필요합니다. 이러한 권한이 있어야만 사용자가 자신의 AWS 계정에서 사용할 Amazon Aurora 리소스를 설명하고, Amazon EC2 보안 및 네트워크 정보 등 다른 관련 정보를 입력할 수 있습니다.

최소 필수 권한보다 더 제한적인 IAM 정책을 만들면 콘솔은 해당 IAM 정책에 연결된 사용자에 대해 의도대로 작동하지 않습니다. 이 사용자가 콘솔을 사용할 수 있도록 하려면 `AmazonRDSReadOnlyAccess` 관리형 정책을 사용자에게 연결합니다(정책을 사용하여 액세스 관리 참조).

AWS CLI 또는 Amazon RDS API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다.

다음 정책은 루트 AWS 계정의 모든 Amazon Aurora 리소스에 대해 전체 액세스 권한을 부여합니다.

```
AmazonRDSFullAccess
```

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 완료할 수 있는 권한이 포함됩니다.

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

Aurora에서 리소스를 생성, 수정 및 삭제하기 위한 권한 정책

다음 섹션에는 리소스에 대한 액세스를 허용하고 제한하는 권한 정책의 예제가 나와 있습니다.

사용자에게 AWS 계정에서 DB 인스턴스를 생성하도록 허용

다음은 ID가 123456789012인 계정에 AWS 계정에 대한 DB 인스턴스를 생성하도록 허용하는 정책 예제입니다. 이 정책에 따라 새 DB 인스턴스의 이름은 `test`로 시작해야 합니다. 또한 새 DB 인스턴스는 MySQL 데이터베이스 엔진 및 `db.t2.micro` DB 인스턴스 클래스를 사용해야 합니다. 또한 새로운 DB 인스턴스는 `default`로 시작하는 옵션 그룹과 DB 파라미터 그룹을, 그리고 `default` 서브넷 그룹을 사용해야 합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
```

```
            ],
            "Resource": [
               "arn:aws:rds:*:123456789012:db:test*",
               "arn:aws:rds:*:123456789012:og:default*",
               "arn:aws:rds:*:123456789012:pg:default*",
               "arn:aws:rds:*:123456789012:subgrp:default"
            ],
            "Condition": {
               "StringEquals": {
                  "rds:DatabaseEngine": "mysql",
                  "rds:DatabaseClass": "db.t2.micro"
               }
            }
         }
      ]
}
```

다음 사용자 권한을 지정하는 단일 명령문이 정책에 포함됩니다.

- 정책은 계정이 CreateDBInstance API 작업을 사용하여 DB 인스턴스를 생성할 수 있도록 허용합니다. 이는 create-db-instance AWS CLI 명령과 AWS Management Console에도 적용됩니다.
- `Resource` 요소는 사용자가 리소스 위치에서 또는 리소스를 사용하여 작업을 수행할 수 있도록 지정합니다. 리소스는 Amazon 리소스 이름(ARN)을 사용하여 지정합니다. 이 ARN에는 리소스가 속하는 서비스 이름(`rds`), AWS 리전(`*`는 위의 예제에서 사용하는 모든 리전을 의미함), AWS 계정 번호(이 예에서는 `123456789012`가 계정 번호임) 및 리소스 유형이 포함됩니다. ARN 생성에 대한 자세한 내용은 Amazon RDS의 Amazon 리소스 이름(ARN) 단원을 참조하십시오.

  위의 예제에서 `Resource` 요소는 사용자 리소스에 대해 다음과 같은 정책 제약 조건을 지정합니다.

  - 새 DB 인스턴스에 대한 DB 인스턴스 식별자는 `test`로 시작해야 합니다(예: `testCustomerData1`, `test-region2-data`).
  - 새로운 DB 인스턴스의 옵션 그룹은 `default`로 시작해야 합니다.
  - 새로운 DB 인스턴스의 DB 파라미터 그룹은 `default`로 시작해야 합니다.
  - 새로운 DB 인스턴스의 서브넷 그룹은 `default` 서브넷 그룹이 되어야 합니다.
- `Condition` 요소는 DB 엔진은 MySQL이 되고, DB 인스턴스 클래스는 `db.t2.micro`가 되도록 지정합니다. `Condition` 요소는 정책 적용 시 조건을 지정합니다. 그 밖에도 `Condition` 요소를 사용하여 다른 권한이나 제한을 추가할 수 있습니다. 조건 지정에 대한 자세한 내용은 Aurora의 정책 조건 키 단원을 참조하십시오. 이 예제에서는 `rds:DatabaseEngine` 및 `rds:DatabaseClass` 조건을 지정합니다. `rds:DatabaseEngine`의 유효 조건 값에 대한 정보는 CreateDBInstance의

Engine 파라미터 아래 목록을 참조하십시오. rds:DatabaseClass의 유효 조건 값에 대한 정보는 DB 인스턴스 클래스에 지원되는 DB 엔진 단원을 참조하십시오.

자격 증명 기반 정책에서는 권한을 가질 보안 주체를 지정하지 않으므로 이 정책은 Principal 요소를 지정하지 않습니다. 정책을 사용자에게 연결할 경우 사용자는 암시적인 보안 주체입니다. IAM 역할에 권한 정책을 연결할 경우 역할의 신뢰 정책에 식별된 보안 주체는 권한을 가집니다.

Aurora 작업 목록을 보려면 서비스 권한 부여 참조에서 Amazon RDS에서 정의한 작업을 참조하세요.

사용자가 모든 RDS 리소스에서 Describe 작업을 수행할 수 있도록 허용

다음 권한 정책은 사용자에게 Describe로 시작하는 모든 작업을 실행할 수 있는 권한을 부여합니다. 이러한 작업은 DB 인스턴스와 같은 RDS 리소스에 대한 정보를 보여 줍니다. Resource 요소에 와일드카드 문자(*)가 있으면 계정이 소유한 모든 Amazon Aurora 리소스에 대해 작업이 허용됩니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

사용자가 지정된 DB 파라미터 그룹 및 서브넷 그룹을 사용하는 DB 인스턴스를 생성할 수 있도록 허용

다음 권한 정책은 사용자가 mydbpg DB 파라미터 그룹 및 mydbsubnetgroup DB 서브넷 그룹을 사용해야 하는 DB 인스턴스만 생성할 수 있도록 허용하는 권한을 부여합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
```

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "rds:CreateDBInstance",
            "Resource": [
                "arn:aws:rds:*:*:pg:mydbpg",
                "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
            ]
        }
    ]
}
```

두 가지 값이 있는 특정 태그를 사용하는 리소스 작업에 대한 권한 부여

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 Aurora 리소스에 대한 액세스를 제어할 수 있습니다. 다음 정책은 `development` 또는 `test`로 설정된 `stage` 태그를 사용하여 DB 인스턴스에서 `CreateDBSnapshot` API 작업을 수행할 수 있는 권한을 허용합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowAnySnapshotName",
            "Effect":"Allow",
            "Action":[
                "rds:CreateDBSnapshot"
            ],
            "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
        },
        {
            "Sid":"AllowDevTestToCreateSnapshot",
            "Effect":"Allow",
            "Action":[
                "rds:CreateDBSnapshot"
            ],
            "Resource":"arn:aws:rds:*:123456789012:db:*",
            "Condition":{
                "StringEquals":{
                    "rds:db-tag/stage":[
```

```
                    "development",
                    "test"
                 ]
              }
           }
        }
    ]
}
```

다음 정책은 `development` 또는 `test`로 설정된 `stage` 태그를 사용하여 DB 인스턴스에서 `ModifyDBInstance` API 작업을 수행할 수 있는 권한을 허용합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
```

```
                ]
            }
        }
    }
  ]
}
```

사용자의 DB 인스턴스 삭제 방지

다음 권한 정책은 사용자의 특정 DB 인스턴스 삭제를 방지하는 권한을 부여합니다. 예를 들어, 관리자가 아닌 모든 사용자에 대해 프로덕션 DB 인스턴스를 삭제할 수 있는 권한을 거부해야 할 수 있습니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

리소스에 대한 모든 액세스 거부

리소스에 대한 액세스를 명시적으로 거부할 수 있습니다. 거부 정책은 허용 정책보다 우선합니다. 다음 정책은 사용자가 리소스를 관리할 수 있는 권한을 명시적으로 거부합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
```

```
        {
            "Effect": "Deny",
            "Action": "rds:*",
            "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
        }
    ]
}
```

정책 예: 조건 키 사용

다음은 Amazon Aurora IAM 권한 정책에서 조건 키를 사용할 수 있는 방법의 예입니다.

예제 1: 특정 DB 엔진을 사용하고 MultiAZ가 아닌 DB 인스턴스를 생성할 수 있는 권한 부여

다음 정책은 RDS 조건 키를 사용하며, 사용자가 MySQL 데이터베이스 엔진을 사용하는 DB 인스턴스만 생성할 수 있도록 허용하며, MultiAZ를 사용하지 않습니다. `Condition` 요소는 데이터베이스 엔진이 MySQL이라는 요구 사항을 나타냅니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

예제 2: 특정 DB 인스턴스 클래스에 대한 DB 인스턴스를 만들고 프로비저닝된 IOPS를 사용하는 DB 인스턴스를 만들 수 있는 권한을 명시적으로 거부

다음 정책은 가장 크고 가장 비싼 DB 인스턴스 클래스인 DB 인스턴스 클래스 `r3.8xlarge` 및 `m4.10xlarge`를 사용하는 DB 인스턴스를 만들 수 있는 권한을 명시적으로 거부합니다. 또한 이 정책은 추가 비용이 발생하는 프로비저닝된 IOPS를 사용하는 DB 인스턴스를 사용자가 생성하지 못하도록 합니다.

명시적으로 거부하는 권한은 이미 부여된 다른 모든 권한에 우선합니다. 따라서 부여하지 않으려는 권한을 자격 증명이 우연히 획득하지 않도록 할 수 있습니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

예제 3: 리소스에 태그 지정하는 데 사용할 수 있는 태그 키와 값 집합 제한

다음 정책은 RDS 조건 키를 사용하고 키가 `stage`인 태그를 값이 `test`, `qa`, `production`인 리소스에 추가할 수 있도록 허용합니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource": "*",
         "Condition": {
            "streq": {
               "rds:req-tag/stage": [
                  "test",
                  "qa",
                  "production"
               ]
            }
         }
      }
   ]
}
```

조건 지정: 사용자 지정 태그 사용

Amazon Aurora에서는 사용자 지정 태그를 사용하여 IAM 정책에서 조건을 지정할 수 있습니다.

예를 들어 이름이 `environment`인 태그를 `beta`, `staging`, `production` 등의 값으로 DB 인스턴스에 추가한다고 가정하겠습니다. 그러면 `environment` 태그 값에 따라 특정 사용자를 DB 인스턴스로 제한하는 정책을 생성할 수 있습니다.

> **Note**
>
> 사용자 지정 태그 식별자는 대/소문자를 구분합니다.

다음 표에는 `Condition` 요소에서 사용할 수 있는 RDS 태그 식별자가 나와 있습니다.

RDS 태그 식별자	적용 대상
`db-tag`	읽기 전용 복제본을 포함하는 DB 인스턴스입니다.
`snapshot-tag`	DB 스냅샷
`ri-tag`	예약 DB 인스턴스
`og-tag`	DB 옵션 그룹
`pg-tag`	DB 파라미터 그룹
`subgrp-tag`	DB 서브넷 그룹
`es-tag`	이벤트 구독
`cluster-tag`	DB 클러스터
`cluster-pg-tag`	DB 클러스터 파라미터 그룹
`cluster-snapshot-tag`	DB 클러스터 스냅샷

사용자 지정 태그 조건의 구문은 다음과 같습니다.

```
"Condition":{"StringEquals":{"rds:rds-tag-identifier/tag-name":
["value"]} }
```

예를 들어, 다음 `Condition` 요소는 태그 이름이 `environment`이고 태그 값이 `production`인 DB 인스턴스에 적용됩니다.

```
"Condition":{"StringEquals":{"rds:db-tag/environment": ["production"]} }
```

태그 생성에 대한 자세한 내용은 [Amazon Aurora 및 Amazon RDS 리소스에 태그 지정](#) 단원을 참조하십시오.

> **Important**
>
> 태깅을 사용하여 RDS 리소스에 대한 액세스를 관리하는 경우 RDS 리소스의 태그에 대한 액세스의 보안을 유지하는 것이 좋습니다. `AddTagsToResource` 및

RemoveTagsFromResource 작업에 대한 정책을 생성하여 태그에 대한 액세스를 관리할 수 있습니다. 예를 들어, 다음 정책은 모든 리소스에 대해 태그를 추가하거나 제거할 수 있는 사용자의 권한을 거부합니다. 그런 다음 특정 사용자가 태그를 추가하거나 제거할 수 있도록 허용하기 위한 정책을 생성할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

Aurora 작업 목록을 보려면 서비스 권한 부여 참조에서 [Amazon RDS에서 정의한 작업]을 참조하세요.

정책 예: 사용자 지정 태그 사용

다음은 Amazon Aurora IAM 권한 정책에서 사용자 지정 태그를 사용할 수 있는 방법의 예입니다. Amazon Aurora 리소스에 태그를 추가하는 방법에 대한 자세한 내용은 [Amazon RDS의 Amazon 리소스 이름(ARN)] 단원을 참조하십시오.

> **Note**
>
> 모든 예는 us-west-2 리전을 사용하며 가상의 계정 ID를 포함합니다.

예제 1: 두 개의 값을 갖는 특정 태그를 사용하는 리소스 작업에 대한 권한 부여

다음 정책은 development 또는 test로 설정된 stage 태그를 사용하여 DB 인스턴스에서 CreateDBSnapshot API 작업을 수행할 수 있는 권한을 허용합니다.

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
       {
          "Sid":"AllowAnySnapshotName",
          "Effect":"Allow",
          "Action":[
             "rds:CreateDBSnapshot"
          ],
          "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
       },
       {
          "Sid":"AllowDevTestToCreateSnapshot",
          "Effect":"Allow",
          "Action":[
             "rds:CreateDBSnapshot"
          ],
          "Resource":"arn:aws:rds:*:123456789012:db:*",
          "Condition":{
             "StringEquals":{
                 "rds:db-tag/stage":[
                   "development",
                   "test"
                ]
             }
          }
       }
    ]
}
```

다음 정책은 development 또는 test로 설정된 stage 태그를 사용하여 DB 인스턴스에서 ModifyDBInstance API 작업을 수행할 수 있는 권한을 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
       {
          "Sid":"AllowChangingParameterOptionSecurityGroups",
          "Effect":"Allow",
          "Action":[
             "rds:ModifyDBInstance"
          ],
          "Resource":" [
             "arn:aws:rds:*:123456789012:pg:*",
```

```
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

예제 2: 지정된 DB 파라미터 그룹을 사용하는 DB 인스턴스를 만들 수 있는 권한을 명시적으로 거부

다음 정책은 특정 태그 값이 있는 DB 파라미터 그룹을 사용하는 DB 인스턴스를 만들 수 있는 권한을 명시적으로 거부합니다. DB 인스턴스를 생성할 때 특정 고객 생성 DB 파라미터 그룹을 사용해야 할 경우 이 정책을 적용할 수 있습니다. Deny를 사용하는 정책은 더 광범위한 정책에서 부여한 액세스 권한을 제한하기 위해 가장 자주 사용됩니다.

명시적으로 거부하는 권한은 이미 부여된 다른 모든 권한에 우선합니다. 따라서 부여하지 않으려는 권한을 자격 증명이 우연히 획득하지 않도록 할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
```

```
            "Condition":{
               "StringEquals":{
                  "rds:pg-tag/usage":"prod"
               }
            }
         }
      ]
   }
```

예제 3: 인스턴스 이름에 사용자 이름이 접두사로 붙은 DB 인스턴스 작업에 대한 권한 부여

다음 정책은 DB 인스턴스 이름에 사용자 이름이 접두사로 붙어 있고 AddTagsToResource와 동일한 RemoveTagsFromResource라는 태그가 있거나 stage라는 태그가 없는 DB 인스턴스에서 API(devo 또는 stage 제외)를 호출할 수 있는 권한을 허용합니다.

정책의 Resource 줄은 Amazon 리소스 이름(ARN)을 기준으로 리소스를 식별합니다. Amazon Aurora 리소스에서 ARN을 사용하는 방법에 대한 자세한 내용은 [Amazon RDS의 Amazon 리소스 이름(ARN)](#) 단원을 참조하십시오.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
               "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

생성 시 Aurora 리소스를 태그하는 권한 부여

일부 RDS API 작업을 사용하면 리소스를 생성할 때 태그를 지정할 수 있습니다. 리소스 태그를 사용하여 속성 기반 제어(ABAC)를 구현할 수 있습니다. 자세한 내용은 What is ABAC for AWS? 및 태그를 사용한 AWS 리소스 액세스 제어를 참조하세요.

사용자가 생성 시 리소스에 태그를 지정할 수 있으려면 `rds:CreateDBCluster`와 같은 리소스를 생성하는 작업을 사용할 권한이 있어야 합니다. 생성 작업에서 태그가 지정되면 RDS는 `rds:AddTagsToResource` 작업에서 추가 권한 부여를 수행해 사용자에게 태그를 생성할 권한이 있는지 확인합니다. 따라서 사용자는 `rds:AddTagsToResource` 작업을 사용할 명시적 권한도 가지고 있어야 합니다.

`rds:AddTagsToResource` 작업에 대한 IAM 정책 정의에서 `aws:RequestTag` 조건 키를 사용하여 리소스에 태그를 지정하는 요청에 태그를 요구할 수 있습니다.

예를 들어 다음 정책은 사용자가 DB 인스턴스를 생성하고 DB 인스턴스 생성 중에 태그를 적용할 수 있도록 허용하지만 특정 태그 키(`environment` 또는 `project`)를 사용해야 합니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringEquals": {
                   "aws:RequestTag/environment": ["production", "development"],
                   "aws:RequestTag/project": ["dataanalytics", "webapp"]
               },
               "ForAllValues:StringEquals": {
                   "aws:TagKeys": ["environment", "project"]
               }
```

```
            }
        }
    ]
}
```

이 정책은 `environment` 또는 `project` 태그 이외의 태그를 포함하거나 이러한 태그를 지정하지 않는 DB 인스턴스 생성 요청을 거부합니다. 또한 사용자는 정책의 허용된 값과 일치하는 태그 값을 지정해야 합니다.

다음 정책은 사용자가 DB 클러스터를 생성하고 생성 중에 `environment=prod` 태그를 제외하고는 어떤 태그든 적용하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestTag/environment": "prod"
                }
            }
        }
    ]
}
```

생성 시 태그 지정이 지원되는 RDS API 작업

리소스를 생성할 때 태그를 지원하는 RDS API 작업은 다음과 같습니다. 이러한 작업의 경우 리소스를 생성할 때 태그를 지정할 수 있습니다.

- CreateBlueGreenDeployment
- CreateCustomDBEngineVersion
- CreateDBCluster
- CreateDBClusterEndpoint
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- CreateDBInstanceReadReplica
- CreateDBParameterGroup
- CreateDBProxy
- CreateDBProxyEndpoint
- CreateDBSecurityGroup
- CreateDBShardGroup
- CreateDBSnapshot
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3

- `RestoreDBInstanceToPointInTime`
- `PurchaseReservedDBInstancesOffering`

AWS CLI 또는 API를 사용하여 태그가 있는 리소스를 생성하는 경우 생성 중에 리소스에 태그를 적용하는 데 `Tags` 파라미터가 사용됩니다.

이러한 API 작업의 경우 태그 지정에 실패하면 리소스가 생성되지 않고 오류와 함께 요청이 실패합니다. 리소스가 태그와 함께 생성되거나 전혀 생성되지 않으므로 의도한 태그 없이는 리소스가 생성되지 않습니다.

Amazon RDS에 대한 AWS 관리형 정책

권한 세트 및 역할에 권한을 추가하려면 정책을 직접 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 편리합니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하려면 시간과 전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용하면 됩니다. 이 정책은 일반적인 사용 사례를 다루며 사용자의 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 정보는 IAM 사용 설명서에서 AWS 관리형 정책을 참조하십시오.

AWS 서비스는 AWS 관리형 정책을 유지하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스는 때때로 추가 권한을 AWS관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 자격 증명(권한 세트 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않기 때문에 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한 AWS(은)는 여러 서비스의 직무에 대한 관리형 정책을 지원합니다. 예를 들어 `ReadOnlyAccess` AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스 권한을 제공합니다. 서비스에서 새 기능을 시작하면 AWS(이)가 새 작업 및 리소스에 대한 읽기 전용 권한을 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한 AWS 관리형 정책을 참조하십시오.

주제

AWS 관리형 정책: AmazonRDSReadOnlyAccess

이 정책은 AWS Management Console을 통해 Amazon RDS에 대한 읽기 전용 액세스를 허용합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `rds` - 보안 주체가 Amazon RDS 리소스를 설명하고 Amazon RDS 리소스의 태그를 나열하도록 허용합니다.
- `cloudwatch` - 보안 주체가 Amazon CloudWatch 지표 통계를 가져오도록 허용합니다.
- `ec2` - 보안 주체가 가용 영역 및 네트워킹 리소스를 설명하도록 허용합니다.
- `logs` - 보안 주체가 로그 그룹의 CloudWatch Logs 로그 스트림을 설명하고 CloudWatch Logs 로그 이벤트를 가져오도록 허용합니다.
- `devops-guru` - 보안 주체가 CloudFormation 스택 이름 또는 리소스 태그로 지정되는 Amazon DevOps Guru 적용 범위가 있는 리소스를 설명할 수 있습니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSReadOnlyAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSFullAccess

이 정책은 AWS Management Console을 통해 Amazon RDS에 대한 전체 액세스 권한을 제공합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `rds` – 보안 주체에게 Amazon RDS에 대한 전체 액세스 권한을 허용합니다.
- `application-autoscaling` - 보안 주체가 Application Auto Scaling 크기 조정 대상 및 정책을 설명하고 관리하도록 허용합니다.
- `cloudwatch` - 보안 주체가 CloudWatch 지표 통계를 가져오고 CloudWatch 경보를 관리하도록 허용합니다.
- `ec2` - 보안 주체가 가용 영역 및 네트워킹 리소스를 설명하도록 허용합니다.
- `logs` - 보안 주체가 로그 그룹의 CloudWatch Logs 로그 스트림을 설명하고 CloudWatch Logs 로그 이벤트를 가져오도록 허용합니다.
- `outposts` - 보안 주체가 AWS Outposts 인스턴스 유형을 가져오도록 허용합니다.
- `pi` - 보안 주체가 성능 개선 도우미 지표를 가져오도록 허용합니다.
- `sns` - 보안 주체에게 Amazon Simple Notification Service(Amazon SNS) 구독 및 주제를 허용하고 Amazon SNS 메시지를 게시하도록 허용합니다.

- devops-guru - 보안 주체가 CloudFormation 스택 이름 또는 리소스 태그로 지정되는 Amazon DevOps Guru 적용 범위가 있는 리소스를 설명할 수 있습니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSFullAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSDataFullAccess

이 정책은 특정 AWS 계정의 Aurora Serverless 클러스터에서 데이터 API 및 쿼리 편집기를 사용할 수 있는 전체 액세스 권한을 허용합니다. 이 정책은 AWS 계정가 AWS Secrets Manager에서 암호 값을 가져오도록 허용합니다.

AmazonRDSDataFullAccess 정책을 IAM 보안 인증에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- dbqms - 보안 주체에게 쿼리를 액세스, 생성, 삭제, 설명, 업데이트를 할 수 있도록 허용합니다. Database Query Metadata Service(dbqms)는 내부 전용 서비스입니다. Amazon RDS를 포함한 여러 AWS 서비스에 사용되는 AWS Management Console의 쿼리 편집기에 대해 최근 쿼리와 저장된 쿼리를 제공합니다.
- rds-data - 보안 주체가 Aurora Serverless 데이터베이스에 SQL 문을 실행하도록 허용합니다.
- secretsmanager - 보안 주체가 AWS Secrets Manager에서 암호 값을 가져오도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSDataFullAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSEnhancedMonitoringRole

이 정책은 Amazon RDS Enhanced Monitoring을 위해 Amazon CloudWatch Logs 로그에 대한 액세스를 제공합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- logs - 보안 주체가 CloudWatch Logs 로그 그룹 및 보존 정책을 생성하고 로그 그룹의 CloudWatch Logs 로그 스트림을 생성 및 설명하도록 허용합니다. 또한 보안 주체가 CloudWatch Logs 로그 이벤트를 배치하고 가져오도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSEnhancedMonitoringRole을 참조하세요.

AWS 관리형 정책: AmazonRDSPerformanceInsightsReadOnly

이 정책은 Amazon RDS DB 인스턴스 및 Amazon Aurora DB 클러스터에 대한 Amazon RDS 성능 개선 도우미에 읽기 전용 액세스를 제공합니다.

이제 이 정책에 `Sid`(문 ID)가 정책 문의 식별자로 포함됩니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `rds` - 보안 주체가 Amazon RDS DB 인스턴스와 Amazon Aurora DB 클러스터를 설명하도록 허용합니다.
- `pi` - 보안 주체가 Amazon RDS 성능 개선 도우미 API를 호출하고 성능 개선 도우미 지표에 액세스하도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSPerformanceInsightsReadOnly를 참조하세요.

AWS 관리형 정책: AmazonRDSPerformanceInsightsFullAccess

이 정책은 Amazon RDS DB 인스턴스 및 Amazon Aurora DB 클러스터에 대해 Amazon RDS 성능 개선 도우미에 전체 액세스를 제공합니다.

이제 이 정책에 `Sid`(문 ID)가 정책 문의 식별자로 포함됩니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `rds` - 보안 주체가 Amazon RDS DB 인스턴스와 Amazon Aurora DB 클러스터를 설명하도록 허용합니다.
- `pi` - 보안 주체가 Amazon RDS 성능 개선 도우미 API를 호출하고 성능 분석 보고서를 생성, 확인 및 삭제하도록 허용합니다.
- `cloudwatch` - 보안 주체가 모든 Amazon CloudWatch 지표를 나열하고 지표 데이터와 통계를 가져오도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 가이드의 AmazonRDSPerformanceInsightsFullAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSDirectoryServiceAccess

이 정책은 Amazon RDS에서 AWS Directory Service를 호출하도록 허용합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- ds - 보안 주체가 AWS Directory Service 디렉터리를 설명하고 AWS Directory Service 디렉터리에 대한 권한 부여를 제어하도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSDirectoryServiceAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSServiceRolePolicy

`AmazonRDSServiceRolePolicy` 정책을 IAM 엔터티에 연결할 수 없습니다. 이 정책은 Amazon RDS에 사용자를 대신하여 작업을 수행할 수 있도록 하는 서비스 연결 역할에 연결됩니다. 자세한 내용은 Amazon Aurora에 대한 서비스 연결 역할 권한 단원을 참조하십시오.

AWS 관리형 정책: AmazonRDSPreviewServiceRolePolicy

`AmazonRDSPreviewServiceRolePolicy`를 IAM 엔터티에 연결하면 안 됩니다. 이 정책은 Amazon RDS가 RDS DB 리소스를 대신하여 AWS 서비스를 직접 호출할 수 있는 서비스 연결 역할에 연결되어 있습니다. 자세한 내용은 Amazon RDS 미리 보기에 대한 서비스 연결 역할 단원을 참조하십시오.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `ec2` - 보안 주체가 가용 영역 및 네트워킹 리소스를 설명하도록 허용합니다.
- `secretsmanager` - 보안 주체가 AWS Secrets Manager에서 암호 값을 가져오도록 허용합니다.
- `cloudwatch`, `logs` - Amazon RDS가 CloudWatch 에이전트를 통해 DB 인스턴스 지표와 로그를 CloudWatch에 업로드할 수 있도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSDataFullAccess를 참조하세요.

AWS 관리형 정책: AmazonRDSBetaServiceRolePolicy

`AmazonRDSBetaServiceRolePolicy`를 IAM 엔터티에 연결하면 안 됩니다. 이 정책은 Amazon RDS가 RDS DB 리소스를 대신하여 AWS 서비스를 직접 호출할 수 있는 서비스 연결 역할에 연결되어 있습니다. 자세한 내용은 Amazon RDS 베타에 대한 서비스 연결 역할 권한 단원을 참조하십시오.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `ec2` - Amazon RDS가 특정 시점 복원 기능을 제공하는 DB 인스턴스에서 백업 작업을 수행할 수 있도록 허용합니다.
- `secretsmanager` - Amazon RDS가 Amazon RDS로 만든 DB 인스턴스별 암호를 관리할 수 있도록 허용합니다.
- `cloudwatch`, `logs` - Amazon RDS가 CloudWatch 에이전트를 통해 DB 인스턴스 지표와 로그를 CloudWatch에 업로드할 수 있도록 허용합니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSBetaServiceRolePolicy를 참조하세요.

AWS 관리형 정책에 대한 Amazon RDS 업데이트

이 서비스가 이러한 변경 내용을 추적하기 시작한 이후부터 Amazon RDS의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon RDS 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AWS 관리형 정책: AmazonRDSPreviewServiceRolePolicy - 기존 정책에 대한 업데이트	Amazon RDS는 `AWSServiceRoleForRDSPreview` 서비스 연결 역할의 `AmazonRDSPreviewServiceRolePolicy` 에서 `sns:Publish` 권한을 제거했습니다. 자세한 내용은 AWS 관리형 정책: AmazonRDSPreviewServiceRolePolicy 단원을 참조하십시오.	2024년 8월 7일
AWS 관리형 정책: AmazonRDSBetaServiceRolePolicy - 기존 정책에 대한 업데이트	Amazon RDS는 `AWSServiceRoleForRDSBeta` 서비스 연결 역할의 `AmazonRDSBetaServiceRolePolicy` 에서 `sns:Publish` 권한을 제거했습니다. 자세한 내용은 AWS 관리형 정책: AmazonRDSBetaServiceRolePolicy 단원을 참조하십시오.	2024년 8월 7일
AWS 관리형 정책: AmazonRDSServiceRolePolicy - 기존 정책에 대한 업데이트	Amazon RDS는 `AWSServiceRoleForRDS` 서비스 연결 역할의 `AmazonRDSServiceRolePolicy` 에서 `sns:Publish` 권한을 제거했습니다. 자세한 내용은 AWS 관리형 정책: AmazonRDS	2024년 7월 2일

변경 사항	설명	날짜
	ServiceRolePolicy 단원을 참조하십시오.	
Amazon RDS에 대한 AWS 관리형 정책 - 기존 정책에 대한 업데이트	Amazon RDS는 RDS Custom for SQL Server가 기본 데이터베이스 호스트 인스턴스 유형을 수정할 수 있도록 `AWSServiceRoleForRDSCustom` 서비스 연결 역할의 `AmazonRDSCustomServiceRolePolicy` 에 새 권한을 추가했습니다. 또한 RDS는 데이터베이스 호스트의 인스턴스 유형 정보를 가져올 수 있는 `ec2:DescribeInstanceTypes` 권한을 추가했습니다. 자세한 내용은 Amazon RDS에 대한 AWS 관리형 정책 단원을 참조하십시오.	2024년 4월 8일
Amazon RDS에 대한 AWS 관리형 정책 - 새 정책	Amazon RDS는 RDS Custom이 EC2 인스턴스 프로파일을 통해 자동화 작업 및 데이터베이스 관리 작업을 수행할 수 있도록 `AmazonRDSCustomInstanceProfileRolePolicy` 로 이름이 지정된 새 관리형 정책을 추가했습니다. 자세한 내용은 Amazon RDS에 대한 AWS 관리형 정책 단원을 참조하십시오.	2024년 2월 27일

변경 사항	설명	날짜
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS는 `AWSServiceRoleForRDS` 서비스 연결 역할의 `AmazonRDSServiceRolePolicy` 에 새로운 문 ID를 추가했습니다. 자세한 내용은 Amazon Aurora에 대한 서비스 연결 역할 권한 단원을 참조하십시오.	2024년 1월 19일
Amazon RDS에 대한 AWS 관리형 정책 - 기존 정책에 대한 업데이트	`AmazonRDSPerformanceInsightsReadOnly` 및 `AmazonRDSPerformanceInsightsFullAccess` 관리형 정책에서 이제 정책 문에 `Sid`(문 ID)를 식별자로 포함합니다. 자세한 정보는 AWS 관리형 정책: AmazonRDSPerformanceInsightsReadOnly 및 AWS 관리형 정책: AmazonRDSPerformanceInsightsFullAccess 섹션을 참조하세요.	2023년 10월 23일

변경 사항	설명	날짜
Amazon RDS에 대한 AWS 관리형 정책 - 기존 정책에 대한 업데이트	Amazon RDS에서 `AmazonRDSFullAccess` 관리형 정책에 대한 새로운 권한을 추가했습니다. 이 권한을 통해 일정 기간 동안 성능 분석 보고서를 생성, 확인 및 삭제할 수 있습니다. 성능 개선 도우미의 액세스 정책 구성에 대한 자세한 내용은 Performance Insights에 대한 액세스 정책 구성 섹션을 참조하세요.	2023년 8월 17일
Amazon RDS에 대한 AWS 관리형 정책 – 신규 정책 및 기존 정책에 대한 업데이트	Amazon RDS에서 `AmazonRDSPerformanceInsightsReadOnly` 관리형 정책 및 `AmazonRDSPerformanceInsightsFullAccess` 라는 신규 관리형 정책에 대한 새로운 권한을 추가했습니다. 이러한 권한을 통해 일정 기간 동안 성능 개선 도우미를 분석하고, 권장 사항과 함께 분석 결과를 보고, 보고서를 삭제할 수 있습니다. 성능 개선 도우미의 액세스 정책 구성에 대한 자세한 내용은 Performance Insights에 대한 액세스 정책 구성 섹션을 참조하세요.	2023년 8월 16일

변경 사항	설명	날짜
Amazon RDS에 대한 AWS 관리형 정책 -기존 정책 업데이트	Amazon RDS에서는 새로운 Amazon CloudWatch 네임스페이스 `ListMetrics` 를 `AmazonRDSFullAccess` 및 `AmazonRDSReadOnlyAccess` 에 추가했습니다. 이 네임스페이스는 Amazon RDS가 특정한 리소스 사용량 지표를 나열하는 데 필요합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch 리소스에 대한 액세스 권한 관리 개요를 참조하세요.	2023년 4월 4일

변경 사항	설명	날짜
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS는 AWS Secrets Manager와의 통합을 위해 `AWSServiceRoleForRDS` 서비스 연결 역할의 `AmazonRDSServiceRolePolicy` 에 새로운 권한을 추가했습니다. Secrets Manager에서 마스터 사용자 암호를 관리하려면 RDS와 Secrets Manager의 통합이 필요합니다. 암호는 예약된 명명 규칙을 사용하며 고객 업데이트를 제한합니다. 자세한 내용은 Amazon Aurora 및 AWS Secrets Manager를 통한 암호 관리 단원을 참조하십시오.	2022년 12월 22일
Amazon RDS에 대한 AWS 관리형 정책 - 기존 정책에 대한 업데이트	Amazon RDS는 RDS 콘솔에서 Amazon DevOps Guru를 사용할 수 있도록 `AmazonRDSFullAccess` 및 `AmazonRDSReadOnlyAccess` 관리형 정책에 새로운 권한을 추가했습니다. 이 권한은 DevOps Guru가 켜져 있는지 확인하는 데 필요합니다. 자세한 내용은 DevOps Guru for RDS에 사용되는 IAM 액세스 정책 구성 단원을 참조하십시오.	2022년 12월 19일

변경 사항	설명	날짜
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS에서는 `PutMetricData` 용 `AmazonRDSPreviewServiceRolePolicy` 에 새로운 Amazon CloudWatch 네임스페이스를 추가했습니다. 이 네임스페이스는 Amazon RDS가 리소스 사용량 지표를 게시하는 데 필요합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 조건 키를 사용하여 CloudWatch 네임스페이스에 대한 액세스 제한을 참조하세요.	2022년 6월 7일
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS에서는 `PutMetricData` 용 `AmazonRDSBetaServiceRolePolicy` 에 새로운 Amazon CloudWatch 네임스페이스를 추가했습니다. 이 네임스페이스는 Amazon RDS가 리소스 사용량 지표를 게시하는 데 필요합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 조건 키를 사용하여 CloudWatch 네임스페이스에 대한 액세스 제한을 참조하세요.	2022년 6월 7일

변경 사항	설명	날짜
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS에서는 `PutMetricData` 용 `AWSServiceRoleForRDS` 에 새로운 Amazon CloudWatch 네임스페이스를 추가했습니다. 이 네임스페이스는 Amazon RDS가 리소스 사용량 지표를 게시하는 데 필요합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 조건 키를 사용하여 CloudWatch 네임스페이스에 대한 액세스 제한을 참조하세요.	2022년 4월 22일
Amazon RDS에 대한 AWS 관리형 정책 - 새 정책	Amazon RDS에서는 Amazon RDS가 DB 인스턴스를 대신하여 AWS를 호출할 수 있도록 `AmazonRDSPerformanceInsightsReadOnly` 라는 새로운 관리형 정책을 추가했습니다. 성능 개선 도우미의 액세스 정책 구성에 대한 자세한 내용은 Performance Insights에 대한 액세스 정책 구성 섹션을 참조하세요.	2022년 3월 10일

변경 사항	설명	날짜
Amazon Aurora에 대한 서비스 연결 역할 권한 -기존 정책 업데이트	Amazon RDS에서는 `PutMetricData` 용 `AWSServiceRoleForRDS` 에 새로운 Amazon CloudWatch 네임스페이스를 추가했습니다. Amazon DocumentDB(MongoDB 호환) 및 Amazon Neptune에서 CloudWatch 지표를 게시하려면 이러한 네임스페이스가 필요합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 조건 키를 사용하여 CloudWatch 네임스페이스에 대한 액세스 제한을 참조하세요.	2022년 3월 4일
Amazon RDS에서 변경 사항 추적 시작	Amazon RDS가 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 10월 26일

교차 서비스 혼동된 대리자 문제 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에 작업을 수행하도록 강요할 수 있는 보안 문제입니다. AWS에서는 교차 서비스 가장으로 인해 혼동된 대리자 문제가 발생할 수 있습니다.

교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 직접적으로 호출할 때 발생할 수 있습니다. 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 호출 서비스를 조작할 수 있습니다. 이를 방지하기 위해 AWS에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다. 자세한 내용은 IAM 사용 설명서의 혼동된 대리자 문제를 참조하세요.

Amazon RDS가 다른 서비스에 제공하는 리소스에 대한 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하는 것이 좋습니다.

경우에 따라 `aws:SourceArn` 값에 계정 ID가 포함되지 않습니다. Amazon S3 버킷에 Amazon 리소스 이름(ARN)을 사용하는 경우를 예로 들 수 있습니다. 이러한 경우 두 전역 조건 컨텍스트 키를 모두 사용하여 권한을 제한해야 합니다. 경우에 따라 두 전역 조건 컨텍스트 키를 모두 사용하고 `aws:SourceArn` 값에 계정 ID가 포함됩니다. 이러한 경우 `aws:SourceAccount` 값과 `aws:SourceArn`의 계정이 동일한 정책 문에서 사용될 때 동일한 계정 ID를 사용하는지 확인합니다. 하나의 리소스만 교차 서비스 액세스와 연결되게 하려는 경우 `aws:SourceArn`을 사용합니다. 지정된 AWS 계정의 모든 리소스가 교차 서비스 사용과 연결되게 하려는 경우 `aws:SourceAccount`를 사용합니다.

`aws:SourceArn`의 값이 Amazon RDS 리소스 유형에 대한 ARN인지 확인하세요. 자세한 내용은 Amazon RDS의 Amazon 리소스 이름(ARN) 섹션을 참조하세요.

혼동된 대리인 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다. 경우에 따라 리소스의 전체 ARN을 모르거나 여러 리소스를 지정하고 있을 수 있습니다. 이러한 경우 ARN의 알 수 없는 부분에 대해 와일드카드(*)와 함께 `aws:SourceArn` 전역 컨텍스트 조건 키를 사용하세요. 예를 들면, `arn:aws:rds:*:123456789012:*`입니다.

다음 예에서는 Amazon RDS에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다.

JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하는 정책의 추가 예는 다음 섹션을 참조하세요.

- Amazon SNS 주제에 알림을 게시할 권한 부여
- Amazon S3 버킷에 대한 액세스 권한 설정(PostgreSQL 가져오기)
- Amazon S3 버킷에 대한 액세스 권한 설정(PostgreSQL 내보내기)

IAM 데이터베이스 인증

AWS Identity and Access Management(IAM) 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다. IAM 데이터베이스 인증은 Aurora MySQL, Aurora PostgreSQL과 함께 작동합니다. 이러한 인증 방식은 DB 클러스터에 연결할 때 암호를 사용할 필요 없습니다. 대신에 인증 토큰을 사용합니다.

인증 토큰이란 요청이 있을 때 Amazon Aurora가 생성하는 고유 문자열입니다. 인증 토큰은 AWS 서명 버전 4를 통해 생성됩니다. 각 토큰의 수명은 15분입니다. 인증을 외부에서 IAM을 사용해 관리하기 때문에 사용자 자격 증명을 데이터베이스에 저장할 필요도 없습니다. 또한 표준 데이터베이스 인증 방식도 사용 가능합니다. 토큰은 인증에만 사용되며 설정된 후에는 세션에 영향을 주지 않습니다.

IAM 데이터베이스 인증은 다음과 같은 이점이 있습니다.

- 데이터베이스를 오가는 네트워크 트래픽은 SSL(Secure Sockets Layer) 또는 TLS(Transport Layer Security)를 통해 암호화됩니다. Amazon Aurora에서 SSL/TLS를 사용하는 방법에 대한 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 단원을 참조하십시오.
- 데이터베이스 리소스에 대한 액세스는 DB 클러스터에서 개별적으로 관리할 필요 없이 IAM을 통해 중앙에서 관리할 수 있습니다.
- Amazon EC2에서 실행되는 애플리케이션의 경우, 암호가 아닌 EC2 인스턴스용 프로파일 자격 증명을 사용해 데이터베이스에 액세스하기 때문에 보안을 더욱 강화하는 효과가 있습니다.

일반적으로 애플리케이션에서 초당 200개 미만의 연결을 생성하고 애플리케이션 코드에서 사용자 이름과 암호를 직접 관리하지 않으려는 경우 IAM 데이터베이스 인증을 사용하는 것이 좋습니다.

Amazon Web Services(AWS) JDBC 드라이버는 IAM 데이터베이스 인증을 지원합니다. 자세한 내용은 Amazon Web Services (AWS) JDBC Driver GitHub repository에서 AWS IAM Authentication Plugin을 참조하세요.

Amazon Web Services(AWS) Python 드라이버는 IAM 데이터베이스 인증을 지원합니다. 자세한 내용은 Amazon Web Services (AWS) Python Driver GitHub repository에서 AWS IAM Authentication Plugin을 참조하세요.

다음 주제를 탐색하여 DB 인증을 위한 IAM을 설정하는 프로세스를 알아봅니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

- IAM 인증을 사용하여 DB 클러스터에 연결

리전 및 버전 사용 가능 여부

기능 가용성 및 해당 지원은 각 Aurora 데이터베이스 엔진의 특정 버전 및 AWS 리전에 따라 다릅니다. Aurora 및 IAM 데이터베이스 인증을 통한 버전 및 리전에서 사용 가능한 버전에 대한 자세한 내용은 IAM 데이터베이스 인증을 지원하는 리전 및 Aurora DB 엔진 섹션을 참조하세요.

Aurora MySQL의 경우 지원되는 모든 DB 인스턴스 클래스는 db.t2.small 및 db.t3.small 을 제외한 IAM 데이터베이스 인증을 지원합니다. 지원되는 DB 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스에 지원되는 DB 엔진 섹션을 참조하세요.

CLI 및 SDK 지원

AWS CLI 및 다음 언어별 AWS SDK에서 IAM 데이터베이스 인증을 사용할 수 있습니다.

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

IAM 데이터베이스 인증 방식의 제한 사항

IAM 데이터베이스 인증을 사용하는 경우, 다음 한도가 적용됩니다.

- IAM 데이터베이스 인증은 초당 200개의 연결로 연결을 스로틀링합니다.

  동일한 인증 토큰을 사용하는 연결은 제한되지 않습니다. 가능하면 인증 토큰을 재사용하는 것이 좋습니다.
- 현재 IAM 데이터베이스 인증은 모든 전역 조건 컨텍스트 키를 지원하지 않습니다.

  전역 조건 컨텍스트 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

- PostgreSQL의 경우 IAM 역할(`rds_iam`)이 사용자(마스터 사용자 RDS 포함)에 추가되면 IAM 인증이 암호 인증보다 우선하므로, 사용자는 IAM 사용자로 로그인해야 합니다.
- Aurora PostgreSQL의 경우 IAM 인증을 사용하여 복제 연결을 설정할 수 없습니다.
- 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드를 사용할 수 없습니다.
- CloudWatch와 CloudTrail은 IAM 인증을 로깅하지 않습니다. 이러한 서비스는 데이터베이스 연결을 활성화하도록 IAM 역할을 승인하는 `generate-db-auth-token` API 직접 호출을 추적하지 않습니다.
- IAM DB 권한 부여에는 데이터베이스 클러스터의 컴퓨팅 리소스가 필요합니다. 신뢰할 수 있는 연결을 위해 데이터베이스에 300MiB 및 1,000MiB 사이의 추가 메모리가 있어야 합니다. 워크로드에 필요한 메모리를 보려면 IAM DB 인증을 사용하기 전과 후에 확장 모니터링 프로세스 목록에서 RDS 프로세스에 대한 RES 열을 비교합니다. RDS 콘솔에서 OS 지표 보기을(를) 참조하세요.

  버스트 가능 클래스 인스턴스를 사용하는 경우 버퍼 및 캐시와 같은 다른 파라미터에서 사용되는 메모리를 동일한 양만큼 줄여 메모리가 부족하지 않도록 합니다.
- Aurora MySQL의 경우 IAM 인증으로 구성한 데이터베이스 사용자에 암호 기반 인증을 사용할 수 없습니다.
- 엔진에 상관없이 RDS on Outposts에서는 IAM DB 인증이 지원되지 않습니다.

IAM 데이터베이스 인증에 대한 권장 사항

IAM 데이터베이스 인증을 사용할 경우 다음을 따르는 게 좋습니다.

- 애플리케이션에 초당 200개 미만의 새 IAM 데이터베이스 인증 연결이 필요한 경우에는 IAM 데이터베이스 인증 방식을 사용합니다.

  Amazon Aurora에서 작동하는 데이터베이스 엔진은 초당 인증 횟수에 제한이 없습니다. 하지만 IAM 데이터베이스 인증 방식을 사용할 때는 애플리케이션이 인증 토큰을 생성해야 합니다. 이렇게 생성된 토큰은 애플리케이션이 DB 클러스터에 연결하는 데 사용됩니다. 초당 허용되는 새 연결의 최대 수를 초과하면 IAM 데이터베이스 인증에 오버헤드가 추가로 발생하여 연결 병목 현상이 발생할 수 있습니다.

  지속적인 연결 생성을 줄이려면 애플리케이션에서 연결 풀링을 사용하는 것이 좋습니다. 이렇게 하면 IAM DB 인증으로 인한 오버헤드를 줄이고 애플리케이션이 기존 연결을 재사용할 수 있습니다. 또는 이러한 사용 사례에 RDS 프록시를 사용하는 것도 좋습니다. RDS 프록시에는 추가 비용이 부과됩니다. RDS 프록시 요금을 참조하세요.

- IAM 데이터베이스 인증 토큰의 크기는 IAM 태그 수, IAM 서비스 정책, ARN 길이, 기타 IAM 및 데이터베이스 속성 등 여러 요소에 따라 달라집니다. 이 토큰의 최소 크기는 일반적으로 약 1KB지만 더 클 수도 있습니다. 이 토큰은 IAM 인증을 사용하는 데이터베이스에 대한 연결 문자열에서 암호로 사용되므로 데이터베이스 드라이버(예: ODBC) 및/또는 어떤 도구도 크기로 인해 이 토큰을 제한하거나 자르지 않도록 해야 합니다. 토큰이 잘리면 데이터베이스와 IAM에서 수행하는 인증 검증이 실패합니다.
- IAM 데이터베이스 인증 토큰을 생성할 때 임시 보안 인증 정보를 사용하는 경우, IAM 데이터베이스 인증 토큰을 사용하여 연결을 요청할 때 임시 보안 인증 정보가 여전히 유효해야 합니다.

지원되지 않는 AWS 전역 조건 컨텍스트 키

IAM 데이터베이스 인증은 다음과 같은 AWS 전역 조건 컨텍스트 키의 하위 집합을 지원하지 않습니다.

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

자세한 내용은 IAM 사용 설명서의 AWS 글로벌 조건 컨텍스트 키를 참조하세요.

IAM 데이터베이스 인증의 활성화 및 비활성화

DB 클러스터에서는 기본적으로 IAM 데이터베이스 인증이 비활성화되어 있습니다. IAM 데이터베이스 인증은 AWS Management Console, AWS CLI 또는 API를 사용하여 활성화하거나 비활성화할 수 있습니다.

다음 작업 중 하나를 수행할 때 IAM 데이터베이스 인증을 활성화할 수 있습니다.

- IAM 데이터베이스 인증이 활성화된 새 DB 클러스터를 생성하려면 Amazon Aurora DB 클러스터 생성 단원을 참조하세요.
- IAM 데이터베이스 인증을 사용하도록 DB 클러스터를 수정하려면 Amazon Aurora DB 클러스터 수정 단원을 참조하세요.
- IAM 데이터베이스 인증이 활성화된 스냅샷에서 DB 클러스터를 복원하려면 DB 클러스터 스냅샷에서 복원 섹션을 참조하세요.

- IAM 데이터베이스 인증이 사용 설정된 시점으로 DB 클러스터를 복원하려면 지정된 시간으로 DB 클러스터 복원 섹션을 참조하세요.

콘솔

각 생성 또는 수정 워크플로에는 IAM 데이터베이스 인증을 활성화하거나 비활성화할 수 있는 데이터베이스 인증(Database authentication) 섹션이 있습니다. 이 섹션에서 암호 및 IAM 데이터베이스 인증(Password and IAM database authentication)을 선택하여 IAM 데이터베이스 인증을 활성화합니다.

기존 DB 클러스터에서 IAM 데이터베이스 인증을 활성화하거나 비활성화하려면

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 수정하려는 DB 클러스터를 선택합니다.

> **Note**
> DB 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다. 호환성 요구 사항은 리전 및 버전 사용 가능 여부 단원을 참조하십시오.

4. 수정을 선택합니다.
5. 이 [데이터베이스 인증(Database authentication)] 섹션에서 [암호 및 IAM 데이터베이스 인증(Password and IAM database authentication)]을 선택하여 IAM 데이터베이스 인증을 활성화합니다. 암호 인증 또는 암호 및 Kerberos 인증을 선택하여 IAM 인증을 비활성화합니다.
6. 또한 CloudWatch Logs에 IAM DB 인증 로그 게시를 사용하도록 선택할 수 있습니다. 로그 내보내기에서 iam-db-auth-error 로그 옵션을 선택합니다. CloudWatch Logs에 로그를 게시하면 스토리지가 사용되며 해당 스토리지에 대한 요금이 부과됩니다. 더 이상 필요하지 않은 CloudWatch Logs는 삭제해야 합니다.
7. [Continue]를 선택합니다.
8. 변경 사항을 즉시 적용하려면 수정 예약(Scheduling of modifications) 섹션에서 즉시(Immediately)를 선택합니다.
9. 클러스터 수정을 선택합니다.

AWS CLI

AWS CLI를 사용하여 새로운 DB 클러스터를 IAM 인증 방식으로 생성하려면 `create-db-cluster` 명령을 사용하십시오. `--enable-iam-database-authentication` 옵션을 지정합니다.

기존 DB 클러스터를 업데이트하여 IAM 인증을 사용하게 하거나 사용하지 않게 하려면 AWS CLI 명령 `modify-db-cluster`를 사용합니다. 상황에 따라 `--enable-iam-database-authentication` 또는 `--no-enable-iam-database-authentication` 옵션을 지정합니다.

> **Note**
>
> DB 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다. 호환성 요구 사항은 리전 및 버전 사용 가능 여부 단원을 참조하십시오.

기본적으로 Aurora는 다음 유지 관리 기간에 수정 작업을 수행합니다. 이러한 기본 설정을 무시하고 IAM DB 인증을 최대한 빠르게 활성화하려면 `--apply-immediately` 파라미터를 사용합니다.

DB 클러스터를 복원하는 경우에는 다음 AWS CLI 명령 중 하나를 사용하십시오.

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

IAM 데이터베이스 인증은 기본적으로 원본 스냅샷으로 기본 설정됩니다. 이 설정을 변경하려면 상황에 따라 `--enable-iam-database-authentication` 또는 `--no-enable-iam-database-authentication` 옵션을 설정합니다.

RDS API

API를 사용하여 새로운 DB 인스턴스를 IAM 인증 방식으로 생성하려면 API 작업 `CreateDBCluster`를 사용하십시오. `EnableIAMDatabaseAuthentication` 파라미터를 `true`로 설정합니다.

기존 DB 클러스터를 업데이트하여 IAM 인증을 사용하게 하거나 사용하지 않게 하려면 API 작업 `ModifyDBCluster`를 사용합니다. `EnableIAMDatabaseAuthentication` 파라미터를 `true`로 설정하여 IAM 인증을 활성화하거나, `false`로 설정하여 비활성화합니다.

> **Note**
>
> DB 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다. 호환성 요구 사항은 리전 및 버전 사용 가능 여부 단원을 참조하십시오.

DB 클러스터를 복원하는 경우에는 다음 API 작업 중 하나를 사용하십시오.

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

IAM 데이터베이스 인증은 기본적으로 원본 스냅샷으로 기본 설정됩니다. 이 설정을 변경하려면 `EnableIAMDatabaseAuthentication` 파라미터를 `true`로 설정하여 IAM 인증을 활성화하거나, 혹은 `false`로 설정하여 비활성화합니다.

IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용

사용자 또는 역할이 DB 클러스터에 연결할 수 있도록 허용하려면 IAM 정책을 생성해야 합니다. 그런 다음 정책을 권한 세트 또는 역할에 연결합니다.

> **Note**
>
> IAM 정책에 대한 자세한 정보는 Amazon Aurora의 자격 증명 및 액세스 관리 단원을 참조하십시오.

다음은 사용자가 IAM 데이터베이스 인증 방식을 사용해 DB 클러스터에 연결할 수 있도록 허용하는 정책 예제입니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
```

```
        ],
        "Resource": [
            "arn:aws:rds-db:us-east-2:1234567890:dbuser:db-ABCDEFGHIJKL01234/
db_user"
        ]
      }
   ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
        "Effect": "Allow",
        "Action": [
            "rds-db:connect"
        ],
        "Resource": [
            "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
        ]
      }
   ]
}
```

> **Important**
>
> 관리자 권한이 있는 사용자는 IAM 정책에서 명시적 권한 없이도 DB 클러스터에 액세스할 수 있습니다. DB 클러스터에 대한 관리자 액세스를 제한하고 싶은 경우에는 더 적은 적정 권한을 가진 IAM 역할을 생성하고 이를 관리자에게 할당할 수 있습니다.

> **Note**
>
> `rds-db:` 접두사를 `rds:`로 시작하는 다른 RDS API 작업 접두사와 혼동하지 마십시오. `rds-db:` 접두사와 `rds-db:connect` 작업은 IAM 데이터베이스 인증 전용입니다. 다른 컨텍스트에서는 유효하지 않습니다.

위의 예제 정책에는 다음 요소와 함께 단일 문이 포함되어 있습니다.

- `Effect` – `Allow`를 지정하여 DB 클러스터에 대한 액세스를 부여합니다. 액세스를 명시적으로 허용하지 않으면 액세스가 기본적으로 거부됩니다.
- `Action` – `rds-db:connect`를 지정하여 DB 클러스터에 대한 연결을 허용합니다.
- `Resource` – 하나의 DB 클러스터의 한 데이터베이스 계정을 기술하는 Amazon 리소스 이름(ARN)을 지정합니다. ARN 형식은 다음과 같습니다.

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

  이 형식에서 다음 항목을 교체합니다.

  - *`region`*은 클러스터의 AWS 리전입니다. 정책 예제에서 사용되는 AWS 리전은 `us-east-2`입니다.
  - *`account-id`*은 DB 클러스터의 AWS 계정 번호입니다. 정책 예제에서 사용되는 계정 번호는 `1234567890`입니다. 사용자는 DB 클러스터의 계정과 동일한 계정에 있어야 합니다.

    크로스 계정 액세스를 수행하려면 DB 클러스터의 계정에서 위에 있는 정책을 사용하여 IAM 역할을 생성하고 다른 계정이 해당 역할을 맡도록 허용합니다.
  - *`DbClusterResourceId`*는 DB 클러스터의 식별자입니다. 이 식별자는 AWS 리전에 고유하며, 절대로 바뀌지 않습니다. 정책 예제에서 사용되는 식별자는 `cluster-ABCDEFGHIJKL01234`입니다.

    Amazon Aurora용 AWS Management Console에서 DB 클러스터 리소스 ID를 찾으려면 DB 클러스터를 선택하여 세부 정보를 확인하세요. 그런 다음 구성 탭을 선택합니다. 그러면 리소스 ID가 구성 섹션에 표시됩니다.

    그 밖에 다음과 같이 AWS CLI 명령을 사용하여 현재 AWS 리전에 속한 모든 DB 클러스터의 식별자와 리소스 ID 목록을 조회하는 방법도 있습니다.

```
aws rds describe-db-clusters --query "DBClusters[*].
[DBClusterIdentifier,DbClusterResourceId]"
```

> **Note**
>
> RDS 프록시를 통해 데이터베이스에 연결하는 경우, 프록시 리소스 ID(예: `prx-ABCDEFGHIJKL01234`)를 지정합니다. RDS 프록시에서 IAM 데이터베이스 인증을 사용하는 방법에 대한 자세한 내용은 IAM 인증을 사용하여 프록시에 연결 단원을 참조하십시오.

- *db-user-name*은 IAM 인증과 연결할 데이터베이스 계정 이름입니다. 정책 예제에서 사용되는 데이터베이스 계정은 `db_user`입니다.

다른 ARN을 구성하여 다양한 액세스 패턴을 지원할 수 있습니다. 다음 정책에서는 DB 클러스터에서 서로 다른 데이터베이스 계정 2개에 대한 액세스를 허용합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

다음 정책에서는 "*" 문자를 사용하여 특정 AWS 계정과 AWS 리전의 모든 DB 클러스터 및 데이터베이스 계정을 일치시킵니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:*/*"
            ]
        }
    ]
}
```

다음 정책은 특정 AWS 계정과 AWS 리전의 모든 DB 클러스터를 일치시킵니다. 하지만 정책에 따라 `jane_doe` 데이터베이스 계정을 가지고 있는 DB 클러스터에게만 액세스 권한이 부여됩니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

사용자 또는 역할은 데이터베이스 사용자가 액세스할 수 있는 데이터베이스에만 액세스할 수 있습니다. 예를 들어 DB 클러스터에 이름이 dev인 데이터베이스와 test인 데이터베이스가 있다고 가정하겠습니다. 이때 데이터베이스 사용자인 `jane_doe`가 dev에 대한 액세스 권한만 가지고 있다면 사용자 `jane_doe`와 함께 해당 DB 클러스터에 액세스하는 모든 사용자 또는 역할도 dev 액세스 권한만 갖게 됩니다. 이러한 액세스 제한은 테이블, 뷰 등 다른 데이터베이스 객체에 대해서도 똑같이 적용됩니다.

관리자는 지정된 필요한 리소스에서 특정 API 작업을 수행할 수 있는 권한을 엔티티에 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 권한 세트 또는 역할에 이러한 정책을 연결해야 합니다. 정책에 대한 예시는 Amazon Aurora 자격 증명 기반 정책 예 단원을 참조하십시오.

권한 세트 또는 역할에 IAM 정책 연결

데이터베이스 인증을 위한 IAM 정책을 생성하였으면 이제 정책을 권한 세트 또는 역할에 연결해야 합니다. 이번 주제에 대한 자습서는 IAM 사용자 안내서의 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

자습서를 읽어보면 이번 단원에서 소개하는 정책 예제 중 한 가지를 출발점으로 자신만의 요건에 따라 지정하여 사용할 수 있습니다. 자습서를 끝까지 따르다 보면 연결된 정책을 통해 `rds-db:connect` 작업이 가능한 권한 세트를 얻게 될 것입니다.

> **Note**
>
> 여러 권한 세트 또는 역할을 동일한 데이터베이스 사용자 계정에 매핑할 수 있습니다. 예를 들어 IAM 정책이 다음과 같은 리소스 ARN을 지정하였다고 가정하겠습니다.
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/
> jane_doe
> ```
>
> Jane, Bob 및 Diego에게 정책을 연결하면 각 사용자는 `jane_doe` 데이터베이스 계정을 사용하여 지정된 DB 클러스터에 연결할 수 있습니다.

IAM 인증을 사용하여 데이터베이스 계정 생성

IAM 데이터베이스 인증 방식에서는 데이터베이스 암호를 사용자 계정에 할당할 필요 없습니다. 데이터베이스 계정에 매핑되어 있는 사용자를 제거할 경우에는 `DROP USER` 문으로 데이터베이스 계정도 제거해야 합니다.

> **Note**
>
> IAM 인증에 사용되는 사용자 이름은 데이터베이스에 있는 사용자 이름과 대소문자가 일치해야 합니다.

주제

- Aurora MySQL에서 IAM 인증 사용
- Aurora PostgreSQL에서 IAM 인증 사용

Aurora MySQL에서 IAM 인증 사용

Aurora MySQL에서는 AWS에서 제공하는 플러그인인 `AWSAuthenticationPlugin`에서 인증을 처리합니다. 이 플러그인은 IAM과 원활하게 연동되어 사용자를 인증합니다. 마스터 사용자 또는 사용자

를 생성하고 권한을 부여할 수 있는 다른 사용자로 DB 클러스터에 연결합니다. 연결한 후 다음 예제와 같이 CREATE USER 문을 실행합니다.

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

IDENTIFIED WITH 절을 사용하면 Aurora MySQL이 AWSAuthenticationPlugin을 통해 데이터베이스 계정(jane_doe)을 인증할 수 있습니다. AS 'RDS' 절은 인증 방법을 참조합니다. 지정한 데이터베이스 사용자 이름이 IAM 데이터베이스 액세스에 대한 IAM 정책의 리소스와 같은지 확인합니다. 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 섹션을 참조하세요.

> **Note**
>
> 다음과 같은 메시지가 표시되면 현재 DB 클러스터에서 AWS 제공 플러그인을 사용할 수 없는 것입니다.
> ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded
> 위와 같은 오류 문제를 해결하려면 지원되는 구성을 사용하고 있는지, 그리고 DB 클러스터에서 IAM 데이터베이스 인증이 활성화되어 있는지 확인하십시오. 자세한 내용은 리전 및 버전 사용 가능 여부 및 IAM 데이터베이스 인증의 활성화 및 비활성화 단원을 참조하십시오.

AWSAuthenticationPlugin을 사용하여 계정을 생성한 이후 계정 관리 방법은 다른 데이터베이스 계정과 동일합니다. 예를 들어 GRANT 및 REVOKE 문으로 계정 권한을 수정하거나, 혹은 ALTER USER 문으로 여러 가지 계정 속성을 변경할 수 있습니다.

IAM을 사용하는 경우 데이터베이스 네트워크 트래픽은 SSL/TLS를 사용하여 암호화됩니다. SSL 연결을 허용하려면 다음 명령을 사용해 사용자 계정을 수정합니다.

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Aurora PostgreSQL에서 IAM 인증 사용

Aurora PostgreSQL에서 IAM 인증을 사용하려면 마스터 사용자 또는 사용자를 생성하고 권한을 부여할 수 있는 다른 사용자로 DB 클러스터에 연결해야 합니다. 연결한 후 데이터베이스 사용자를 생성하고 다음 예제에 나온 것처럼 사용자에게 rds_iam 역할을 부여합니다.

```
CREATE USER db_userx;
```

```
GRANT rds_iam TO db_userx;
```

지정한 데이터베이스 사용자 이름이 IAM 데이터베이스 액세스에 대한 IAM 정책의 리소스와 같은지 확인합니다. 자세한 내용은 IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 단원을 참조하십시오. IAM 인증을 사용하려면 rds_iam 역할을 부여해야 합니다. 중첩된 멤버십이나 역할의 간접 부여도 사용할 수 있습니다.

PostgreSQL 데이터베이스 사용자는 IAM 또는 Kerberos 인증 중 하나를 사용할 수 있지만 둘 다 사용할 수는 없으므로 이 사용자에게 rds_ad 역할을 할당할 수 없습니다. 이는 중첩된 멤버십에도 적용됩니다. 자세한 내용은 7단계: Kerberos 보안 주체를 위한 PostgreSQL 사용자 생성 단원을 참조하십시오.

IAM 인증을 사용하여 DB 클러스터에 연결

IAM 데이터베이스 인증 방식에서는 DB 클러스터에 연결할 때 인증 토큰을 사용합니다. 인증 토큰이란 암호 대신 사용하는 문자열을 말합니다. 인증 토큰은 생성 후 15분 동안만 유효하며 이 시간이 지나면 만료됩니다. 만료된 토큰을 사용하여 연결하려고 하면 연결 요청이 거부됩니다.

모든 인증 토큰은 AWS 서명 버전 4를 사용하여 유효한 서명이 있어야 합니다. (자세한 내용은 AWS 일반 참조의 서명 버전 4 서명 프로세스를 참조하세요.) AWS SDK for Java 또는 AWS SDK for Python (Boto3)와 같은 AWS CLI 및 AWS SDK는 사용자가 생성한 각 토큰에 자동으로 서명할 수 있습니다.

AWS Lambda와 같은 AWS 서비스에서 Amazon Aurora에 연결할 때 인증 토큰을 사용할 수 있습니다. 토큰을 사용하면 코드에 암호를 넣지 않아도 됩니다. 그 밖에 AWS SDK를 사용하여 인증 토큰을 프로그래밍 방식으로 생성하고 프로그래밍 방식으로 서명하는 방법도 있습니다.

IAM 인증 토큰에 서명까지 마쳤으면 이제 Aurora DB 클러스터에 연결할 수 있습니다. 다음 섹션에서는 명령줄 도구 또는 AWS SDK(예: AWS SDK for Java 또는 AWS SDK for Python (Boto3))를 사용하여 연결하는 방법에 대해 알아보겠습니다.

자세한 내용은 다음 블로그 게시물을 참조하십시오.

- IAM 인증을 사용하여 SQL Workbench/J를 Aurora MySQL 또는 Amazon RDS for MySQL에 연결
- Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

주제

- AWS 드라이버와 함께 IAM 인증을 사용하여 DB 클러스터에 연결
- 명령줄에서 IAM 인증을 사용하여 DB 클러스터에 연결: AWS CLI 및 mysql 클라이언트
- 명령줄: AWS CLI 및 psql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결
- IAM 인증 및 AWS SDK for .NET를 사용하여 DB 클러스터에 연결
- IAM 인증 및 AWS SDK for Go를 사용하여 DB 클러스터에 연결
- IAM 인증 및 AWS SDK for Java를 사용하여 DB 클러스터에 연결
- IAM 인증 및 AWS SDK for Python (Boto3)를 사용하여 DB 클러스터에 연결

AWS 드라이버와 함께 IAM 인증을 사용하여 DB 클러스터에 연결

더 빠른 전환 및 장애 조치 시간, AWS Secrets Manager, AWS Identity and Access Management(IAM) 및 페더레이션 ID를 사용한 인증을 지원하도록 설계된 AWS 드라이버 제품군입니다. AWS 드라이버는 DB 클러스터 상태 모니터링과 클러스터 토폴로지 파악을 통해 새 라이터를 결정합니다. 이 접근 방식은 전환 및 장애 조치 시간을 오픈 소스 드라이버의 경우 수십 초였던 것에 비해 10초 미만으로 단축합니다.

AWS 드라이버에 대한 자세한 내용은 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터의 해당 언어 드라이버를 참조하세요.

명령줄에서 IAM 인증을 사용하여 DB 클러스터에 연결: AWS CLI 및 mysql 클라이언트

아래 설명과 같이 AWS CLI 및 `mysql` 명령줄 도구를 사용하여 명령줄에서 Aurora DB 클러스터로 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

> **Note**
>
> IAM 인증을 통한 SQL Workbench/J를 사용하여 데이터베이스에 연결하는 방법에 대한 자세한 내용은 블로그 게시물 IAM 인증을 사용하여 SQL Workbench/J를 Aurora MySQL 또는 Amazon RDS for MySQL에 연결을 참조하세요.

주제

- IAM 인증 토크 생성
- DB 클러스터에 연결

IAM 인증 토크 생성

다음은 AWS CLI를 사용하여 서명이 되어 있는 인증 토큰을 생성하는 방법을 설명한 예제입니다.

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

위의 예제에서 각 파라미터는 다음과 같습니다.

- `--hostname` – 액세스할 DB 클러스터의 호스트 이름입니다.
- `--port` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `--region` - DB 클러스터가 실행되는 AWS 리전입니다.
- `--username` – 액세스할 데이터베이스 계정입니다.

토큰에서 가장 앞의 일부 문자는 다음과 같은 모습입니다.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

DB 클러스터에 연결

일반적인 연결 형식은 다음과 같습니다.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-cleartext-plugin --user=userName --password=authToken
```

파라미터는 다음과 같습니다.

- `--host` – 액세스할 DB 클러스터의 호스트 이름입니다.
- `--port` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `--ssl-ca` - 퍼블릭 키를 포함하는 SSL 인증서 파일의 전체 경로

  자세한 내용은 [Aurora MySQL DB 클러스터에 대한 TLS 연결](#) 단원을 참조하십시오.

  SSL 인증서를 다운로드하려면 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화](#) 섹션을 참조하세요.
- `--enable-cleartext-plugin` – 현재 연결에서 `AWSAuthenticationPlugin`을 사용하도록 지정하는 값입니다.

  MariaDB 클라이언트를 사용하는 경우 `--enable-cleartext-plugin` 옵션이 필요하지 않습니다.
- `--user` – 액세스할 데이터베이스 계정입니다.
- `--password` – 서명된 IAM 인증 토큰입니다.

인증 토큰은 수백 자의 문자로 구성됩니다. 그렇기 때문에 명령줄에서는 다루지 불편할 수도 있습니다. 이러한 문제를 해결하기 위해 토큰을 환경 변수로 저장한 후 연결할 때 이 변수를 사용하는 것도 한 가지 방법입니다. 다음은 이러한 문제 해결 방법을 설명한 예제입니다. 이 예에서 */sample_dir/*는 퍼블릭 키를 포함하는 SSL 인증서 파일의 전체 경로입니다.

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
```

```
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

`AWSAuthenticationPlugin`을 사용하여 연결할 때는 SSL을 통해 보안을 유지합니다. 이러한 보안 여부를 확인하려면 `mysql>` 명령 프롬프트에 다음과 같이 입력합니다.

```
show status like 'Ssl%';
```

그러면 출력 시 다음과 같이 자세하게 표시됩니다.

```
+---------------+-------------+
| Variable_name | Value                                                                                                                                                                                                                                |
+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA                                                                                                                                                                                                                           |
| ...           | ...
| Ssl_version   | TLSv1.1                                                                                                                                                                                                                              |
| ...           | ...
+-----------------------------+
```

프록시를 통해 DB 클러스터 연결하려는 경우 IAM 인증을 사용하여 프록시에 연결을 참조하세요.

명령줄: AWS CLI 및 psql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS CLI 및 psql 명령줄 도구를 사용하여 명령줄에서 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용

- IAM 인증을 사용하여 데이터베이스 계정 생성

> **Note**
>
> IAM 인증을 통한 pgAdmin을 사용하여 데이터베이스에 연결하는 방법에 대한 자세한 내용은 블로그 게시물 Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL을 참조하세요.

주제

IAM 인증 토크 생성

인증 토큰은 수백 자의 문자로 구성되므로 명령줄에서는 다루기 불편할 수 있습니다. 이러한 문제를 해결하기 위해 토큰을 환경 변수로 저장한 후 연결할 때 이 변수를 사용하는 것도 한 가지 방법입니다. 다음 예제는 AWS CLI에서 `generate-db-auth-token` 명령을 사용하여 서명된 인증 토큰을 받고 이를 `PGPASSWORD` 환경 변수에 저장하는 방법을 보여 줍니다.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

예제에서 `generate-db-auth-token` 명령에 대한 파라미터는 다음과 같습니다.

- `--hostname` – 액세스할 DB 클러스터(클러스터 엔드포인트)의 호스트 이름입니다.
- `--port` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `--region` - DB 클러스터가 실행되는 AWS 리전입니다.
- `--username` – 액세스할 데이터베이스 계정입니다.

생성된 토큰에서 처음 몇 글자는 다음과 같은 모습입니다.

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

Aurora PostgreSQL 클러스터에 연결

psql을 사용한 일반적인 연결 형식은 다음과 같습니다.

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

파라미터는 다음과 같습니다.

- `host` – 액세스할 DB 클러스터(클러스터 엔드포인트)의 호스트 이름입니다.
- `port` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `sslmode` – 사용할 SSL 모드입니다.

  `sslmode=verify-full`을 사용하면 SSL 연결에서 SSL 인증서의 엔드포인트와 비교하여 DB 클러스터 엔드포인트를 확인합니다.
- `sslrootcert` - 퍼블릭 키를 포함하는 SSL 인증서 파일의 전체 경로

  자세한 내용은 SSL/TLS를 이용한 Aurora PostgreSQL 데이터 보안 단원을 참조하십시오.

  SSL 인증서를 다운로드하려면 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.
- `dbname` – 액세스할 데이터베이스입니다.
- `user` – 액세스할 데이터베이스 계정입니다.
- `password` – 서명된 IAM 인증 토큰입니다.

> **Note**
>
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

다음 예제는 psql을 사용하여 연결하는 방법을 보여줍니다. 예에서 psql은 호스트에 대해 환경 변수 RDSHOST를 사용하고 생성된 토큰에 대해 환경 변수 PGPASSWORD를 사용합니다. 또한 */sample_dir/*은 퍼블릭 키를 포함하는 SSL 인증서 파일의 전체 경로입니다.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

프록시를 통해 DB 클러스터 연결하려는 경우 [IAM 인증을 사용하여 프록시에 연결]을 참조하세요.

IAM 인증 및 AWS SDK for .NET를 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS SDK for .NET를 사용하여 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

예시

다음 코드 예제는 인증 토큰을 생성한 다음 이 토큰을 사용하여 DB 클러스터에 연결하는 방법을 보여줍니다.

이 코드 예제를 실행하려면 AWS SDK for .NET가 필요하며 이는 AWS 사이트에서 받을 수 있습니다. `AWSSDK.CORE` 및 `AWSSDK.RDS` 패키지가 필요합니다. DB 클러스터에 연결하려면 DB 엔진용 .NET 데이터베이스 커넥터(예: MariaDB 또는 MySQL용 MySQL커넥터 또는 PostgreSQL용 Npgsql)를 사용합니다.

이 코드는 Aurora MySQL DB 클러스터에 연결됩니다. 필요하다면 다음 변수 값을 변경합니다.

- `server` - 액세스할 DB 클러스터의 엔드포인트입니다.
- `user` – 액세스할 데이터베이스 계정입니다.

- database – 액세스할 데이터베이스입니다.
- port – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- SslMode – 사용할 SSL 모드입니다.

  SslMode=Required을 사용하면 SSL 연결에서 SSL 인증서의 엔드포인트와 비교하여 DB 클러스터 엔드포인트를 확인합니다.
- SslCa - Amazon Aurora에 대한 SSL 인증서의 전체 경로입니다.

  인증서를 다운로드하려면 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.

> **Note**
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();
```

```
      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

이 코드는 Aurora PostgreSQL DB 클러스터에 연결됩니다.

필요하다면 다음 변수 값을 변경합니다.

- `Server` - 액세스할 DB 클러스터의 엔드포인트입니다.
- `User ID` – 액세스할 데이터베이스 계정입니다.
- `Database` – 액세스할 데이터베이스입니다.
- `Port` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `SSL Mode` – 사용할 SSL 모드입니다.

  `SSL Mode=Required`을 사용하면 SSL 연결에서 SSL 인증서의 엔드포인트와 비교하여 DB 클러스터 엔드포인트를 확인합니다.
- `Root Certificate` - Amazon Aurora에 대한 SSL 인증서의 전체 경로입니다.

  인증서를 다운로드하려면 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.

Note

인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

프록시를 통해 DB 클러스터 연결하려는 경우 IAM 인증을 사용하여 프록시에 연결을 참조하세요.

IAM 인증 및 AWS SDK for Go를 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS SDK for Go를 사용하여 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

예시

이 코드 예제를 실행하려면 AWS SDK for Go가 필요하며 이는 AWS 사이트에서 받을 수 있습니다.

필요하다면 다음 변수 값을 변경합니다.

- `dbName` – 액세스할 데이터베이스입니다.
- `dbUser` – 액세스할 데이터베이스 계정입니다.
- `dbHost` - 액세스할 DB 클러스터의 엔드포인트입니다.

> **Note**
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

- `dbPort` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `region` - DB 클러스터가 실행되는 AWS 리전입니다.

또한 샘플 코드에서 가져온 라이브러리가 시스템에 있는지 확인합니다.

> **Important**
> 이 섹션의 예에서는 다음 코드를 사용하여 로컬 환경에서 데이터베이스에 액세스하는 자격 증명을 제공합니다.
> `creds := credentials.NewEnvCredentials()`

Amazon EC2 또는 Amazon ECS 같은 AWS 서비스에서 데이터베이스에 액세스하는 경우 코드를 다음 코드로 바꿀 수 있습니다.

```
sess := session.Must(session.NewSession())
creds := sess.Config.Credentials
```

이 변경 작업을 수행하는 경우 다음 가져오기를 추가해야 합니다.

```
"github.com/aws/aws-sdk-go/aws/session"
```

주제

- IAM 인증 및 AWS SDK for Go V2를 사용하여 연결
- IAM 인증 및 AWS SDK for Go V1을 사용하여 연결

IAM 인증 및 AWS SDK for Go V2를 사용하여 연결

IAM 인증 및 AWS SDK for Go V2를 사용하여 DB 클러스터에 연결할 수 있습니다.

다음 코드 예제는 인증 토큰을 생성한 다음 이 토큰을 사용하여 DB 클러스터에 연결하는 방법을 보여 줍니다.

이 코드는 Aurora MySQL DB 클러스터에 연결됩니다.

```
package main

import (
	"context"
	"database/sql"
	"fmt"

	"github.com/aws/aws-sdk-go-v2/config"
	"github.com/aws/aws-sdk-go-v2/feature/rds/auth"
	_ "github.com/go-sql-driver/mysql"
)

func main() {

	var dbName string = "DatabaseName"
	var dbUser string = "DatabaseUser"
	var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
	var dbPort int = 3306
```

```
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

이 코드는 Aurora PostgreSQL DB 클러스터에 연결됩니다.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)
```

```
func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

프록시를 통해 DB 클러스터 연결하려는 경우 IAM 인증을 사용하여 프록시에 연결을 참조하세요.

IAM 인증 및 AWS SDK for Go V1을 사용하여 연결

IAM 인증 및 AWS SDK for Go V1을 사용하여 DB 클러스터에 연결할 수 있습니다.

다음 코드 예제는 인증 토큰을 생성한 다음 이 토큰을 사용하여 DB 클러스터에 연결하는 방법을 보여줍니다.

이 코드는 Aurora MySQL DB 클러스터에 연결됩니다.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

이 코드는 Aurora PostgreSQL DB 클러스터에 연결됩니다.

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

프록시를 통해 DB 클러스터 연결하려는 경우 [IAM 인증을 사용하여 프록시에 연결]을 참조하세요.

IAM 인증 및 AWS SDK for Java를 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS SDK for Java를 사용하여 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성
- AWS SDK for Java 설정

Java 2.x용 SDK를 사용하는 방법에 대한 예제는 Java 2.x용 SDK를 사용한 Amazon RDS 예제를 참조하세요. 또한 AWS 고급 JDBC 래퍼를 사용할 수 있습니다. AWS Advanced JDBC Wrapper documentation을 참조하세요.

주제

- IAM 인증 토크 생성
- IAM 인증 토큰 수동 구성
- DB 클러스터에 연결

IAM 인증 토크 생성

AWS SDK for Java로 프로그램을 개발할 때는 `RdsIamAuthTokenGenerator` 클래스를 사용하여 서명된 인증 토큰을 가져올 수 있습니다. 이 클래스를 사용하려면 AWS 자격 증명을 입력해야 합니다. 이렇게 하려면 `DefaultAWSCredentialsProviderChain` 클래스의 인스턴스를 생성합니다. `DefaultAWSCredentialsProviderChain`은 기본 자격 증명 공급자 체인에서 찾은 첫 번째 AWS 액세스 키와 보안 키를 사용합니다. AWS 액세스 키에 대한 자세한 내용은 사용자의 액세스 키 관리를 참조하세요.

> **Note**
>
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

RdsIamAuthTokenGenerator 인스턴스를 생성한 후에는 getAuthToken 메서드를 호출하여 서명된 토큰을 가져올 수 있습니다. 이때 AWS 리전, 호스트 이름, 포트 이름 및 사용자 이름을 입력합니다. 다음은 각 정보의 입력 방법을 설명한 코드 예제입니다.

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

IAM 인증 토큰 수동 구성

Java에서 인증 토큰을 가장 쉽게 생성할 수 있는 방법은 RdsIamAuthTokenGenerator를 사용하는 것입니다. 이 클래스는 인증 토큰을 생성한 후 AWS 서명 버전 4를 사용해 서명까지 마칩니다. (자세한 내용은 AWS 일반 참조의 서명 버전 4 서명 프로세스를 참조하세요.)

그 밖에 다음 코드 예제와 같이 인증 토큰을 수동으로 구성하여 서명하는 방법도 있습니다.

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
```

```
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
     String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
     String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4:  Add the signing info to the request");

        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
        canonicalQueryParameters.put("Action", action);
```

```
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
 "%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
 canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
 currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
```

```
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
 serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                       SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
```

```
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

DB 클러스터에 연결

다음은 인증 토큰을 생성하고, 이를 사용하여 Aurora MySQL을 실행하는 클러스터에 연결하는 방법을 보여주는 코드 예입니다.

이 코드 예제를 실행하려면 AWS SDK for Java가 필요하며 이는 AWS 사이트에서 받을 수 있습니다. 또한 다음이 필요합니다.

- MySQL Connector/J. 이 코드 예제는 `mysql-connector-java-5.1.33-bin.jar`로 테스트됩니다.
- AWS 리전에 고유한 Amazon Aurora에 대한 중간 인증서입니다. (자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.) 예제가 실행되면 클래스 로더가 쉽게 찾을 수 있도록 이 Java 코드 예제와 동일한 디렉터리에서 인증서를 찾기 시작합니다.
- 필요하다면 다음 변수 값을 변경합니다.
  - `RDS_INSTANCE_HOSTNAME` – 액세스할 DB 클러스터의 호스트 이름입니다.
  - `RDS_INSTANCE_PORT` - PostgreSQL DB 클러스터에 연결할 때 사용할 포트 이름입니다.
  - `REGION_NAME` - DB 클러스터가 실행되는 AWS 리전입니다.
  - `DB_USER` – 액세스할 데이터베이스 계정입니다.
  - `SSL_CERTIFICATE` - AWS 리전에 고유한 Amazon Aurora에 대한 SSL 인증서입니다.

    AWS 리전에 대한 인증서를 다운로드하려면 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요. SSL 인증서는 예제 실행 시 클래스 로더가 인증서를 찾을 수 있도록 이 Java 프로그램 파일과 동일한 디렉터리에 설치합니다.

다음은 기본 자격 증명 공급자 체인에서 AWS 자격 증명을 가져오는 코드 예제입니다.

> **Note**
>
> 보안 모범 사례로 여기에 표시된 프롬프트 이외에 DEFAULT_KEY_STORE_PASSWORD에 대한 암호를 지정하는 것이 좋습니다.

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();
```

```
    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
```

```
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
 AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
 AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

 .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
```

```
     * This method sets the SSL properties which specify the key store file, its type 
and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword", 
DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL 
verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
```

```
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
 Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

프록시를 통해 DB 클러스터 연결하려는 경우 [IAM 인증을 사용하여 프록시에 연결](#)을 참조하세요.

IAM 인증 및 AWS SDK for Python (Boto3)를 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS SDK for Python (Boto3)를 사용하여 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

사전 조건

다음은 IAM 인증을 사용하여 DB 클러스터에 연결하기 위한 사전 조건입니다.

- IAM 데이터베이스 인증의 활성화 및 비활성화
- IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용
- IAM 인증을 사용하여 데이터베이스 계정 생성

또한 샘플 코드에서 가져온 라이브러리가 시스템에 있는지 확인합니다.

예시

코드 예제에서는 공유 자격 증명에 프로필을 사용합니다. 자격 증명 지정에 대한 자세한 내용은 AWS SDK for Python (Boto3) 설명서의 자격 증명을 참조하십시오.

다음 코드 예제는 인증 토큰을 생성한 다음 이 토큰을 사용하여 DB 클러스터에 연결하는 방법을 보여 줍니다.

이 코드 예제를 실행하려면 AWS SDK for Python (Boto3)가 필요하며 이는 AWS 사이트에서 받을 수 있습니다.

필요하다면 다음 변수 값을 변경합니다.

- `ENDPOINT` - 액세스할 DB 클러스터의 엔드포인트입니다.
- `PORT` – DB 클러스터에 연결할 때 사용할 포트 번호입니다.
- `USER` – 액세스할 데이터베이스 계정입니다.
- `REGION` - DB 클러스터가 실행되는 AWS 리전입니다.
- `DBNAME` – 액세스할 데이터베이스입니다.
- `SSLCERTIFICATE` - Amazon Aurora에 대한 SSL 인증서의 전체 경로입니다.

  `ssl_ca`의 경우 SSL 인증서를 지정합니다. SSL 인증서를 다운로드하려면 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 섹션을 참조하세요.

> **Note**
>
> 인증 토큰을 생성할 때는 DB 클러스터 엔드포인트 대신 사용자 지정 Route 53 DNS 레코드 또는 Aurora 사용자 지정 엔드포인트를 사용할 수 없습니다.

이 코드는 Aurora MySQL DB 클러스터에 연결됩니다.

이 코드를 실행하기 전에 Python Package 색인에 있는 지침에 따라 PyMySQL 드라이버를 설치하세요.

```
import pymysql
import sys
import boto3
import os
```

```
ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

이 코드는 Aurora PostgreSQL DB 클러스터에 연결됩니다.

이 코드를 실행하기 전에 [Psycopg 설명서]의 지침에 따라 psycopg2를 설치하세요.

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
```

```
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

프록시를 통해 DB 클러스터 연결하려는 경우 IAM 인증을 사용하여 프록시에 연결을 참조하세요.

IAM DB 인증 문제 해결

여기에서는 몇 가지 일반적인 IAM DB 인증 문제에 대한 문제 해결 아이디어와 IAM DB 인증에 대한 CloudWatch 로그 관련 정보를 확인할 수 있습니다.

IAM DB 인증 오류 로그를 CloudWatch Logs로 내보내기

IAM DB 인증 오류 로그는 데이터베이스 호스트에 저장되며 이러한 로그를 CloudWatch Logs 계정으로 내보낼 수 있습니다. 이 페이지의 로그 및 문제 해결 방법을 사용하여 IAM DB 인증 문제를 해결합니다.

콘솔, AWS CLI 및 RDS API에서 CloudWatch Logs로 로그 내보내기를 사용 설정할 수 있습니다. 콘솔 지침은 Amazon CloudWatch Logs에 데이터베이스 로그 게시(을)를 참조하세요.

AWS CLI에서 DB 클러스터를 만들 때 IAM DB 인증 오류 로그를 CloudWatch Logs로 내보내려면 다음 명령을 사용합니다.

```
aws rds create-db-cluster --db-cluster-identifier mydbinstance \
--region us-east-1 \
--engine postgres \
--engine-version 16 \
--master-username master \
--master-user-password password \
```

```
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

AWS CLI에서 DB 클러스터를 수정할 때 IAM DB 인증 오류 로그를 CloudWatch Logs로 내보내려면 다음 명령을 사용합니다.

```
aws rds modify-db-cluster --db-instance-identifier mydbcluster \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

DB 클러스터가 IAM DB 인증 로그를 CloudWatch Logs로 내보내고 있는지 확인하려면 `describe-db-instances` 명령의 출력에서 `EnabledCloudwatchLogsExports` 파라미터가 `iam-db-auth-error`로 설정되어 있는지 확인합니다.

```
aws rds describe-db-cluster --region us-east-1 --db-cluster-identifier mydbcluster
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
            ],
            ...
```

IAM DB 인증 CloudWatch 지표

Amazon Aurora는 IAM DB 인증에 대한 실시간에 가까운 지표를 Amazon CloudWatch 계정에 제공합니다. 다음 표에는 CloudWatch를 사용하여 사용할 수 있는 IAM DB 인증 지표가 나열되어 있습니다.

지표	설명
`IamDbAuthConnectionRequests`	IAM DB 인증으로 구성된 총 연결 요청 개수입니다.
`IamDbAuthConnectionSuccess`	성공한 IAM DB 인증 요청의 총 개수입니다.
`IamDbAuthConnectionFailure`	실패한 IAM DB 인증 요청의 총 개수입니다.
`IamDbAuthConnectionFailureInvalidToken`	유효하지 않은 토큰으로 인해 실패한 IAM DB 인증 요청의 총 개수입니다.

지표	설명
IamDbAuthConnectionFailureInsufficientPermissions	잘못된 정책 또는 권한으로 인해 실패한 IAM DB 인증 요청의 총 개수입니다.
IamDbAuthConnectionFailureThrottling	IAM DB 인증 스로틀링으로 인해 실패한 IAM DB 인증 요청의 총 개수입니다.
IamDbAuthConnectionFailureServerError	IAM DB 인증 기능의 내부 서버 오류로 인해 실패한 IAM DB 인증 요청의 총 개수입니다.

일반적인 문제 및 해결 방법

IAM DB 인증 사용 시 다음과 같은 문제가 발생할 수 있습니다. 표에 나와 있는 문제 해결 단계를 사용하여 문제를 해결합니다.

오류	지표	원인	솔루션

`[ERROR] Failed to authenticate the connection request for user` *`db_user`* `because the provided token is malformed or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureInvalidToken`
		연결 요청의 IAM DB 인증 토큰이 유효한 SigV4a 토큰이 아니거나 올바른 형식이 아닙니다.	애플리케이션에서 토큰 생성 전략을 확인합니다. 경우에 따라 토큰을 유효한 형식으로 전달해야 합니다. 토큰을 잘라내거나 잘못된 문자열 형식으로 지정하면 토큰이 무효화됩니다.

`[ERROR] Failed to authenticate the connection request for user` *`db_user`* `because`	`IamDbAuthConnectionFailure`	IAM DB 인증 토큰이 만료되었습니다. 토큰은 15분 동안만 유효합니다.	애플리케이션에서 토큰 캐싱 및/또는 토큰 재사용 로직을 확인합니다. 15분보다 오래된

오류	지표	원인	솔루션
the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)	IamDbAuthConnectionFailureInvalidToken		토큰은 재사용해서는 안 됩니다.
[ERROR] Failed to authorize the connection request for user *db_user* because the IAM policy assumed by the caller 'arn:aws:sts::123456789012:assumed-role/ <RoleName>/ <RoleSession>' is not authorized to perform \`rds-db:connect\` on the DB instance. (Status Code: 403, Error Code:NotAuthorized)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInsufficientPermissions	이 오류는 다음과 같은 이유 때문일 수 있습니다. • 애플리케이션에서 수임하는 IAM 정책은 `rds-db:connect` 작업을 승인하지 않습니다. • *db_user*가 데이터베이스에 연결하는 데 잘못된 역할/정책을 수임하고 있습니다. • *db_user*에 대해 올바른 정책을 수임하고 있지만 올바른 데이터베이스에 연결되어 있지 않습니다.	애플리케이션에서 수임하는 IAM 역할 및/또는 정책을 확인합니다. DB에 연결할 때와 동일한 정책을 수임하여 토큰을 생성해야 합니다.

오류	지표	원인	솔루션

`[ERROR] Failed to authorize the connection request for user db_user due to IAM DB authentication throttling. (Status Code: 429, Error Code: ThrottlingException)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureThrottling`
		짧은 시간 내에 DB에 너무 많은 연결 요청을 하고 있습니다. IAM DB 인증 스로틀링 한도는 초당 200개의 연결입니다.	IAM 인증으로 새 연결을 설정하는 속도를 줄입니다. 애플리케이션에서 설정된 연결을 재사용하려면 RDS 프록시를 사용하여 연결 풀링을 구현하는 것이 좋습니다.

`[ERROR] Failed to authorize the connection request for user db_user due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureThrottling`
		IAM DB 인증으로 DB 연결을 승인하는 동안 내부 오류가 발생했습니다.	문제를 조사하려면 https://aws.amazon.com/premiumsupport/를 통해 문의하시기 바랍니다.

Amazon Aurora 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Aurora 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- Aurora에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 AWS 계정 외부의 사용자가 내 Aurora 리소스에 액세스할 수 있도록 하려고 합니다.

Aurora에서 작업을 수행할 권한이 없음

AWS Management Console에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예제 오류는 mateojackson 사용자가 콘솔을 사용하여 *##*에 대한 세부 정보를 보려고 하지만 rds:*GetWidget* 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

이 경우 Mateo는 *my-example-widget* 태스크를 사용하여 rds:*GetWidget* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 태스크를 수행할 권한이 없다는 오류가 수신되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다. 역할을 Aurora로 전달하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 이름이 marymajor인 사용자가 콘솔을 사용하여 Aurora에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 태스크를 수행하려면 서비스에 서비스 역할이 부여한 권한이 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary는 iam:PassRole 태스크를 수행하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

내 AWS 계정 외부의 사용자가 내 Aurora 리소스에 액세스할 수 있도록 하려고 합니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제

어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Aurora에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Aurora에서 IAM을 사용하는 방법 단원을 참조하십시오.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공을 참조하십시오.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 서드 파티가 소유한 AWS 계정에 대한 액세스 제공을참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하십시오.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.

Amazon Aurora의 로깅 및 모니터링

모니터링은 Amazon Aurora와 AWS 솔루션의 신뢰성, 가용성 및 성능을 유지하는 중요한 역할을 합니다. 다중 지점 실패가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집해야 합니다. AWS는 Amazon Aurora 리소스를 모니터링하고 잠재적 인시던트에 대응하기 위한 여러 도구를 제공합니다.

Amazon CloudWatch 경보

Amazon CloudWatch 경보를 사용하면 지정한 기간 동안 단일 지표를 감시합니다. 지표가 지정된 임계값을 초과하면 Amazon SNS 주제 또는 AWS Auto Scaling 정책으로 알림이 전송됩니다. CloudWatch 경보는 단순히 특정 상태에 있다고 해서 작업을 호출하지 않습니다. 대신, 상태가 변경되어 지정된 기간 동안 유지되어야 합니다.

AWS CloudTrail 로그

CloudTrail은 Amazon Aurora에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공합니다. CloudTrail은 콘솔의 호출과 Amazon RDS API 작업에 대한 코드 호출을 포함하여 Amazon Aurora에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. CloudTrail에서 수집하는 정보를 사용하여 Amazon Aurora에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다. 자세한 정보는 AWS CloudTrail에서 Amazon Aurora API 호출 모니터링을 참조하세요.

확장 모니터링

Amazon Aurora는 DB 클러스터가 실행되는 운영 체제(OS)에 대한 측정치를 실시간으로 제공합니다. 콘솔을 사용하여 DB 클러스터에 대한 측정치를 보거나, 선택한 모니터링 시스템의 Amazon CloudWatch Logs에서 확장 모니터링 JSON 출력을 사용할 수 있습니다. 자세한 정보는 Enhanced Monitoring을 사용하여 OS 지표 모니터링을 참조하세요.

Amazon RDS 성능 개선 도우미

성능 개선 도우미는 기존 Amazon Aurora 모니터링 기능을 확장한 것으로서 데이터베이스 성능을 표시하여 영향을 미치는 성능 문제를 분석하는 데 효과적입니다. 성능 개선 도우미 대시보드가 데이터베이스 부하를 시각화하여 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 부하를 필터링합니다. 자세한 정보는 성능 개선 도우미를 통한 Amazon Aurora 모니터링을 참조하세요.

데이터베이스 로그

AWS Management Console 콘솔, AWS CLI 또는 RDS API를 사용하여 데이터베이스 로그를 보고 다운로드하고 조사할 수 있습니다. 자세한 정보는 Amazon Aurora 로그 파일 모니터링을 참조하세요.

Amazon Aurora 권장 사항

Amazon Aurora에서 데이터베이스 리소스에 대한 자동 권장 사항을 제공합니다. 이러한 권장 사항은 DB 클러스터 구성, 사용량 및 성능 데이터 분석을 통해 모범 사례 지침을 제공합니다. 자세한 정보는 Amazon Aurora의 권장 사항을 참조하세요.

Amazon Aurora 이벤트 알림

Amazon Aurora는 Amazon Aurora 이벤트 발생 시 Amazon Simple Notification Service(Amazon SNS)를 사용하여 알림 서비스를 제공합니다. 이 서비스는 AWS 리전에 따라 Amazon SNS가 지원하는 알림 메시지 형식에 따라 이메일, 문자 또는 HTTP 엔드포인트 호출 등이 될 수 있습니다. 자세한 정보는 Amazon RDS 이벤트 알림 작업을 참조하세요.

AWS Trusted Advisor

Trusted Advisor는 수십만 명의 AWS 고객에게 서비스를 제공하면서 익힌 모범 사례를 활용합니다. Trusted Advisor는 AWS 환경을 검사한 후 비용 절감, 시스템 가용성 및 성능 향상 또는 보안 격차를 해결할 기회가 있을 때 권장 사항을 제시합니다. 모든 AWS 고객은 5개의 Trusted Advisor 점검 항목에 액세스할 수 있습니다. Business 또는 Enterprise Support 플랜을 보유한 고객은 모든 Trusted Advisor 점검 항목을 볼 수 있습니다.

Trusted Advisor에는 다음과 같이 Amazon Aurora 관련 검사가 있습니다.

- Amazon Aurora 유휴 DB 인스턴스

- Amazon Aurora 보안 그룹 액세스 위험
- Amazon Aurora 백업
- Amazon Aurora 다중 AZ
- Aurora DB 인스턴스 액세스

이러한 사항에 대한 자세한 정보를 알고 싶다면 Trusted Advisor Best Practices (Checks) 단원을 참조하십시오.

데이터베이스 활동 스트림

데이터베이스 활동 스트림이 데이터베이스 활동 스트림에 대한 DBA 액세스를 제어하여 내부 위협으로부터 데이터베이스를 보호할 수 있습니다. 따라서 데이터베이스 활동 스트림의 수집, 전송, 저장 및 후속 처리는 데이터베이스를 관리하는 DBA의 액세스 범위를 벗어납니다. 데이터베이스 활동 스트림은 데이터베이스를 보호하고 규정 준수 및 규제 요건을 충족할 수 있습니다. 자세한 내용은 데이터베이스 활동 스트림을 사용하여 Amazon Aurora 모니터링 섹션을 참조하세요.

Aurora 모니터링에 대한 자세한 내용은 Amazon Aurora 클러스터에서 지표 모니터링 섹션을 참조하세요.

Amazon Aurora의 규정 준수 확인

서드 파티 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 Amazon Aurora의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램 범위에 속하는 AWS 서비스의 목록은 규정 준수 프로그램 제공 AWS 범위 내 서비스를 참조하세요. 일반 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact를 사용하여 제3자 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact의 보고서 다운로드를 참조하십시오.

Amazon Aurora 사용 시 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- Architecting for HIPAA Security and Compliance on Amazon Web Services(Amazon Web Services에서 HIPAA 보안 및 규정 준수를 위한 설계) - 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 생성하는 방법을 설명합니다.
- AWS 규정 준수 리소스 이 워크북 및 안내서는 귀사의 산업 및 위치에 적용될 수 있습니다.
- AWS Config – 이 AWS 서비스로 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.
- AWS Security Hub – 이 AWS 서비스(은)는 AWS 내의 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 제어를 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.

Amazon Aurora의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하십시오.

AWS 글로벌 인프라 외에 Aurora도 데이터 복원성과 백업 요구 사항을 지원하는 여러 가지 기능을 제공합니다.

백업 및 복원

Aurora는 클러스터 볼륨을 자동으로 백업한 후 백업 보존 기간 동안 복원 데이터를 보관합니다. Aurora 백업은 연속적으로 또는 증분식으로 이루어지기 때문에 백업 보관 기간 내에 어떤 시점으로든 신속하게 복구가 가능합니다. 백업 데이터를 쓰는 중에도 성능에 미치는 영향이나 데이터베이스 서비스 중단은 일어나지 않습니다. 백업 보존 기간은 DB 클러스터를 생성 또는 설정 변경할 때 1일에서 35일까지 지정할 수 있습니다.

백업 보존 기간을 넘겨서 백업을 보존하고 싶을 때는 클러스터 볼륨의 데이터 스냅샷을 캡처하는 것도 한 방법입니다. Aurora는 증분 복원 데이터를 전체 백업 기간 동안 보유합니다. 따라서 백업 보존 기간을 넘어서 보관할 데이터에 대한 스냅샷만 생성해야 합니다. 새로운 DB 클러스터를 스냅샷에서 생성할 수 있기 때문입니다.

Aurora에서 유지되는 백업 데이터에서 또는 이전에 저장한 DB 클러스터 스냅샷에서 새 Aurora DB 클러스터를 생성하여 데이터를 복구할 수 있습니다. 백업 보존 기간 중 언제든지 백업 데이터에서 새로운 DB 클러스터 복사본을 빠르게 생성할 수 있습니다. 백업 보존 기간 중 Aurora 백업의 연속 및 증분 특성은 복구 횟수를 늘리기 위해 데이터 스냅샷을 자주 캡처할 필요가 없다는 것을 의미합니다.

자세한 내용은 Amazon Aurora DB 클러스터 백업 및 복구 섹션을 참조하세요.

복제

Aurora 복제본은 Aurora DB 클러스터의 독립 엔드포인트로서, 읽기 작업을 조정하고 가용성을 높이기 위해 사용하기에 가장 적합합니다. 최대 15개의 Aurora 복제본을 AWS 리전 내 DB 클러스터에 포함된 가용 영역에 배포할 수 있습니다. DB 클러스터 볼륨은 DB 클러스터의 데이터 사본들로 구성됩니다. 하지만 DB 클러스터의 기본 DB 인스턴스 및 Aurora 복제본에는 클러스터 볼륨의 데이터가 단 하나의 논

리 볼륨으로 표시됩니다. 기본 DB 인스턴스에 장애가 발생하면 Aurora 복제본이 기본 DB 인스턴스로 승격됩니다.

Aurora에서도 Aurora MySQL과 Aurora PostgreSQL의 고유 복제 옵션이 지원됩니다.

자세한 내용은 Amazon Aurora를 사용한 복제 섹션을 참조하세요.

Failover

Aurora는 단일 AWS 리전에서 다중 가용 영역에 걸쳐 DB 클러스터에 데이터 복사본을 저장합니다. 이러한 복사본 저장은 DB 클러스터 인스턴스의 다중 가용 영역 포괄 여부에 상관없이 발생합니다. 가용 영역에서 Aurora 복제본을 생성할 때 Aurora는 동기식으로 복제본을 자동 프로비저닝 및 유지합니다. 기본 DB 인스턴스는 가용 영역에서 Aurora 복제본으로 동기식으로 복제되어 데이터 이중화를 제공하고 I/O 중지를 제거하며 시스템 백업 시 지연 시간 급증을 최소화합니다. DB 클러스터를 고가용성으로 실행하면 계획된 시스템 유지 관리 중 가용성을 향상시킬 수 있으며, 데이터베이스에서 오류 및 가용 영역 중단이 일어나는 것을 방지할 수 있습니다.

자세한 내용은 Amazon Aurora의 고가용성 단원을 참조하십시오.

Amazon Aurora의 인프라 보안

관리형 서비스인 Amazon Relational Database Service는 AWS 전역 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon RDS에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

또한 Aurora는 인프라 보안을 지원하는 기능을 제공합니다.

보안 그룹

보안 그룹은 DB 클러스터에서 송수신되는 트래픽에 대한 액세스를 제어합니다. 기본적으로 DB 클러스터에 대한 네트워크 액세스는 해제되어 있습니다. IP 주소 범위, 포트 또는 보안 그룹에서 액세스를 허용하는 보안 그룹의 규칙을 지정할 수 있습니다. 수신 규칙이 설정되면 동일한 규칙이 해당 보안 그룹과 연결된 모든 DB 클러스터에 적용됩니다.

자세한 내용은 보안 그룹을 통한 액세스 제어 단원을 참조하세요.

퍼블릭 액세스 가능성

Amazon VPC 서비스 기반 Virtual Private Cloud(VPC)에서 DB 인스턴스를 시작할 때 해당 DB 인스턴스의 퍼블릭 액세스를 켜거나 끌 수 있습니다. 퍼블릭 액세스 가능성 파라미터를 사용하여 사용자가 생성한 DB 인스턴스가 퍼블릭 IP 주소로 확인되는 DNS 이름을 가지도록 지정할 수 있습니다. 이 파라미터를 사용하여 DB 인스턴스에 대한 퍼블릭 액세스 여부를 지정할 수 있습니다. Public accessibility 파라미터를 수정하여 퍼블릭 액세스 가능성을 켜거나 끄도록 DB 인스턴스를 수정할 수 있습니다.

자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 섹션을 참조하세요.

> **Note**
>
> DB 인스턴스가 VPC에 있지만 공개적으로 액세스할 수 없는 경우 AWS Site-to-Site VPN 연결 또는 AWS Direct Connect 연결을 사용하여 프라이빗 네트워크에서 액세스할 수도 있습니다. 자세한 내용은 인터네트워크 트래픽 개인 정보 단원을 참조하세요.

Amazon RDS API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스VPC 엔드포인트를 생성하여 VPC와 Amazon RDS API 엔드포인트 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 로 구동됩니다AWS PrivateLink

AWS PrivateLink를 사용하면 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 Amazon RDS API 작업에 비공개로 액세스할 수 있습니다. VPC의 DB 인스턴스는 DB 인스턴스 및 DB 클러스터를 시작, 수정 또는 종료하기 위해 Amazon RDS API 엔드포인트와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다. 또한 DB 인스턴스가 사용 가능한 RDS API 작업을 사용하기 위해 퍼블릭 IP 주소가 필요하지 않습니다. VPC와 Amazon RDS 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다. 탄력적 네트워크 인터페이스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 탄력적 네트워크 인터페이스를 참조하세요.

VPC 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트 (AWS PrivateLink)를 참조하세요. RDS API 작업에 대한 자세한 내용은 Amazon RDS API 참조를 참조하세요.

DB 클러스터에 연결하기 위해 인터페이스 VPC 엔드포인트가 필요하지 않습니다. 자세한 내용은 VPC에서 DB 클러스터에 액세스하는 시나리오을 참조하세요.

VPC 엔드포인트에 대한 고려 사항

Amazon RDS API 엔드포인트에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서에서 인터페이스 엔드포인트 속성 및 제한 사항을 검토해야 합니다.

Amazon Aurora 리소스 관리와 관련된 모든 RDS API 작업은 AWS PrivateLink를 사용하여 VPC에서 사용할 수 있습니다.

VPC 엔드포인트 정책은 RDS API 엔드포인트에 대해 지원됩니다. 기본적으로, 엔드포인트를 통해 RDS API 작업에 대한 전체 액세스가 허용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

가용성

현재 Amazon RDS API가 VPC 엔드포인트를 지원하는 AWS 리전은 다음과 같습니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오레곤)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 캐나다 서부(캘거리)
- 중국(베이징)
- 중국(닝샤)
- 유럽(프랑크푸르트)
- 유럽(취리히)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스톡홀름)
- 유럽(밀라노)
- 이스라엘(텔아비브)
- 중동(바레인)
- 남아메리카(상파울루)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

Amazon RDS API에 대한 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon RDS API에 대한 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하십시오.

서비스 이름 `com.amazonaws.region.rds`을 사용하여 Amazon RDS API에 대한 VPC 엔드포인트를 생성합니다.

중국의 AWS 리전을 제외하고, 엔드포인트에 프라이빗 DNS를 활성화한 경우 AWS 리전에 대한 기본 DNS 이름(예:`rds.us-east-1.amazonaws.com`)을 사용하여 VPC 엔드포인트를 통해 Amazon RDS에 API 요청을 수행할 수 있습니다. 중국(베이징) 및 중국(닝샤) AWS 리전의 경우 각각 `rds-api.cn-north-1.amazonaws.com.cn` 및 `rds-api.cn-northwest-1.amazonaws.com.cn`을 사용하여 VPC 엔드포인트를 통해 API 요청을 수행할 수 있습니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하십시오.

Amazon RDS API에 대한 VPC 엔드포인트 정책 생성

Amazon RDS API에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하십시오.

예제: Amazon RDS API 작업에 대한 VPC 엔드포인트 정책

다음은 Amazon RDS API에 대한 엔드포인트 정책의 예입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 Amazon RDS API 작업에 부여합니다.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
```

```
        "Action":[
           "rds:CreateDBInstance",
           "rds:ModifyDBInstance",
           "rds:CreateDBSnapshot"
        ],
        "Resource":"*"
     }
  ]
}
```

예제: 지정된 AWS 계정의 모든 액세스를 거부하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 AWS 계정 123456789012가 엔드포인트를 사용하는 리소스에 대한 모든 액세스를 거부합니다. 이 정책은 다른 계정의 모든 작업을 허용합니다.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
  ]
}
```

Amazon Aurora의 보안 모범 사례

AWS Identity and Access Management(IAM) 계정을 사용해 Amazon RDS API 작업, 특히 Amazon Aurora 리소스를 만들거나, 수정하거나, 삭제하는 작업에 대한 액세스를 제어합니다. 이러한 리소스 중에는 DB 클러스터, 보안 그룹 및 파라미터 그룹이 있습니다. 또한 IAM 을 사용해 DB 클러스터 백업 및 복구 같은 공통 관리 작업을 수행하는 작업을 제어합니다.

- 본인을 포함하여 Amazon Aurora 리소스를 관리하는 각 개인에 대해 개별 사용자를 만듭니다. AWS 루트 자격 증명을 사용하여 Amazon Aurora 리소스를 관리하지 마세요.

- 각 사용자에게 각자의 임무를 수행하는 데 필요한 최소 권한 집합을 부여합니다.
- IAM 그룹을 사용해 여러 사용자에 대한 권한을 효과적으로 관리합니다.
- IAM 자격 증명을 정기적으로 순환합니다.
- Amazon Aurora 보안 암호를 자동으로 교체할 수 있도록 AWS Secrets Manager를 구성합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서에서 AWS Secrets Manager 비밀 교체를 참조하세요. AWS Secrets Manager 프로그래밍 방식에서 자격 증명을 검색할 수도 있습니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 비밀 값 검색을 참조하세요.

Amazon Aurora 보안에 대한 자세한 내용은 Amazon Aurora의 보안 섹션을 참조하세요. IAM에 대한 자세한 내용은 AWS Identity and Access Management 단원을 참조하십시오. IAM 모범 사례에 대한 자세한 내용은 IAM 모범 사례 단원을 참조하십시오.

AWS Security Hub는 보안 제어를 사용하여 리소스 구성 및 보안 표준을 평가하여 다양한 규정 준수 프레임워크를 준수할 수 있도록 지원합니다. Security Hub를 사용하여 RDS 리소스를 평가하는 방법에 대한 자세한 내용은 AWS Security Hub 사용 설명서의 Amazon Relational Database Service 제어를 참조하세요.

Security Hub를 사용하여 보안 모범 사례와 관련된 RDS의 사용량을 모니터링할 수 있습니다. 자세한 내용은 AWS Security Hub란 무엇인가요?를 참조하세요.

AWS Management Console, AWS CLI 또는 RDS API를 사용하여 마스터 사용자의 암호를 변경합니다. SQL 클라이언트 등과 같은 다른 도구를 사용하여 마스터 사용자 암호를 변경할 경우 의도치 않게 사용자에 대해 권한이 취소될 수 있습니다.

Amazon GuardDuty는 Amazon RDS 로그인 활동을 비롯한 다양한 데이터 소스를 분석하고 처리하는 지속적 보안 모니터링 서비스입니다. 위협 인텔리전스 피드와 기계 학습을 사용하여 AWS 환경 내에서 예기치 않게 발생하는 무단의 잠재적인 의심스러운 로그인 동작과 악의적 활동을 찾아냅니다.

Amazon GuardDuty RDS 보호에서 데이터베이스에 대한 위협을 나타내는 잠재적으로 의심스럽거나 변칙적인 로그인 시도가 탐지되면 GuardDuty는 손상되었을지도 모르는 데이터베이스 관련 세부 정보가 포함된 새 조사 결과를 생성합니다. 자세한 정보는 Amazon Aurora용 Amazon GuardDuty RDS Protection을 이용한 위협 모니터링을 참조하세요.

보안 그룹을 통한 액세스 제어

VPC 보안 그룹은 DB 클러스터에서 송수신되는 트래픽에 대한 액세스를 제어합니다. 기본적으로 DB 클러스터에 대한 네트워크 액세스는 해제되어 있습니다. IP 주소 범위, 포트 또는 보안 그룹에서 액세

스를 허용하는 보안 그룹의 규칙을 지정할 수 있습니다. 수신 규칙이 설정되면 동일한 규칙이 해당 보안 그룹과 연결된 모든 DB 클러스터에 적용됩니다. 보안 그룹에서 최대 20개의 규칙을 지정할 수 있습니다.

VPC 보안 그룹 개요

각 VPC 보안 그룹 규칙을 설정하면 특정 소스가 해당 VPC 보안 그룹과 연결되어 있는 VPC의 DB 클러스터에 액세스할 수 있습니다. 소스는 주소 범위(예: 203.0.113.0/24) 또는 다른 VPC 보안 그룹일 수 있습니다. VPC 보안 그룹을 소스로 지정하면 소스 VPC 보안 그룹을 사용하는 모든 인스턴스(일반적으로 애플리케이션 서버)에서 수신 트래픽이 허용됩니다. VPC 보안 그룹에는 인바운드 및 아웃바운드 트래픽을 모두 제어하는 규칙이 있을 수 있습니다. 그러나 아웃바운드 트래픽 규칙은 일반적으로 DB 클러스터에 적용되지 않습니다. 아웃바운드 트래픽 규칙은 DB 클러스터가 클라이언트 역할을 하는 경우에만 적용됩니다. Amazon EC2 API를 사용하거나, 혹은 VPC 콘솔에서 보안 그룹 옵션을 선택하여 VPC 보안 그룹을 생성해야 합니다.

VPC의 클러스터에 대한 액세스를 허용하는 VPC 보안 그룹과 관련된 규칙을 만들 때 그 규칙이 액세스를 허용하는 주소들의 각 범위에 대해 포트를 지정해야 합니다. 예를 들어, VPC의 인스턴스에 대한 Secure Shell(SSH) 액세스를 활성화하고 싶다면 지정된 주소 범위와 관련해 TCP 포트 22에 대한 액세스를 허용하는 규칙을 생성합니다.

VPC의 서로 다른 인스턴스에게 서로 다른 포트에 대한 액세스를 허용하는 여러 개의 VPC 보안 그룹을 구성할 수 있습니다. 예를 들어 VPC의 웹 서버에 대해서는 TCP 포트 80으로 액세스가 가능하도록 VPC 보안 그룹을 생성합니다. 그런 다음 VPC의 Aurora MySQL DB 인스턴스에 대해서는 TCP 포트 3306으로 액세스할 수 있도록 다른 VPC 보안 그룹을 만들면 됩니다.

> **Note**
>
> Aurora DB 클러스터에서 DB 클러스터와 연결된 VPC 보안 그룹은 DB 클러스터의 모든 DB 인스턴스와도 연결되어 있습니다. DB 클러스터 또는 DB 인스턴스의 VPC 보안 그룹을 변경하면 이 변경 사항은 DB 클러스터의 모든 DB 인스턴스에 자동으로 적용됩니다.

VPC 보안 그룹에 관한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹을 참조하십시오.

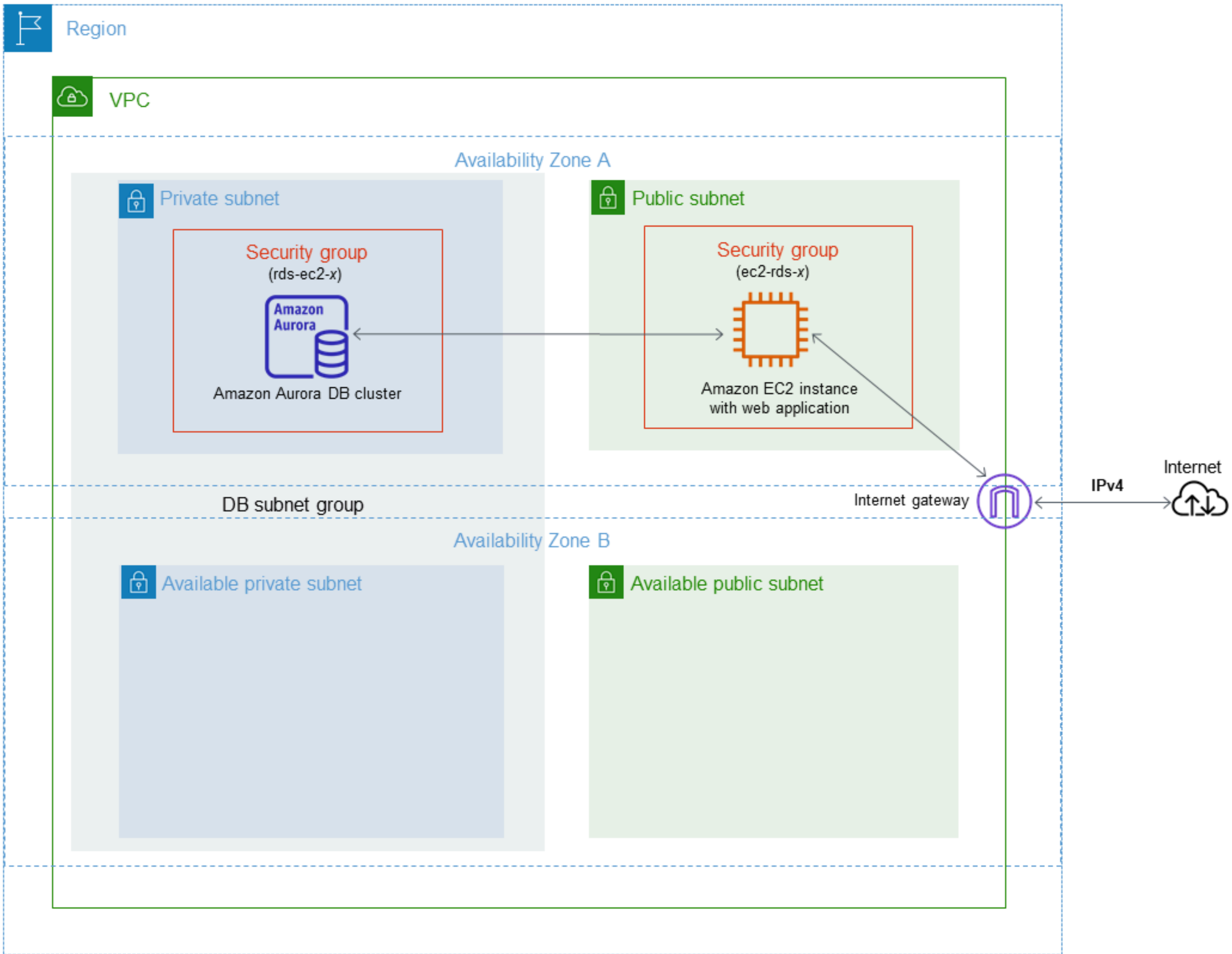
> **Note**
>
> DB 클러스터가 VPC에 있지만 공개적으로 액세스할 수 없는 경우 AWS Site-to-Site VPN 연결 또는 AWS Direct Connect 연결을 사용하여 프라이빗 네트워크에서 액세스할 수도 있습니다. 자세한 정보는 [인터네트워크 트래픽 개인 정보]을 참조하세요.

보안 그룹 시나리오

VPC에서 DB 클러스터를 사용하는 일반적인 사례는 동일한 VPC의 Amazon EC2 인스턴스에서 실행 중이며 VPC 외부의 클라이언트 애플리케이션에서 액세스한 애플리케이션 서버를 사용하여 데이터를 공유하는 것입니다. 이러한 시나리오에서는 AWS Management Console의 RDS 및 VPC 페이지를 사용하거나, 혹은 RDS 및 EC2 API 작업을 사용하여 필요한 인스턴스와 보안 그룹을 생성합니다.

1. VPC 보안 그룹(예: `sg-0123ec2example`)을 생성하고 클라이언트 애플리케이션의 IP 주소를 소스로 사용하는 인바운드 규칙을 생성합니다. 이 보안 그룹에서는 클라이언트 애플리케이션이 이 보안 그룹을 사용하는 VPC의 EC2 인스턴스에 연결할 수 있습니다.
2. 애플리케이션에 대한 EC2 인스턴스를 생성하고 이전 단계에서 생성한 VPC 보안 그룹(`sg-0123ec2example`)에 EC2 인스턴스를 추가합니다.
3. 두 번째 VPC 보안 그룹(예: `sg-6789rdsexample`)을 생성하고 1단계에서 만든 VPC 보안 그룹(`sg-0123ec2example`)을 소스로 지정해 새 규칙을 생성합니다.
4. 새 DB 클러스터를 만들고 이전 단계에서 만든 VPC 보안 그룹(`sg-6789rdsexample`)에 DB 클러스터를 추가합니다. DB 클러스터를 만들 때 3단계에서 생성한 VPC 보안 그룹(`sg-6789rdsexample`) 규칙에 대해 지정한 것과 동일한 보안 포트 번호를 사용합니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.



이 시나리오의 VPC 구성에 대한 자세한 지침은 자습서: DB 클러스터에 사용할 Amazon VPC 생성 (IPv4 전용) 섹션을 참조하세요. VPC 사용에 대한 자세한 내용은 Amazon VPC 및 Amazon Aurora 섹션을 참조하세요.

VPC 보안 그룹 생성

VPC 콘솔을 사용하여 DB 인스턴스에 대한 VPC 보안 그룹을 생성할 수 있습니다. 보안 그룹 생성에 대한 자세한 내용은 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다. 및 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹을 참조하십시오.

보안 그룹과 DB 클러스터의 연결

보안 그룹은 RDS 콘솔의 [클러스터 수정(Modify cluster)] 옵션을 사용하거나, `ModifyDBCluster` Amazon RDS API 또는 `modify-db-cluster` AWS CLI 명령을 사용해 DB 클러스터와 연결할 수 있습니다.

다음 CLI 예제는 특정 VPC 그룹을 연결하고 DB 클러스터에서 DB 보안 그룹을 제거합니다.

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

마스터 사용자 계정 권한

새로운 클러스터를 만들 때 사용되는 기본 마스터 사용자는 해당 DB 클러스터에 대한 특정 권한을 갖습니다. DB 클러스터가 만들어진 후에는 마스터 사용자 이름을 변경할 수 없습니다.

> **Important**
>
> 애플리케이션에서 직접 마스터 사용자를 사용하지 않는 것이 좋습니다. 대신에 애플리케이션에 필요한 최소 권한으로 생성한 데이터베이스 사용자를 사용하는 모범 사례를 준수하십시오.

> **Note**
>
> 마스터 사용자의 권한을 실수로 삭제한 경우, DB 클러스터를 수정하고 새 마스터 사용자 암호를 설정하여 복원할 수 있습니다. DB 클러스터를 수정하는 것에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 섹션을 참조하세요.

다음 표에는 마스터 사용자가 각 데이터베이스 엔진에 대해 갖는 권한 및 데이터베이스 역할이 나와 있습니다.

데이터베이스 엔진	시스템 권한	데이터베이스 역할
Aurora MySQL	버전 2: ALTER,ALTER ROUTINE, CREATE,CREATE ROUTINE,CREATE TEMPORARY TABLES,CREATE USER,CREATE VIEW,DELETE,DROP, EVENT,EXECUTE,GRANT OPTION,INDEX,INSERT, LOAD FROM S3,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATI ON CLIENT , REPLICATION SLAVE ,SELECT, SELECT INTO S3,SHOW DATABASES , SHOW VIEW,TRIGGER, UPDATE	—
	버전 3: ALTER,APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE,CONNECTION_ADMIN , CREATE,CREATE ROLE,CREATE ROUTINE,CREATE TEMPORARY TABLES, CREATE USER,CREATE VIEW, DELETE,DROP,DROP ROLE, EVENT,EXECUTE,INDEX, INSERT,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATI ON SLAVE ,ROLE_ADMIN , SET_USER_ ID ,SELECT,SHOW DATABASES ,SHOW VIEW,TRIGGER, UPDATE,XA_RECOVER_ADMIN Aurora MySQL 버전 3.04.0부터 마스터 사용자에게도 SHOW_ROUTINE 권한이 부여됩니다. Aurora MySQL 버전 3.09.0부터 마스터 사용자에게도 FLUSH_OPTIMIZER_COSTS , FLUSH_STATUS , FLUSH_TABLES 및 FLUSH_USER_RESOURCES 권한이 부여됩니다.	rds_superuser_role rds_superuser_role에 대한 자세한 정보는 역할 기반 권한 모델 섹션을 참조하세요.

데이터베이스 엔진	시스템 권한	데이터베이스 역할
Aurora PostgreSQL	`LOGIN`,`NOSUPERUSER` , `INHERIT`,`CREATEDB`, `CREATEROLE` ,`NOREPLICATION` ,`VALID UNTIL 'infinity'`	`RDS_SUPERUSER` RDS_SUPERUSER에 대한 자세한 내용은 PostgreSQL 역할 및 권한 이해 섹션을 참조하세요.

Amazon Aurora에 서비스 연결 역할 사용

Amazon Aurora는 AWS Identity and Access Management(IAM) 서비스 연결 역할 사용 서비스 연결 역할은 Amazon Aurora에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon Aurora에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon Aurora를 더 쉽게 설정할 수 있습니다. Amazon Aurora에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않으면, Amazon Aurora만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며, 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 역할의 관련 리소스를 삭제해야만 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 부주의로 삭제할 수 없기 때문에 Amazon Aurora 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하고 서비스 연결 역할(Service-Linked Role) 열에 예(Yes)가 표시된 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon Aurora에 대한 서비스 연결 역할 권한

Amazon Aurora에서는 AWSServiceRoleForRDS라는 서비스 연결 역할을 사용하여 Amazon RDS가 DB 클러스터를 대신하여 AWS 서비스를 호출하도록 허용합니다.

AWSServiceRoleForRDS 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `rds.amazonaws.com`

이 서비스 연결 역할에는 계정에서 운영할 수 있는 권한을 부여하는 `AmazonRDSServiceRolePolicy`라는 권한 정책이 연결되어 있습니다.

JSON 정책 문서를 포함하여 이 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonRDSServiceRolePolicy를 참조하세요.

> **Note**
>
> IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 다음 오류 메시지가 표시되는 경우:
> 리소스를 만들 수 없습니다. 서비스 연결 역할을 생성할 권한이 있는지 확인하십시오. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.

다음 권한이 활성화되어 있는지 확인하십시오.

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon Aurora에 대한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. DB 클러스터를 생성하면 Amazon Aurora에서 서비스 연결 역할을 생성합니다.

> **Important**
>
> 서비스 연결 역할을 지원하기 시작한 2017년 12월 1일 이전에 Amazon Aurora 서비스를 사용하고 있었다면, Amazon Aurora가 계정에 AWSServiceRoleForRDS 역할을 생성했습니다. 자세한 내용은 내 AWS 계정에 표시되는 새 역할을 참조하세요.

이 서비스 연결 역할을 삭제한 다음 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. DB 클러스터를 생성하면 Amazon Aurora에서 서비스 연결 역할을 다시 생성합니다.

Amazon Aurora에 대한 서비스 연결 역할 편집

Amazon Aurora는 AWSServiceRoleForRDS 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

Amazon Aurora에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제할 것을 권합니다. 이렇게 하면 적극적으로 모니터링하거나 유지 관리하지 않은 미사용 엔터티가 없습니다. 그러나 서비스 연결 역할을 삭제하려면 먼저 모든 DB 클러스터를 삭제해야 합니다.

서비스 연결 역할 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에 활성 세션이 없는지 확인하고 역할에서 사용되는 리소스를 모두 제거해야 합니다.

IAM 콘솔에서 서비스 연결 역할에 활성 세션이 있는지 확인하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할(Roles)을 선택합니다. 그런 다음 AWSServiceRoleForRDS 역할의 이름(확인란 아님)을 선택합니다.
3. 선택한 역할의 요약 페이지에서 마지막 액세스 탭을 선택합니다.
4. 마지막 액세스 탭에서 서비스 연결 역할의 최근 활동을 검토합니다.

> **Note**
>
> Amazon Aurora에서 AWSServiceRoleForRDS 역할을 사용하는지 잘 모를 경우에는 역할을 삭제할 수 있습니다. 서비스에서 역할을 사용하는 경우에는 삭제가 안 되어 역할이 사용 중인 AWS 리전을 볼 수 있습니다. 역할이 사용 중인 경우에는 세션이 종료될 때까지 기다렸다가 역할을 삭제해야 합니다. 서비스 연결 역할에 대한 세션은 취소할 수 없습니다.

AWSServiceRoleForRDS 역할을 제거하려면 먼저 모든 DB 클러스터를 삭제해야 합니다.

모든 클러스터 삭제

다음 절차 중 하나에 따라 클러스터 하나를 삭제합니다. 클러스터 각각에 대해 이 절차를 반복합니다.

클러스터를 삭제하려면(콘솔)

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. Databases(데이터베이스) 목록에서, 삭제하려는 클러스터를 선택합니다.
3. 클러스터 작업에서 삭제를 선택합니다.

4. Delete(삭제)를 선택합니다.

클러스터를 삭제하려면(CLI)

AWS CLI 명령 참조에서 `delete-db-cluster` 섹션을 참조하세요.

클러스터를 삭제하려면(API)

Amazon RDS API Reference의 `DeleteDBCluster` 참조.

IAM 콘솔, IAM CLI 또는 IAM API를 사용하여 AWSServiceRoleForRDS 서비스 연결 역할을 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

Amazon RDS 베타에 대한 서비스 연결 역할 권한

Amazon Aurora는 `AWSServiceRoleForRDSBeta`라는 서비스 연결 역할을 사용하여 Amazon Aurora가 RDS DB 리소스를 대신하여 AWS 서비스를 직접 호출할 수 있도록 지원합니다.

AWSServiceRoleForRDSBeta 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `rds.amazonaws.com`

이 서비스 연결 역할에는 계정에서 운영할 수 있는 권한을 부여하는 `AmazonRDSBetaServiceRolePolicy`라는 권한 정책이 연결되어 있습니다. 자세한 내용은 AWS 관리형 정책: AmazonRDSBetaServiceRolePolicy 섹션을 참조하세요.

> **Note**
>
> IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 다음 오류 메시지가 표시되는 경우:
> 리소스를 만들 수 없습니다. 서비스 연결 역할을 생성할 권한이 있는지 확인하십시오. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.
> 다음 권한이 활성화되어 있는지 확인하십시오.
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSBetaServiceRolePolicy",
>     "Condition": {
> ```

```
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon RDS 미리 보기에 대한 서비스 연결 역할

Amazon Aurora는 `AWSServiceRoleForRDSPreview`라는 서비스 연결 역할을 사용하여 Amazon Aurora가 RDS DB 리소스를 대신하여 AWS 서비스를 직접 호출할 수 있도록 지원합니다.

AWSServiceRoleForRDSPreview 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `rds.amazonaws.com`

이 서비스 연결 역할에는 계정에서 운영할 수 있는 권한을 부여하는 `AmazonRDSPreviewServiceRolePolicy`라는 권한 정책이 연결되어 있습니다. 자세한 내용은 AWS 관리형 정책: AmazonRDSPreviewServiceRolePolicy 섹션을 참조하세요.

> **Note**
>
> IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 다음 오류 메시지가 표시되는 경우:
> 리소스를 만들 수 없습니다. 서비스 연결 역할을 생성할 권한이 있는지 확인하십시오. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.
> 다음 권한이 활성화되어 있는지 확인하십시오.

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSPreviewServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
```

```
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon VPC 및 Amazon Aurora

Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 Aurora DB 클러스터와 같은 AWS 리소스를 Virtual Private Cloud(VPC)로 시작할 수 있습니다.

VPC를 사용하면 가상 네트워킹 환경을 완벽하게 제어할 수 있습니다. 자기만의 IP 주소 범위를 선택하고, 서브넷을 생성하고, 라우팅 및 액세스 제어 목록을 구성할 수 있습니다. DB 클러스터를 VPC에서 실행하는 데는 추가 비용이 들지 않습니다.

계정에는 기본 VPC가 있습니다. 달리 지정하지 않는 한 새 DB 클러스터는 모두 기본 VPC에서 생성됩니다.

주제

다음에서는 Amazon Aurora DB 클러스터와 관련된 VPC 기능에 대한 토론을 찾을 수 있습니다. Amazon VPC 대한 자세한 내용은 Amazon VPC 시작 안내서 및 Amazon VPC 사용 설명서를 참조하세요.

VPC에서 DB 클러스터를 사용한 작업

DB 클러스터는 Virtual Private Cloud(VPC) 내에 있어야 합니다. VPC 는 AWS 클라우드의 다른 가상 네트워크에서 논리적으로 격리된 가상 네트워크입니다. Amazon VPC를 사용하면 Amazon Aurora DB 클러스터 또는 Amazon EC2 인스턴스와 같은 AWS 리소스를 VPC로 시작할 수 있습니다. VPC는 계정과 함께 제공되는 기본 VPC일 수도 있고, 사용자가 만들 수도 있습니다. 모든 VPC는 AWS 계정과 연결됩니다.

기본 VPC에는 VPC 내의 리소스를 격리하는 데 사용할 수 있는 3개의 서브넷이 있습니다. 또한 VPC 외부에서 VPC 내부의 리소스에 액세스할 수 있도록 하는 데 사용할 수 있는 인터넷 게이트웨이도 있습니다.

Amazon Aurora DB 클러스터를 포함하는 시나리오 목록은 VPC에서 DB 클러스터에 액세스하는 시나리오 단원을 참조하세요.

주제

- VPC에서 DB 클러스터를 사용한 작업
- DB 서브넷 그룹을 사용한 작업
- 공유 서브넷
- Amazon Aurora IP 주소 지정
- VPC에 있는 DB 클러스터를 인터넷에서 숨기기
- VPC에 DB 클러스터 만들기

다음 자습서에서는 일반적인 Amazon Aurora 시나리오에 사용할 수 있는 VPC를 생성하는 방법을 배울 수 있습니다.

- 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)
- 자습서: DB 클러스터(듀얼 스택 모드)에 사용할 VPC 생성

VPC에서 DB 클러스터를 사용한 작업

다음은 VPC에서 DB 클러스터를 사용하여 작업할 때 유용한 몇 가지 팁입니다.

- VPC는 최소 2개 이상의 서브넷을 보유해야 합니다. 이러한 서브넷은 DB 클러스터를 배포하고자 하는 AWS 리전에 있는 2개의 다른 가용 영역에 있어야 합니다. 서브넷은 사용자가 지정할 수 있고 보안 및 운영상의 필요를 바탕으로 클러스터를 그룹화할 수 있게 해주는 VPC IP 주소 범위의 한 부분입니다.
- VPC에서 DB 클러스터에 공개적으로 액세스할 수 있도록 하려면 VPC 속성인 DNS 호스트 이름과 DNS 확인을 활성화해야 합니다.
- VPC에는 사용자가 만드는 DB 서브넷 그룹이 있어야 합니다. 생성한 서브넷을 지정하여 DB 서브넷 그룹을 생성합니다. Amazon Aurora는 DB 클러스터의 기본 DB 인스턴스와 연결할 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다. 기본 DB 인스턴스는 서브넷이 포함된 가용 영역을 사용합니다.
- VPC에는 DB 클러스터에 대한 액세스를 허용하는 VPC 보안 그룹이 있어야 합니다.

  자세한 내용은 VPC에서 DB 클러스터에 액세스하는 시나리오 단원을 참조하세요.
- 각 서브넷의 CIDR 블록은 장애 조치와 컴퓨팅 조정을 포함한 유지 관리 활동 중에 Amazon Aurora가 사용할 예비 IP 주소를 수용할 만큼 충분히 커야 합니다. 예를 들어, 일반적으로 10.0.0.0/24 및 10.0.1.0/24와 같은 범위는 충분히 큽니다.
- VPC의 인스턴스 테넌시 속성 값은 기본 또는 전용 중 하나일 수 있습니다. 모든 기본 VPC에서는 인스턴스 테넌시 속성이 기본값으로 설정되어 있으며, 기본 VPC는 어떤 DB 인스턴스 클래스라도 지원할 수 있습니다.

인스턴스 테넌시 속성이 전용으로 설정된 전용 VPC에 DB 클러스터를 사용하도록 선택한 경우 DB 클러스터의 DB 인스턴스 클래스는 승인된 Amazon EC2 전용 인스턴스 유형 중 하나여야 합니다. 예를 들어 r5.large EC2 전용 인스턴스는 db.r5.large DB 인스턴스 클래스에 해당합니다. VPC의 인스턴스 테넌시에 대한 자세한 내용은 Amazon Elastic Compute Cloud 사용 설명서의 전용 인스턴스를 참조하세요.

전용 인스턴스에 존재할 수 있는 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 요금 페이지에서 Amazon EC2 전용 인스턴스를 참조하세요.

> **Note**
>
> 인스턴스 테넌시 속성을 DB 클러스터에 대해 전용으로 설정하면 DB 클러스터가 전용 호스트에서 실행된다는 보장이 없습니다.

DB 서브넷 그룹을 사용한 작업

서브넷은 사용자가 보안 및 운영상의 필요를 바탕으로 리소스를 그룹화하기 위해 지정하는 VPC IP 주소 범위의 특정 부분입니다. DB 서브넷 그룹은 사용자가 VPC에서 만든 다음 DB 클러스터에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. DB 서브넷 그룹을 사용하면 AWS CLI 또는 API를 사용하여 DB 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. 콘솔을 사용하는 경우 사용할 VPC와 서브넷 그룹을 선택할 수 있습니다.

각 DB 서브넷 그룹은 지정된 AWS 리전에서 두 개 이상의 가용 영역에 서브넷이 있어야 합니다. VPC에서 DB 클러스터를 생성할 때 이에 대한 DB 서브넷 그룹을 선택합니다. DB 서브넷 그룹에서 Amazon Aurora는 DB 클러스터의 기본 DB 인스턴스와 연결할 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다. DB는 서브넷이 포함된 가용 영역을 사용합니다. Aurora 는 항상 사용 가능한 IP 주소 공간이 있는 서브넷에서 IP 주소를 할당합니다.

DB 서브넷 그룹의 서브넷은 퍼블릭 또는 프라이빗입니다. 서브넷은 네트워크 액세스 제어 목록(ACL) 및 라우팅 테이블에 대해 설정한 구성에 따라 퍼블릭 또는 프라이빗입니다. DB 클러스터에 공개적으로 액세스할 수 있도록 하려면 DB 서브넷 그룹의 모든 서브넷이 퍼블릭이어야 합니다. 공개적으로 액세스할 수 있는 DB 클러스터와 연결된 서브넷이 퍼블릭에서 프라이빗으로 변경되면 DB 클러스터 가용성에 영향을 줄 수 있습니다.

듀얼 스택 모드를 지원하는 DB 서브넷 그룹을 만들려면 DB 서브넷 그룹에 추가하는 각 서브넷에 인터넷 프로토콜 버전 6(IPv6) CIDR 블록이 연결되어 있는지 확인합니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon Aurora IP 주소 지정 및 IPv6로 마이그레이션을 참조하세요.

Amazon Aurora가 VPC에 DB 클러스터를 생성할 때는 DB 서브넷 그룹의 IP 주소를 사용하여 DB 클러스터에 네트워크 인터페이스를 할당합니다. 하지만 기본 IP 주소가 장애 조치 중에 바뀌기 때문에 DB 클러스터에 연결할 때 반드시 도메인 이름 시스템(DNS) 이름을 사용하는 것이 좋습니다. 장애 조치 중에 기본 IP 주소가 변경되므로 이 방법을 사용하는 것이 좋습니다.

> **Note**
>
> VPC에서 실행하는 각각의 DB 클러스터에 대해 Amazon Aurora가 복구 작업에 사용할 수 있도록 DB 서브넷 그룹의 각 서브넷에 한 개 이상의 주소를 예약해야 합니다.

공유 서브넷

공유 VPC에서 DB 클러스터를 생성할 수 있습니다.

공유 VPC를 사용할 때 염두에 두어야 할 몇 가지 고려 사항은 다음과 같습니다.

- DB 클러스터를 공유 VPC 서브넷에서 비공유 VPC 서브넷으로 또는 그 반대로 이동할 수 있습니다.
- 공유 VPC의 참여자는 VPC에 보안 그룹을 만들어야 DB 클러스터를 생성할 수 있습니다.
- 공유 VPC의 소유자와 참여자는 SQL 쿼리를 사용하여 데이터베이스에 액세스할 수 있습니다. 그러나 리소스 생성자만 리소스에 대한 API 호출을 수행할 수 있습니다.

Amazon Aurora IP 주소 지정

IP 주소가 있으면 VPC 내의 리소스가 서로 통신하고, 인터넷을 통해 다른 리소스와 통신할 수 있습니다. Amazon Aurora는 IPv4와 IPv6 주소 지정 프로토콜을 모두 지원합니다. 기본적으로 Amazon Aurora와 Amazon VPC는 IPv4 주소 지정 프로토콜을 사용합니다. 이 동작은 끌 수 없습니다. VPC를 생성할 때 VPC에 IPv4 CIDR 블록(프라이빗 IPv4 주소)를 지정해야 합니다. IPv6 CIDR 블록을 VPC와 서브넷에 할당하고 그 블록에 속한 IPv6 주소를 서브넷의 DB 클러스터에 할당할 수도 있습니다.

IPv6 프로토콜을 지원하면 지원되는 IP 주소 수가 늘어납니다. IPv6 프로토콜을 사용하면 향후에 인터넷이 성장할 때를 대비해 사용 가능한 주소를 충분히 확보할 수 있습니다. 신규 및 기존 RDS 리소스는 VPC 내에서 IPv4 및 IPv6 주소를 사용할 수 있습니다. 애플리케이션의 다른 부분에서 사용되는 두 프로토콜 간의 네트워크 트래픽을 구성, 보안 및 변환하면 운영 오버헤드가 발생할 수 있습니다. Amazon RDS 리소스의 IPv6 프로토콜을 표준화하여 네트워크 구성을 간소화할 수 있습니다.

주제

- IPv4 주소
- IPv6 주소
- 듀얼 스택 모드

IPv4 주소

VPC를 생성할 때 VPC의 IPv4 주소 범위를 `10.0.0.0/16`와 같은 CIDR 블록 형태로 지정해야 합니다. DB 서브넷 그룹은 이 CIDR 블록에서 DB 클러스터가 사용할 수 있는 IP 주소의 범위를 정의합니다. 이 IP 주소는 프라이빗 또는 퍼블릭일 수 있습니다.

프라이빗 IPv4 주소는 인터넷을 통해 연결할 수 없는 IP 주소입니다. 프라이빗 IPv4 주소는 DB 클러스터 및 같은 VPC 내의 Amazon EC2 인스턴스와 같은 기타 리소스 간의 통신을 위해 사용될 수 있습니다. 각 DB 클러스터에는 VPC 내 통신을 위한 프라이빗 IP 주소가 있습니다.

퍼블릭 IP 주소는 인터넷을 통해 연결할 수 있는 IPv4 주소입니다. 퍼블릭 주소는 DB 클러스터 및 SQL 클라이언트와 같이 인터넷상 리소스 간의 통신을 위해 사용될 수 있습니다. DB 클러스터가 퍼블릭 IP 주소를 수신할지 여부를 제어할 수 있습니다.

일반 Amazon Aurora 시나리오에서 사용할 수 있는 IPv4 주소만을 사용하여 VPC를 생성하는 방법을 보여주는 자습서는 자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용) 섹션을 참조하세요.

IPv6 주소

IPv6 CIDR 블록을 VPC와 서브넷에 연결하고 그 블록에 속한 IPv6 주소를 VPC의 리소스에 할당할 수도 있습니다. 각 IPv6 주소는 전역적으로 고유합니다.

VPC에 대한 IPv6 CIDR 블록은 Amazon의 IPv6 주소 풀에서 자동으로 할당되므로 범위를 직접 선택할 수는 없습니다.

IPv6 주소에 연결할 때 다음 조건이 충족되는지 확인하세요.

- IPv6를 통한 클라이언트에서 데이터베이스로의 트래픽이 허용되도록 클라이언트가 구성됩니다.
- DB 인스턴스에서 사용하는 RDS 보안 그룹은 IPv6를 통한 클라이언트에서 데이터베이스로의 트래픽이 허용되도록 올바르게 구성됩니다.
- 클라이언트 운영 체제 스택은 IPv6 주소의 트래픽을 허용하며 운영 체제 드라이버 및 라이브러리는 올바른 기본 DB 인스턴스 엔드포인트(IPv4 또는 IPv6)를 선택하도록 구성됩니다.

IPv6에 대한 자세한 내용은 Amazon VPC 사용 설명서의 IP 주소 지정 섹션을 참조하세요.

듀얼 스택 모드

DB 클러스터가 IPv4 및 IPv6 주소 지정 프로토콜 모두를 통해 통신할 수 있을 때 듀얼 스택 모드로 실행됩니다. 따라서 리소스는 IPv4, IPv6 또는 모두를 통해 DB 클러스터와 통신할 수 있습니다. RDS는 프라이빗 듀얼 스택 모드 DB 인스턴스의 IPv6 엔드포인트에 대한 인터넷 게이트웨이 액세스를 비활성화합니다. RDS는 IPv6 엔드포인트가 프라이빗이며 VPC 내에서만 액세스할 수 있도록 하기 위해 이 작업을 수행합니다.

주제

- 듀얼 스택 모드 및 DB 서브넷 그룹
- 듀얼 스택 모드 DB 인스턴스 작업
- 듀얼 스택 모드를 사용하도록 IPv4 전용 DB 클러스터 수정
- 듀얼 스택 네트워크 DB 클러스터의 가용성
- 듀얼 스택 네트워크 DB 클러스터에 대한 제한 사항

일반 Amazon Aurora 시나리오에서 사용할 수 있는 IPv4와 IPv6 주소 모두를 사용하여 VPC를 생성하는 방법을 보여주는 자습서는 자습서: DB 클러스터(듀얼 스택 모드)에 사용할 VPC 생성 섹션을 참조하세요.

듀얼 스택 모드 및 DB 서브넷 그룹

듀얼 스택 모드를 사용하려면 DB 클러스터와 연결하는 DB 서브넷 그룹의 각 서브넷에 IPv6 CIDR 블록이 연결되어 있어야 합니다. 새 DB 서브넷 그룹을 생성하거나 기존 DB 서브넷 그룹을 수정하여 이 요구 사항을 충족하도록 수정할 수 있습니다. DB 클러스터가 듀얼 스택 모드가 된 후에는 클라이언트가 정상적으로 연결할 수 있습니다. 클라이언트 보안 방화벽과 RDS DB 인스턴스 보안 그룹이 IPv6을 통한 트래픽을 허용하도록 정확하게 구성되어 있는지 확인합니다. 연결하기 위해 클라이언트는 DB 클러스터 기본 인스턴스의 엔드포인트를 사용합니다. 클라이언트 애플리케이션은 데이터베이스에 연결할 때 기본 프로토콜을 지정할 수 있습니다. 듀얼 스택 모드에서 DB 클러스터는 클라이언트의 기본 네트워크 프로토콜(IPv4 또는 IPv6)을 감지하고 연결에 해당 프로토콜을 사용합니다.

서브넷 삭제 또는 CIDR 연결 해제로 인해 DB 서브넷 그룹이 듀얼 스택 모드 지원을 중지하는 경우 DB 서브넷 그룹과 연결된 DB 인스턴스에 대해 네트워크 상태가 호환되지 않을 위험이 있습니다. 또한 새 듀얼 스택 모드 DB 클러스터를 생성할 때 DB 서브넷 그룹을 사용할 수 없습니다.

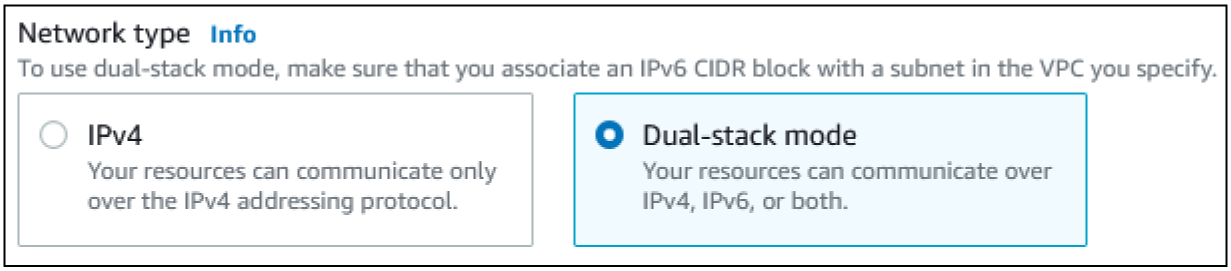
AWS Management Console을 사용하여 DB 서브넷 그룹이 듀얼 스택 모드를 지원하는지 알아보려면 DB 서브넷 그룹의 세부 정보 페이지에서 네트워크 유형(Network type)을 확인합니다. AWS CLI를 사용하여 DB 서브넷 그룹이 듀얼 스택 모드를 지원하는지 알아보려면 describe-db-subnet-groups 명령을 실행하고 출력에서 `SupportedNetworkTypes`를 확인합니다.

읽기 전용 복제본은 독립적인 DB 인스턴스로 취급되며 기본 DB 인스턴스와 다른 네트워크 유형을 가질 수 있습니다. 읽기 전용 복제본의 기본 DB 인스턴스의 네트워크 유형을 변경하는 경우 읽기 전용 복제본은 영향을 받지 않습니다. DB 인스턴스를 복원할 때 지원되는 모든 네트워크 유형으로 복원할 수 있습니다.

듀얼 스택 모드 DB 인스턴스 작업

DB 클러스터를 생성하거나 수정할 때 리소스가 DB 클러스터와 IPv4, IPv6 또는 둘 다를 통해 통신할 수 있도록 허용하기 위해 듀얼 스택 모드를 지정할 수 있습니다.

AWS Management Console을 사용하여 DB 인스턴스를 생성하거나 수정할 때 네트워크 유형 섹션에서 듀얼 스택 모드를 지정할 수 있습니다. 다음 이미지는 콘솔의 네트워크 유형 섹션을 보여줍니다.



AWS CLI를 사용하여 DB 클러스터를 생성하거나 수정할 때 듀얼 스택 모드를 사용하려면 `--network-type` 옵션을 `DUAL`로 설정합니다. 를 사용하여 DB 클러스터를 생성하거나 수정할 때 듀얼 스택 모드를 사용하려면 `NetworkType` 옵션을 `DUAL`로 설정합니다. DB 인스턴스의 네트워크 유형을 수정하는 경우 가동 중지가 발생할 수 있습니다. 지정된 DB 엔진 버전 또는 DB 서브넷 그룹에서 듀얼 스택 모드가 지원되지 않는 경우 `NetworkTypeNotSupported` 오류가 반환됩니다.

DB 클러스터 생성에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 생성](#) 단원을 참조하세요. DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정](#) 단원을 참조하세요.

콘솔을 사용하여 DB 클러스터가 이중 스택 모드인지 확인하려면 DB 클러스터의 연결 및 보안 탭에서 네트워크 유형을 확인하세요.

듀얼 스택 모드를 사용하도록 IPv4 전용 DB 클러스터 수정

듀얼 스택 모드를 사용하도록 IPv4 전용 DB 클러스터를 수정할 수 있습니다. 그러려면 DB 클러스터의 네트워크 유형을 변경합니다. 수정으로 인해 가동 중지가 발생할 수 있습니다.

유지 관리 기간 동안 Amazon Aurora DB 클러스터의 네트워크 유형을 변경하는 것이 좋습니다. 새 인스턴스의 네트워크 유형을 듀얼 스택 모드로 설정하는 것은 현재 지원되지 않습니다. `modify-db-cluster` 명령을 사용하여 네트워크 유형을 수동으로 설정할 수 있습니다.

듀얼 스택 모드를 사용하도록 DB 클러스터를 수정하기 전에 DB 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인하세요. DB 클러스터와 연결된 DB 서브넷 그룹이 듀얼 스택 모드를 지원하지 않는 경우 DB 클러스터를 수정할 때 이를 지원하는 다른 DB 서브넷 그룹을 지정하세요. DB 클러스터의 DB 서브넷 그룹을 수정하면 가동 중지가 발생할 수 있습니다.

듀얼 스택 모드를 사용하도록 DB 클러스터를 변경하기 전에 DB 클러스터의 DB 서브넷 그룹을 수정할 경우, DB 서브넷 그룹이 변경 전후에 DB 클러스터에 대해 유효한지 확인하세요.

Amazon Aurora 클러스터의 네트워크를 듀얼 스택 모드로 변경하려면 `DUAL` 값이 포함된 `--network-type` 파라미터만 사용하여 [modify-db-cluster](#) API를 실행하는 것이 좋습니다. 동일한 API 호출에서 다른 파라미터를 `--network-type` 파라미터와 함께 추가하면 가동 중지가 발생할 수 있습니다.

변경 후 DB 클러스터에 연결할 수 없는 경우 선택한 네트워크(IPv4 또는 IPv6)의 데이터베이스로의 트래픽을 허용하도록 클라이언트 및 데이터베이스 보안 방화벽과 라우팅 테이블이 정확하게 구성되어 있는지 확인하세요. IPv6 주소를 사용하여 연결하려면 운영 체제 파라미터, 라이브러리 또는 드라이버를 수정해야 할 수도 있습니다.

듀얼 스택 모드를 사용하도록 IPv4 전용 DB 클러스터 수정

1. 듀얼 스택 모드를 지원하도록 DB 서브넷 그룹을 수정하거나 듀얼 스택 모드를 지원하는 DB 서브넷 그룹을 생성합니다.

   a. IPv6 CIDR 블록을 VPC와 연결

      자세한 내용은 Amazon VPC 사용 설명서의 [VPC에 IPv6 CIDR 블록 추가](#)를 참조하세요.

   b. IPv6 CIDR 블록을 DB 서브넷 그룹의 모든 서브넷에 연결합니다.

      자세한 내용은 Amazon VPC 사용 설명서의 [서브넷에 IPv6 CIDR 블록 추가](#)를 참조하세요.

   c. DB 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인합니다.

      AWS Management Console을 사용 중인 경우 DB 서브넷 그룹을 선택하고 지원되는 네트워크 유형(Supported network types) 값을 듀얼, IPv4(Dual, IPv4)로 설정하세요.

      AWS CLI를 사용 중인 경우 [describe-db-subnet-groups](#) 명령을 실행하고 DB 인스턴스의 `SupportedNetworkType` 값을 `Dual, IPv4`로 설정하세요.

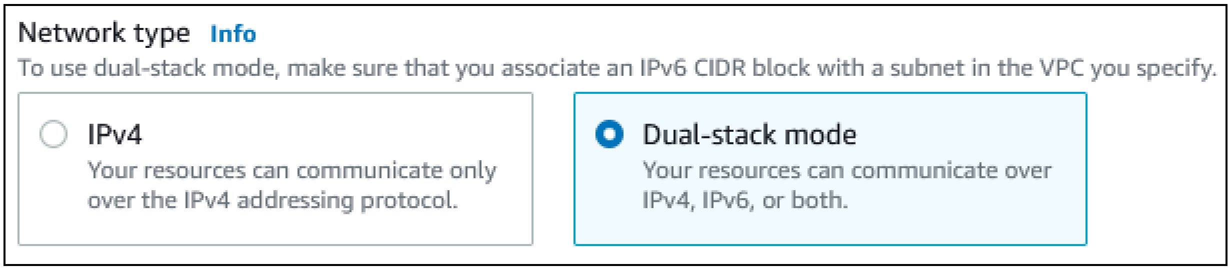
2. 데이터베이스에 대한 IPv6 연결을 허용하도록 DB 클러스터와 연결된 보안 그룹을 수정하거나 IPv6 연결을 허용하는 새 보안 그룹을 생성합니다.

   자세한 내용은 Amazon VPC 사용 설명서의 [보안 그룹 규칙](#)을 참조하세요.

3. 듀얼 스택 모드를 지원하도록 DB 클러스터를 수정합니다. 이렇게 하려면 네트워크 유형을 듀얼 스택 모드로 설정하세요.

   콘솔을 사용 중인 경우 다음 설정이 올바른지 확인합니다.

   - 네트워크 유형 – 듀얼 스택 모드

   - DB 서브넷 그룹 - 이전 단계에서 구성한 DB 서브넷 그룹
   - 보안 그룹 - 이전 단계에서 구성한 보안

   AWS CLI를 사용 중인 경우 다음 설정이 올바른지 확인합니다.

   - `--network-type` – `dual`
   - `--db-subnet-group-name` - 이전 단계에서 구성한 DB 서브넷 그룹
   - `--vpc-security-group-ids` - 이전 단계에서 구성한 VPC 보안 그룹

   예시:

```
aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
```

4. DB 클러스터가 듀얼 스택 모드를 지원하는지 확인합니다.

   콘솔을 사용하는 경우 DB 클러스터에 대한 구성 탭을 선택합니다. 해당 탭에서 네트워크 유형 값이 듀얼 스택 모드인지 확인합니다.

   AWS CLI을 사용 중인 경우 describe-db-instances 명령을 실행하고 DB 클러스터의 `NetworkType` 값을 `dual`로 설정하세요.

   라이터 DB 인스턴스 엔드포인트에서 `dig` 명령을 실행하여 연결된 IPv6 주소를 식별합니다.

```
dig db-instance-endpoint AAAA
```

   IPv6 주소가 아닌 라이터 DB 인스턴스 엔드포인트를 사용하여 DB 클러스터에 연결합니다.

듀얼 스택 네트워크 DB 클러스터의 가용성

듀얼 스택 네트워크 DB 클러스터는 다음을 제외한 모든 AWS 리전에서 사용할 수 있습니다.

- 아시아 태평양(하이데라바드)
- 아시아 태평양(말레이시아)
- 아시아 태평양(멜버른)
- 캐나다 서부(캘거리)
- 유럽(스페인)
- 유럽(취리히)
- 이스라엘(텔아비브)
- 중동(UAE)

다음 DB 엔진 버전은 듀얼 스택 네트워크 DB 클러스터를 지원합니다.

- Aurora MySQL 버전:
  - 3.02 이상의 3 버전
  - 2.09.1 이상의 2 버전

  Aurora MySQL 버전에 대한 자세한 내용은 Aurora MySQL 릴리스 정보를 참조하세요.
- Aurora PostgreSQL 버전
  - 15.2 이상의 모든 버전
  - 14.3 이상의 14 버전
  - 13.7 이상의 13 버전

  Aurora PostgreSQL 버전에 대한 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.

듀얼 스택 네트워크 DB 클러스터에 대한 제한 사항

듀얼 스택 네트워크 DB 클러스터에는 다음과 같은 제한이 적용됩니다.

- DB 클러스터는 IPv6 프로토콜을 단독으로 사용할 수 없습니다. IPv4를 단독으로 사용하거나 IPv4 및 IPv6 프로토콜(듀얼 스택 모드)을 사용할 수 있습니다.
- Amazon RDS에서는 네이티브 IPv6 서브넷을 지원하지 않습니다.

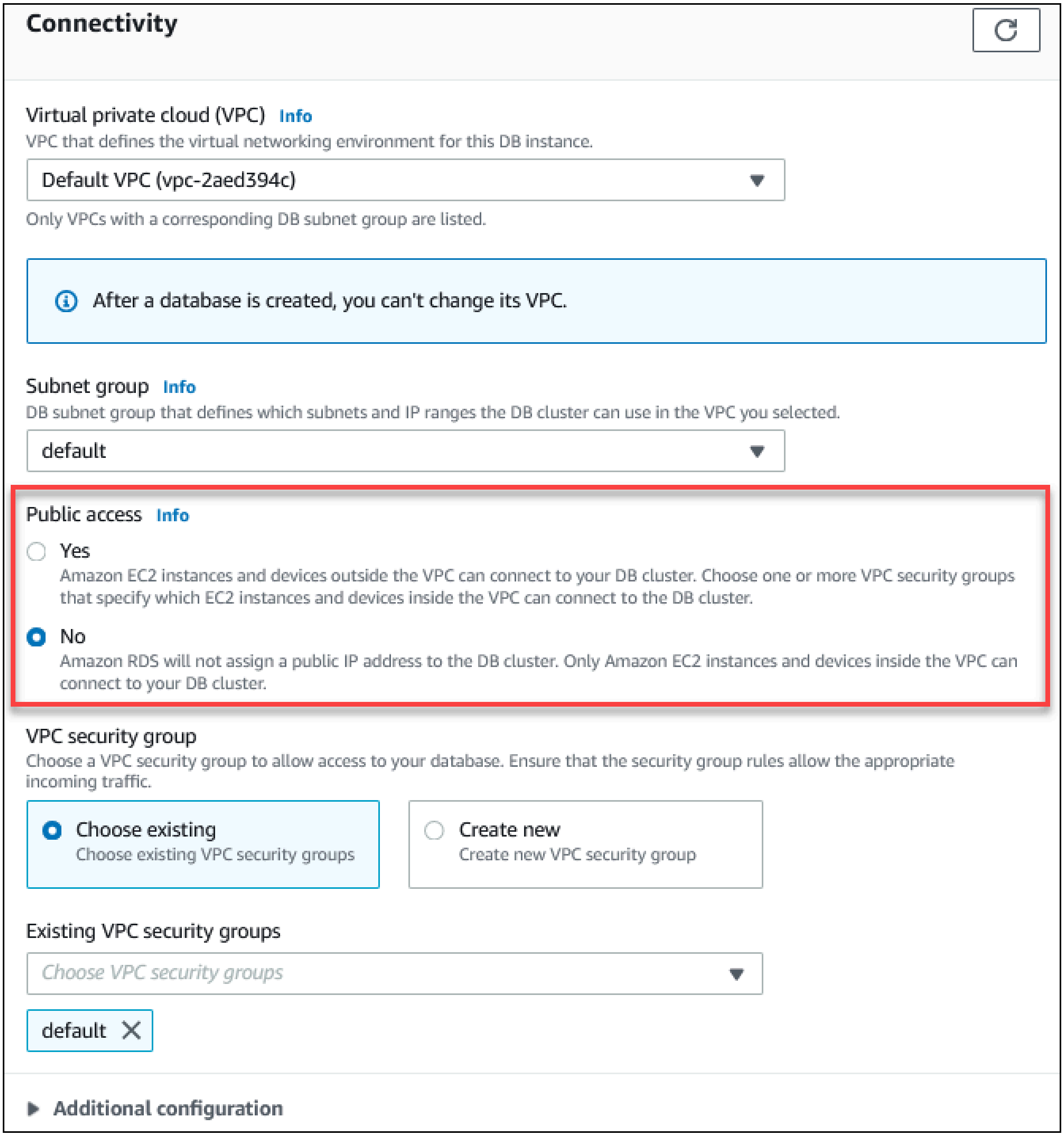
- 듀얼 스택 모드를 사용하는 DB 클러스터는 프라이빗이어야 합니다. 공개적으로 액세스할 수 없습니다.
- 듀얼 스택 모드 DB 클러스터에는 RDS 프록시를 사용할 수 없습니다.

VPC에 있는 DB 클러스터를 인터넷에서 숨기기

한 가지 일반적인 Amazon Aurora 시나리오는 일반에 공개되어 있는 웹 애플리케이션을 포함한 Amazon EC2 인스턴스와 공개적으로 액세스할 수 없는 데이터베이스를 포함한 DB 클러스터가 있는 VPC를 사용하는 경우입니다. 예를 들어, 퍼블릭 서브넷과 프라이빗 서브넷이 있는 VPC를 생성할 수 있습니다. 웹 서버로 작동하는 EC2 인스턴스는 퍼블릭 서브넷에 배포할 수 있습니다. DB 클러스터는 프라이빗 서브넷에 배포됩니다. 이런 배포 환경에서는 웹 서버만 DB 클러스터에 액세스할 수 있습니다. 이 시나리오에 대한 그림은 동일한 VPC에 있는 Amazon EC2 인스턴스가 VPC 내에 있는 DB 클러스터에 액세스 단원을 참조하세요.

VPC 내에서 DB 클러스터를 시작하면 DB 클러스터는 VPC 내부 트래픽에 대한 프라이빗 IP 주소를 가집니다. 이 프라이빗 IP 주소는 공개적으로 액세스할 수 없습니다. 퍼블릭 액세스 옵션을 사용하여 DB 클러스터에 프라이빗 IP 주소 외에 퍼블릭 IP 주소가 있는지 여부를 지정할 수 있습니다. DB 클러스터가 공개적으로 액세스 가능한 것으로 지정된 경우 해당 DNS 엔드포인트는 VPC 내에서 프라이빗 IP 주소로 확인됩니다. VPC 외부에서 퍼블릭 IP 주소로 확인됩니다. DB 클러스터에 대한 액세스는 궁극적으로 사용하는 보안 그룹에 의해 제어됩니다. DB 클러스터에 할당된 보안 그룹에 이를 허용하는 인바운드 규칙이 포함되어 있지 않으면 해당 퍼블릭 액세스가 허용되지 않습니다. 또한 DB 클러스터에 공개적으로 액세스할 수 있으려면 해당 DB 서브넷 그룹의 서브넷에 인터넷 게이트웨이가 있어야 합니다. 자세한 내용은 Amazon RDS DB 인스턴스에 연결할 수 없음 섹션을 참조하세요.

퍼블릭 액세스 옵션을 수정하여 퍼블릭 액세스를 켜거나 끄도록 DB 클러스터를 수정할 수 있습니다. 다음 그림은 추가 연결 구성 섹션의 퍼블릭 액세스 옵션을 보여줍니다. 옵션을 설정하려면 연결(Connectivity) 섹션에서 추가 연결 구성(Additional connectivity configuration) 섹션을 엽니다.



퍼블릭 액세스 옵션을 설정하기 위해 DB 인스턴스를 수정하는 방법에 대한 자세한 내용은 DB 클러스터에서 DB 인스턴스 수정 단원을 참조하세요.

VPC에 DB 클러스터 만들기

다음 절차에 따라 VPC에서 DB 클러스터를 생성할 수 있습니다. 기본 VPC를 사용하려면 2단계부터 시작하고 이미 생성된 VPC 및 DB 서브넷 그룹을 사용할 수 있습니다. 추가 VPC를 만들려면 새 VPC를 만들 수 있습니다.

> **Note**
>
> VPC의 DB 클러스터에 공개적으로 액세스할 수 있도록 하려면 VPC 속성 DNS 호스트 이름 및 DNS 확인을 활성화하여 VPC에 대한 DNS 정보를 업데이트해야 합니다. VPC 인스턴스의 DNS 정보 업데이트에 대한 자세한 내용은 VPC에 대한 DNS 지원 업데이트를 참조하세요.

다음 단계에 따라 VPC에서 DB 인스턴스를 만들 수 있습니다:

- 1단계: VPC 생성
- 2단계: DB 서브넷 그룹 만들기
- 3단계: VPC 보안 그룹 만들기
- 4단계: VPC에서 DB 인스턴스 만들기

1단계: VPC 생성

최소 2개의 가용 영역에 서브넷이 있는 VPC를 생성합니다. DB 서브넷 그룹을 만들 때 이들 서브넷이 사용됩니다. 기본 VPC가 있는 경우에는 해당 AWS 리전의 각 가용 영역에 서브넷이 자동으로 생성됩니다.

자세한 내용은 프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성 단원을 참조하거나 Amazon VPC 사용 설명서의 VPC 생성을 참조하세요.

2단계: DB 서브넷 그룹 만들기

DB 서브넷 그룹은 사용자가 VPC에 대해 만든 다음 DB 클러스터에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. DB 서브넷 그룹을 사용하면 AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. 콘솔을 사용하는 경우 사용할 VPC와 서브넷만 선택할 수 있습니다. 각 DB 서브넷 그룹은 해당 AWS 리전에서 두 개 이상의 가용 영역에 한 개 이상의 서브넷이 있어야 합니다. 모범 사례로서, 각 DB 서브넷 그룹에는 AWS 리전의 가용 영역마다 하나 이상의 서브넷이 있어야 합니다.

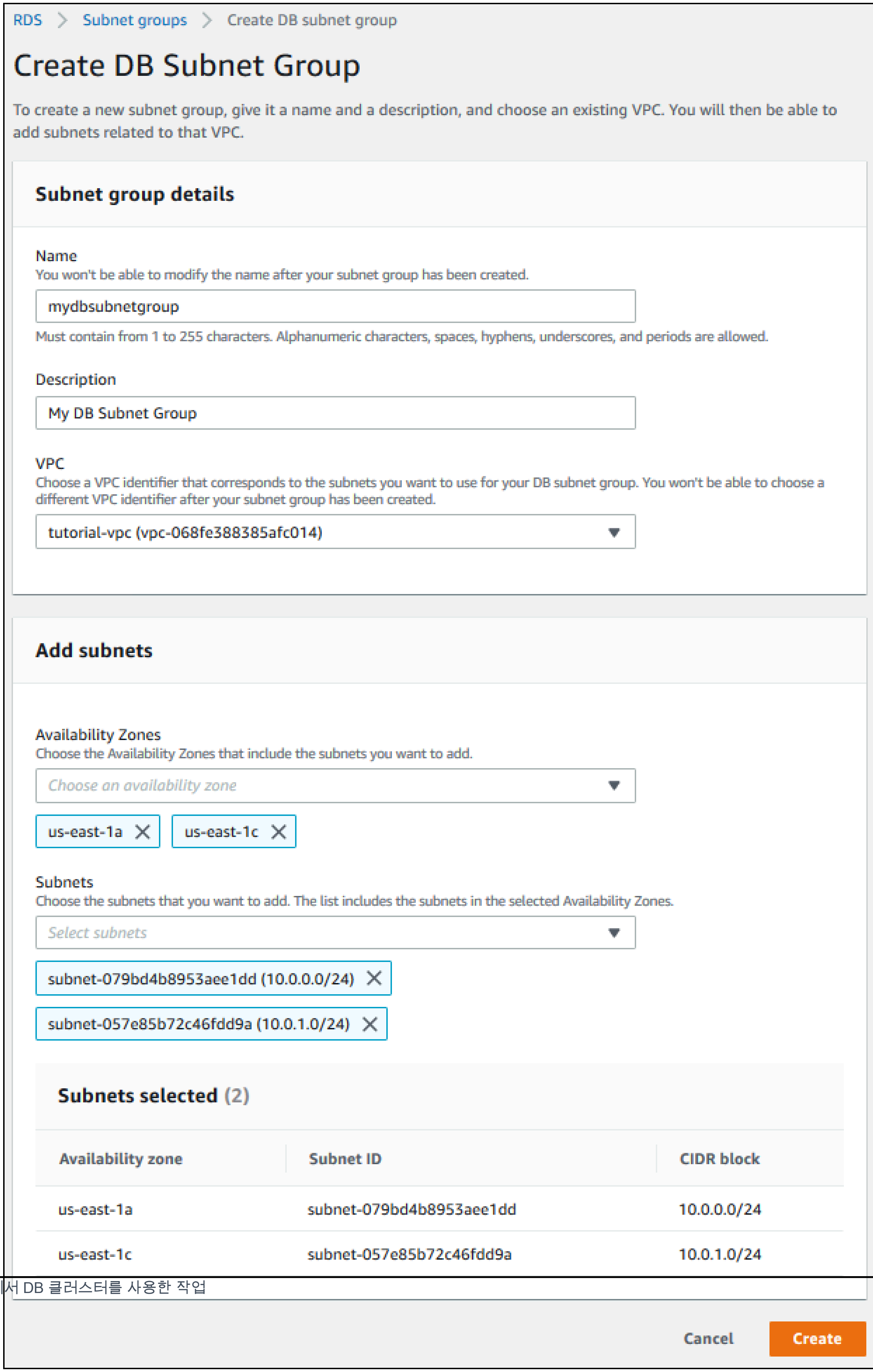
DB 클러스터에 공개적으로 액세스할 수 있도록 하려면 DB 서브넷 그룹의 서브넷에 인터넷 게이트웨이가 있어야 합니다. 서브넷용 인터넷 게이트웨이에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터넷 게이트웨이를 사용하여 인터넷에 연결을 참조하세요.

VPC에서 DB 클러스터를 생성할 때 이에 대한 DB 서브넷 그룹을 선택합니다. mazon Aurora는 DB 클러스터와 연결할 서브넷 내의 서브넷과 IP 주소를 선택합니다. DB 서브넷 그룹이 없는 경우 Amazon Aurora는 DB 클러스터를 생성할 때 기본 서브넷 그룹을 생성합니다. Amazon Aurora에서 탄력적 네트워크 인터페이스를 생성하고 해당 IP 주소로 DB 클러스터에 연결합니다. DB 클러스터는 서브넷이 포함된 가용 영역을 사용합니다.

이 단계에서는 DB 서브넷 그룹을 만들고 VPC용으로 만든 서브넷을 추가합니다.

DB 서브넷 그룹을 만드는 방법

1. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Subnet groups]를 선택합니다.
3. [Create DB Subnet Group]을 선택합니다.
4. [Name]에 DB 서브넷 그룹의 이름을 입력합니다.
5. 설명에 DB 서브넷 그룹에 대한 설명을 입력합니다.
6. VPC에서 기본 VPC 또는 생성한 VPC를 선택합니다.
7. 서브넷 추가 섹션의 가용 영역에서 해당 서브넷을 포함하는 가용 영역을 선택한 다음 서브넷에서 서브넷을 선택합니다.


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. 생성(Create)을 선택합니다.

    새 DB 서브넷 그룹은 RDS 콘솔의 DB 서브넷 그룹 목록에 나타납니다. DB 서브넷 그룹을 선택하면 창 하단에 있는 세부 정보 창에서 그룹과 연결된 모든 서브넷을 포함한 세부 정보를 확인할 수 있습니다.

3단계: VPC 보안 그룹 만들기

DB 클러스터를 만들기 전에 DB 클러스터와 연결할 VPC 보안 그룹을 만들어야 합니다. VPC 보안 그룹을 생성하지 않는 경우 DB 클러스터를 생성할 때 기본 보안 그룹을 사용할 수 있습니다. DB 클러스터에 대한 보안 그룹을 생성하는 방법에 대한 지침은 프라이빗 DB 클러스터에 대한 VPC 보안 그룹 생성 단원을 참조하거나 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 리소스에 대한 트래픽 제어를 참조하세요.

4단계: VPC에서 DB 인스턴스 만들기

이 단계에서는 DB 클러스터를 만들고 이전 단계에서 만든 VPC 이름, DB 서브넷 그룹 및 VPC 보안 그룹을 사용합니다.

> Note
>
> VPC의 DB 클러스터에 공개적으로 액세스할 수 있도록 하려면 VPC 속성 DNS 호스트 이름 및 DNS 확인을 활성화해야 합니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 VPC에 대한 DNS 속성을 참조하세요.

DB 클러스터 생성 방법에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하세요.

연결 섹션에 메시지가 표시되면 VPC 이름, DB 서브넷 그룹 및 VPC 보안 그룹을 입력합니다.

> Note
>
> 현재 DB 클러스터에 대해서는 VPC 업데이트가 지원되지 않습니다.

VPC에서 DB 클러스터에 액세스하는 시나리오

Amazon Aurora에서는 VPC의 DB 클러스터에 액세스하는 다음 시나리오를 지원합니다.

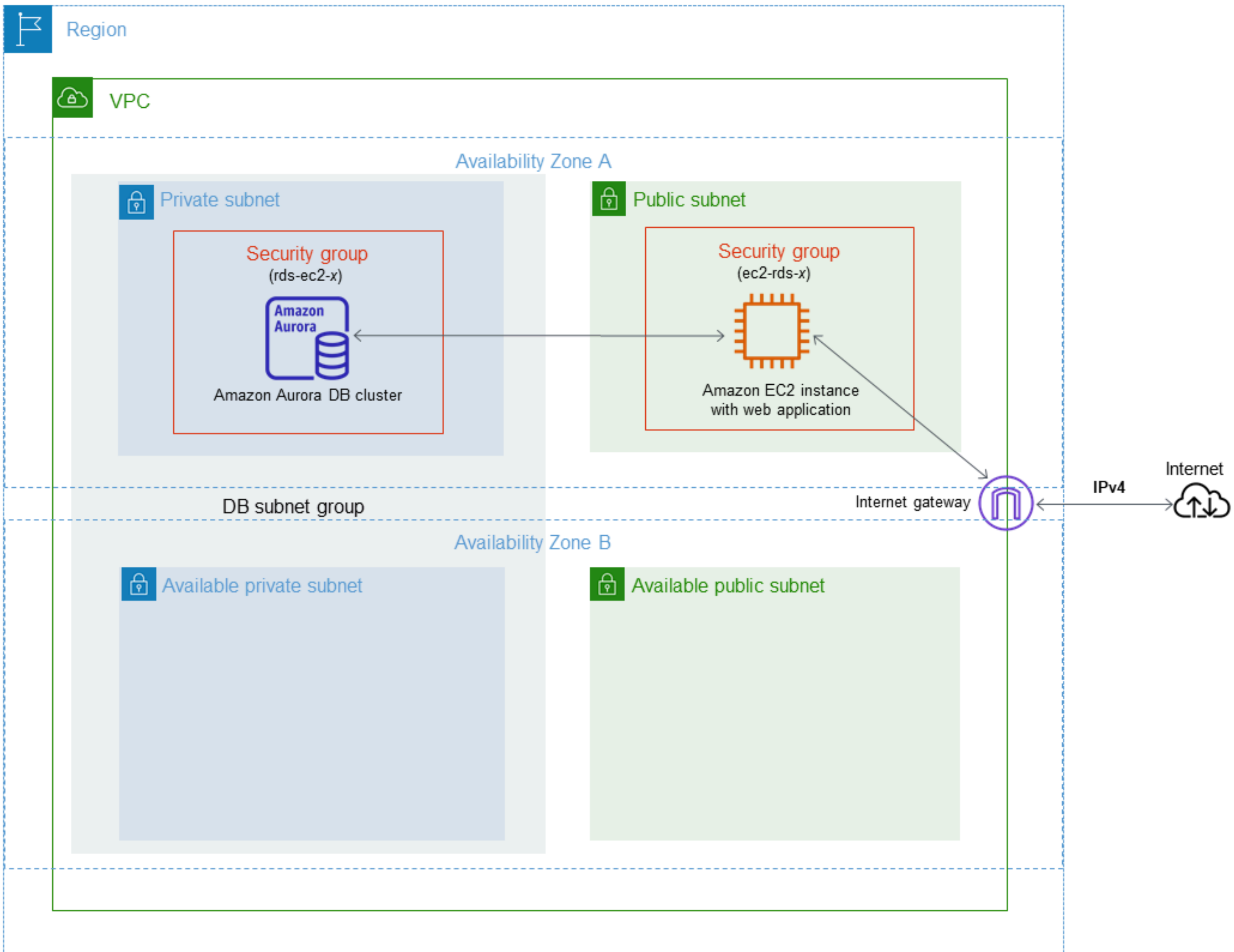
- 동일한 VPC에 있는 Amazon EC2 인스턴스

- [다른 VPC에 있는 EC2 인스턴스]
- [클라이언트 애플리케이션이 인터넷을 통해]
- [프라이빗 네트워크]

동일한 VPC에 있는 Amazon EC2 인스턴스가 VPC 내에 있는 DB 클러스터에 액세스

VPC에서 DB 클러스터를 사용하는 일반적인 사례는 동일한 VPC의 Amazon EC2 인스턴스에서 실행 중인 애플리케이션 서버와 데이터를 공유하는 것입니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.



동일한 VPC에서 EC2 인스턴스와 DB 클러스터 간 액세스를 관리하는 가장 간단한 방법은 다음과 같습니다.

- DB 클러스터가 포함될 VPC 보안 그룹을 생성합니다. 이 보안 그룹을 사용해 DB 클러스터에 대한 액세스를 제한할 수 있습니다. 예를 들어 이 보안 그룹에 대한 사용자 지정 규칙을 생성할 수 있습니다. 이렇게 하면 DB 클러스터를 생성할 때 할당한 포트와 개발 또는 기타 목적으로 DB 클러스터에 액세스하는 데 사용하는 IP 주소를 사용하여 TCP 액세스를 허용할 수 있습니다.
- EC2 인스턴스(웹 서버 및 클라이언트)가 포함될 VPC 보안 그룹을 생성합니다. 이 보안 그룹은 필요할 경우 VPC의 라우팅 테이블을 사용해 인터넷에서 EC2 인스턴스 액세스를 허용할 수 있습니다. 예를 들어, 이 보안 그룹에서 TCP가 포트 22를 통해 EC2 인스턴스에 액세스하도록 허용하는 규칙을 설정할 수 있습니다.
- DB 클러스터에 대한 보안 그룹에서 EC2 인스턴스에 대해 생성한 보안 그룹으로부터의 연결을 허용하는 사용자 지정 규칙을 만듭니다. 이러한 규칙을 통해 보안 그룹의 모든 구성원은 클러스터에 액세스할 수 있게 됩니다.

별도의 가용 영역에서 구성된 추가 퍼블릭 및 프라이빗 서브넷이 있습니다. RDS DB 서브넷 그룹은 최소 2개의 가용 영역에 한 개 이상의 서브넷이 필요합니다. 추가 서브넷을 사용하면 향후 다중 AZ DB 인스턴스 배포로 쉽게 전환할 수 있습니다.

이 시나리오에 대해 퍼블릭 서브넷과 프라이빗 서브넷 모두에서 VPC를 생성하는 방법을 보여주는 자습서는 [자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)](#) 단원을 참조하세요.

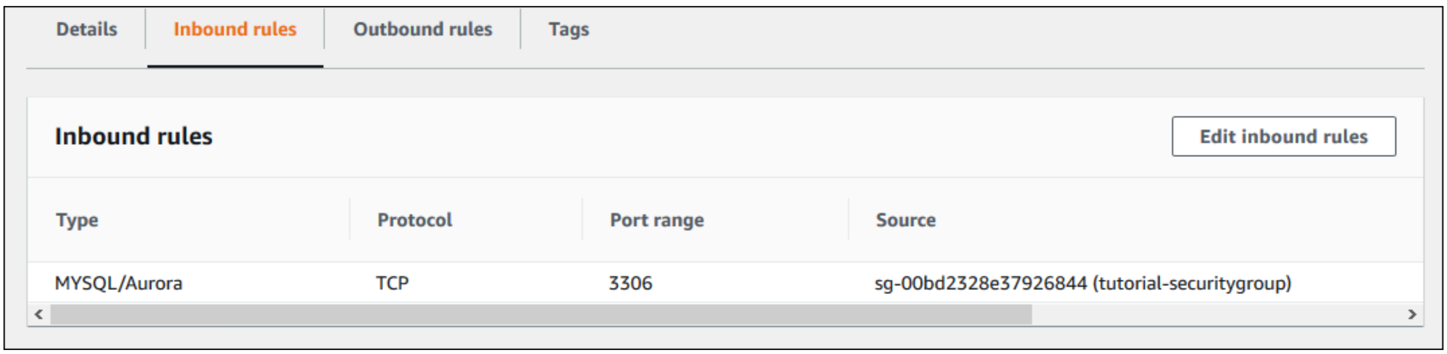
> **Tip**
>
> DB 클러스터를 생성할 때 Amazon EC2 인스턴스와 DB 클러스터 간의 네트워크 연결을 자동으로 설정할 수 있습니다. 자세한 내용은 [EC2 인스턴스와의 자동 네트워크 연결 구성](#) 단원을 참조하세요.

VPC 보안 그룹에서 다른 보안 그룹으로부터의 연결을 허용하는 규칙을 만들려면 다음을 수행합니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 Security groups를 선택합니다.
3. 다른 보안 그룹의 구성원에 대한 액세스를 허용할 보안 그룹을 선택하거나 생성합니다. 위 시나리오에서 이 보안 그룹을 DB 클러스터에 사용합니다. 인바운드 규칙 탭을 선택한 후 인바운드 규칙 편집을 선택합니다.
4. 인바운드 규칙 편집 페이지에서 규칙 추가]를 선택합니다.

5. 유형에서 MYSQL/Aurora와 같이 DB 클러스터를 생성할 때 사용한 포트에 해당하는 항목을 선택합니다.
6. 소스 상자에 일치하는 보안 그룹을 나열하는 보안 그룹의 ID를 입력합니다. 이 보안 그룹에서 보호 중인 리소스에 대한 액세스를 허용할 구성원이 포함된 보안 그룹을 선택합니다. 위 시나리오에서 이 보안 그룹을 EC2 인스턴스에 사용합니다.
7. 필요한 경우 유형으로 모든 TCP를 지정하고 소스 상자에 보안 그룹을 입력한 규칙을 생성하여 TCP 프로토콜에 대해 단계를 반복합니다. UDP 프로토콜을 사용하려는 경우 유형이 모든 UDP이고 소스에 보안 그룹이 있는 규칙을 생성합니다.
8. 규칙 저장을 선택합니다.

다음 화면은 소스에 대한 보안 그룹이 포함된 인바운드 규칙을 보여 줍니다.

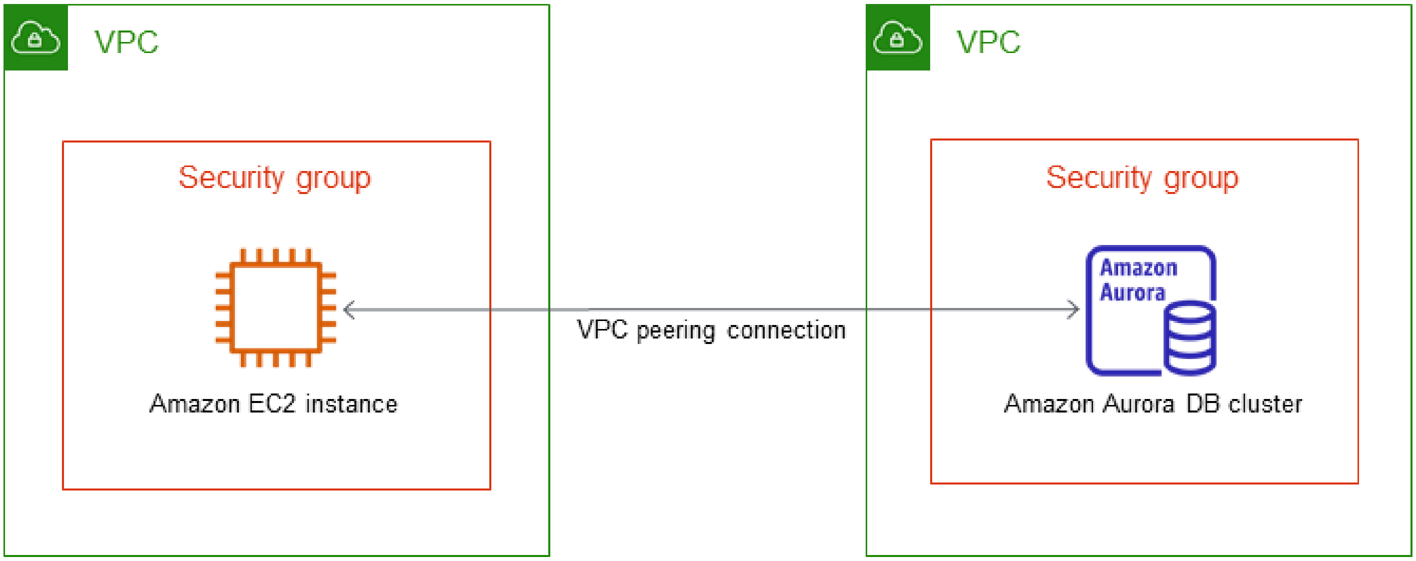


EC2 인스턴스에서 DB 클러스터에 연결하는 방법에 대한 자세한 내용은 Amazon Aurora DB 클러스터에 연결 단원을 참조하세요.

VPC에 있는 DB 클러스터에 다른 VPC에 있는 EC2 인스턴스가 액세스

DB 클러스터가 액세스 시 사용할 EC2 인스턴스와 다른 VPC에 있는 경우 DB 클러스터에 액세스하기 위해 VPC 피어링을 사용할 수 있습니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.

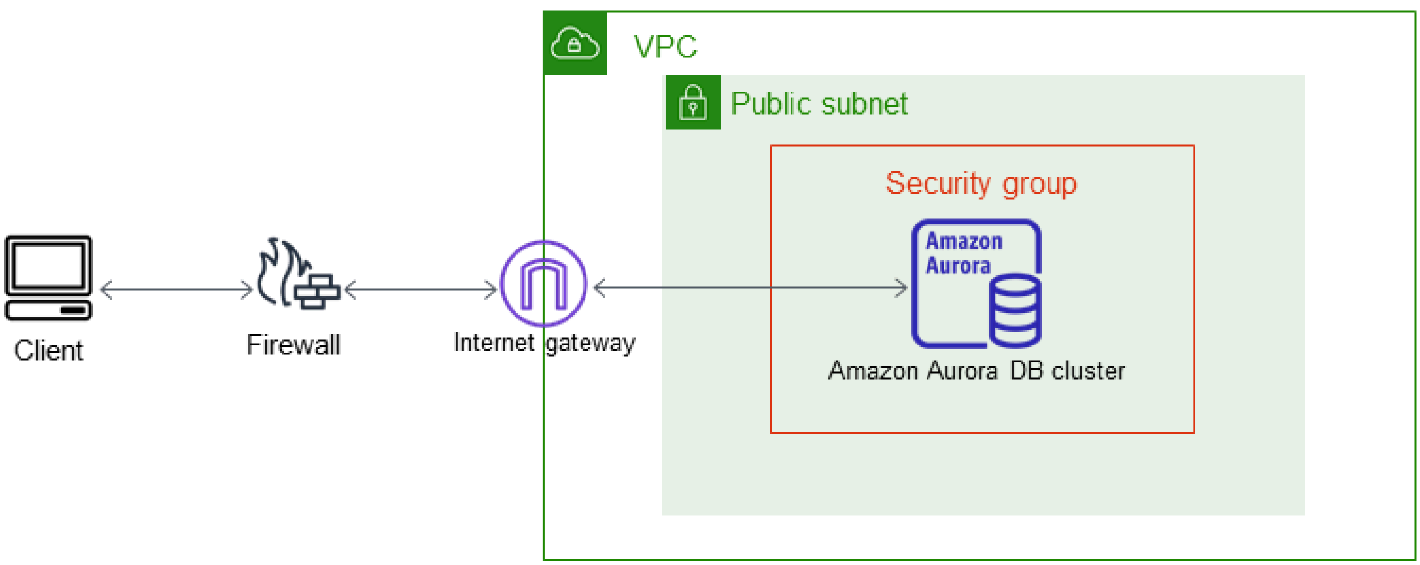


VPC 피어링 연결은 프라이빗 IP 주소를 사용하여 두 VPC 간에 트래픽을 라우팅할 수 있도록 하기 위한 두 VPC 사이의 네트워킹 연결입니다. 동일한 네트워크에 속하는 경우와 같이 VPC의 리소스가 서로 통신할 수 있습니다. 자체 VPC 간, 다른 AWS 계정에서 VPC를 사용하여 또는 다른 AWS 리전에서 VPC를 사용하여 VPC 피어링 연결을 만들 수 있습니다. VPC 피어링에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC 피어링을 참조하세요.

클라이언트 애플리케이션이 인터넷을 통해 VPC에 있는 DB 클러스터에 액세스

인터넷을 통해 클라이언트 애플리케이션에서 VPC에 있는 DB 클러스터에 액세스하려면, 단일 퍼블릭 서브넷이 있는 VPC와 인터넷을 통한 통신을 지원하는 인터넷 게이트웨이를 구성합니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.



다음 구성을 권장합니다.

- 크기가 /16인 VPC(예: CIDR: 10.0.0.0/16). 이 크기는 65,536개의 프라이빗 IP 주소를 제공합니다.
- 크기가 /24인 서브넷(예: CIDR: 10.0.0.0/24). 이 크기는 256개의 프라이빗 IP 주소를 제공합니다.
- Amazon Aurora DB 클러스터는 VPC 및 서브넷과 연결됩니다. Amazon RDS는 서브넷 내의 IP 주소를 DB 클러스터에 할당합니다.
- VPC를 인터넷 및 다른 AWS 제품과 연결하는 인터넷 게이트웨이.
- DB 클러스터와 연결된 보안 그룹입니다. 보안 그룹의 인바운드 규칙은 클라이언트 애플리케이션이 사용자의 DB 클러스터에 액세스할 수 있도록 해줍니다.

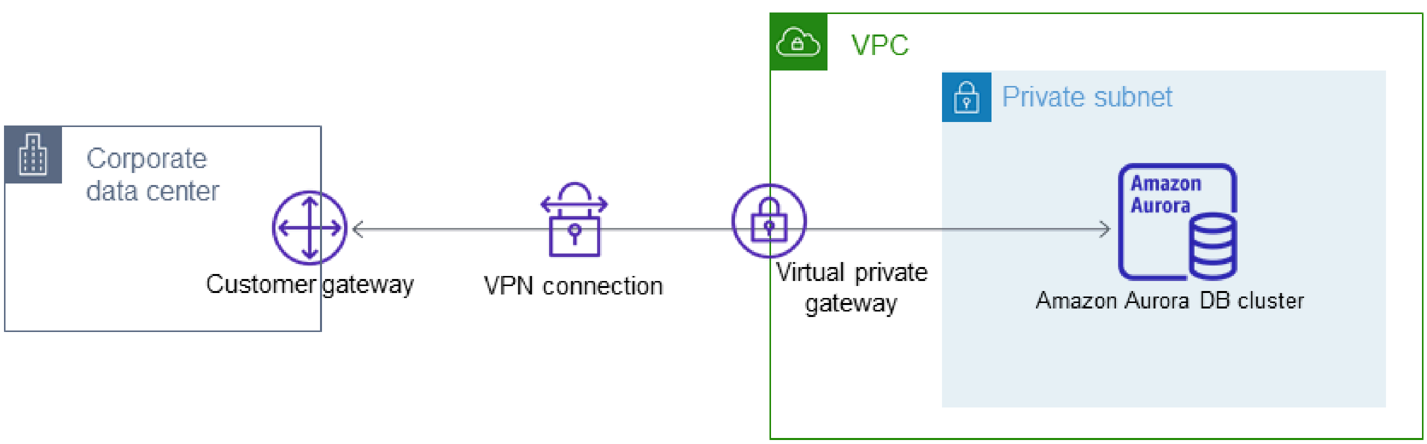
VPC에서 DB 클러스터 생성에 대한 자세한 내용은 VPC에 DB 클러스터 만들기 단원을 참조하세요.

프라이빗 네트워크에서 액세스하는 VPC의 DB 클러스터

DB 클러스터에 공개적으로 액세스할 수 없는 경우 프라이빗 네트워크에서 액세스할 수 있는 옵션은 다음과 같습니다.

- AWS Site-to-Site VPN 연결. 자세한 내용은 AWS Site-to-Site VPN란 무엇입니까?를 참조하십시오.
- AWS Direct Connect 연결. 자세한 내용은 AWS Direct Connect란 무엇인가요?를 참조하세요.
- AWS Client VPN 연결. 자세한 내용은 AWS Client VPN란 무엇입니까?를 참조하세요.

다음 다이어그램은 AWS Site-to-Site VPN 연결 시나리오를 보여줍니다.

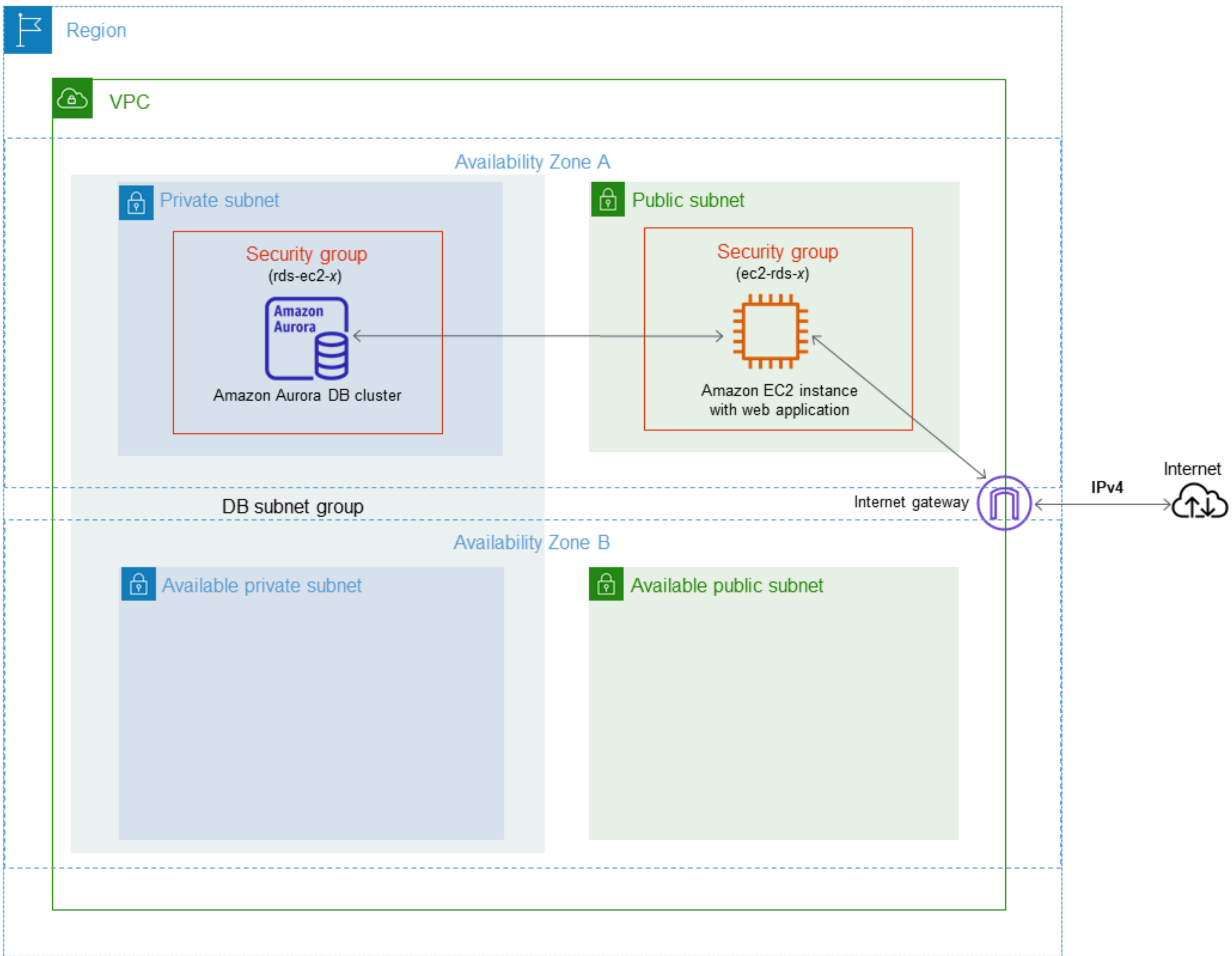


자세한 내용은 인터네트워크 트래픽 개인 정보 단원을 참조하십시오.

자습서: DB 클러스터에 사용할 Amazon VPC 생성(IPv4 전용)

일반적인 시나리오에는 Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)의 DB 클러스터가 포함됩니다. 이 VPC는 동일한 VPC에서 실행 중인 웹 서버와 데이터를 공유합니다. 이 자습에서는 이 시나리오를 위한 VPC를 생성합니다.

다음 다이어그램은 이 시나리오를 보여 줍니다. 다른 시나리오에 대한 자세한 내용은 VPC에서 DB 클러스터에 액세스하는 시나리오을(를) 참조하십시오.



퍼블릭 인터넷이 아닌 웹 서버에서만 DB 클러스터를 사용할 수 있어야 합니다. 따라서 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하여 VPC를 생성합니다. 퍼블릭 서브넷에서 웹 서버를 호스팅하므로 웹 서버에서 퍼블릭 인터넷에 액세스할 수 있습니다. DB 클러스터는 프라이빗 서브넷에서 호스팅됩니다. Amazon EC2 인스턴스는 동일한 VPC 내에서 호스팅되므로 DB 클러스터에 연결할 수 있습니다. 하지만 퍼블릭 인터넷에서는 DB 클러스터를 사용할 수 없으므로 보안이 강화됩니다.

이 자습서에서는 별도의 가용 영역에서 추가 퍼블릭 및 프라이빗 서브넷을 구성합니다. 이러한 서브넷은 자습서에서 사용하지 않습니다. RDS DB 서브넷 그룹은 최소 2개의 가용 영역에 한 개 이상의 서브넷이 필요합니다. 추가 서브넷을 사용하면 둘 이상의 Aurora DB 인스턴스를 쉽게 구성할 수 있습니다.

이 자습서에서는 Amazon Aurora DB 클러스터용 VPC 구성에 대해 설명합니다. 이 VPC 시나리오에 대한 웹 서버를 생성하는 방법을 보여 주는 자습서는 자습서: 웹 서버 및 Amazon Aurora DB 클러스터 생성 단원을 참조하세요. Amazon VPC 대한 자세한 내용은 Amazon VPC 시작 안내서 및 Amazon VPC 사용 설명서를 참조하세요.

> **Tip**
>
> Amazon EC2 인스턴스와 DB 간에 네트워크 연결을 설정할 수 있습니다.클러스터DB를 생성할 때 자동으로클러스터. 네트워크 구성은 이 자습서에서 설명한 것과 비슷합니다. 자세한 내용은 EC2 인스턴스와의 자동 네트워크 연결 구성. 섹션을 참조하세요.

프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성

다음은 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하는 VPC를 생성하는 절차입니다.

VPC 및 서브넷을 생성하는 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 VPC를 생성할 리전을 선택합니다. 이 예에서는 미국 서부(오레곤) 리전을 사용합니다.
3. 왼쪽 상단 모서리에서 VPC 대시보드를 선택합니다. VPC 생성을 시작하려면 VPC 생성을 선택합니다.
4. VPC 설정의 생성할 리소스에서 VPC 등을 선택합니다.
5. VPC 설정을 위해 다음 값을 설정합니다.

   - 네임 태그 자동 생성 - **`tutorial`**
   - IPv4 CIDR block: - **`10.0.0.0/16`**
   - IPv6 CIDR 블록 - No IPv6 CIDR Block(IPv6 CIDR 블록 없음)
   - 테넌시 - 기본값
   - 가용 영역(AZ)의 수 - 2
   - Customize AZs(AZ 사용자 지정) - 기본값을 유지합니다.
   - 퍼블릭 서브넷 수 – 2

- 프라이빗 서브넷 수 – 2
- Customize subnets CIDR blocks(서브넷 CIDR 블록 사용자 지정) - 기본값을 유지합니다.
- NAT 게이트웨이($) – 없음
- VPC 엔드포인트 – 없음
- DNS options(DNS 옵션) - 기본값을 유지합니다.

6. VPC 생성을 선택합니다.

퍼블릭 웹 서버에 대해 VPC 보안 그룹 생성

이제 퍼블릭 액세스를 위한 보안 그룹을 생성합니다. VPC의 퍼블릭 EC2 인스턴스에 연결하려면 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다. 이를 통해 인터넷에서 트래픽을 연결할 수 있습니다.

VPC 보안 그룹의 생성 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름: **`tutorial-securitygroup`**
   - 설명: **`Tutorial Security Group`**
   - VPC: 이전에 생성한 VPC를 선택합니다(예: vpc-***`identifier`***(tutorial-vpc)).
4. 인바운드 규칙을 보안 그룹에 추가합니다.

   a. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 지정합니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다. IP 주소의 예는 `203.0.113.25/32`입니다.

      대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 찾습니다.

      > **Warning**
      >
      > SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH

를 사용하여 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

b. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

c. 새로운 인바운드 규칙으로 다음 값을 설정하여 SSH에서 Amazon EC2 인스턴스에 액세스하도록 허용합니다. 이렇게 하면 Amazon EC2 인스턴스에 연결하여 웹 서버 및 다른 유틸리티를 설치할 수 있습니다. 또한 EC2 인스턴스에 연결하여 웹 서버의 콘텐츠를 업로드할 수도 있습니다.

- Type: **SSH**
- 소스: a단계의 IP 주소 또는 범위(예: **203.0.113.25/32**)

d. 규칙 추가를 선택합니다.

e. 새 인바운드 규칙에 다음 값을 설정하여 웹 서버에 대한 HTTP 액세스를 허용합니다.

- 유형: **HTTP**
- 소스: **0.0.0.0/0**

5. 보안 그룹을 생성하려면 보안 그룹 생성(Create security group)을 선택합니다.

   이 자습서에서 나중에 필요하므로 보안 그룹 ID를 적어 둡니다.

프라이빗 DB 클러스터에 대한 VPC 보안 그룹 생성

DB 클러스터를 프라이빗으로 유지하려면 프라이빗 액세스를 위한 보조 보안 그룹을 생성합니다. VPC의 프라이빗 DB 클러스터에 연결하려면 웹 서버로부터의 트래픽만 허용하는 인바운드 규칙을 VPC 보안 그룹에 추가해야 합니다.

VPC 보안 그룹의 생성 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름: **tutorial-db-securitygroup**
   - 설명: **Tutorial DB Instance Security Group**
   - VPC: 이전에 생성한 VPC를 선택합니다(예: vpc-***identifier***(tutorial-vpc)).
4. 인바운드 규칙을 보안 그룹에 추가합니다.

a. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.
b. 새로운 인바운드 규칙으로 다음 값을 설정하여 Amazon EC2 인스턴스의 포트 3306에서 MySQL 트래픽을 허용합니다. 이렇게 하면 웹 서버에서 DB 클러스터에 연결할 수 있습니다. 그러면 웹 애플리케이션의 데이터를 데이터베이스에 저장하고 검색할 수 있습니다.

- 유형: **`MySQL/Aurora`**
- 소스: 이 자습서에서 이전에 생성한 tutorial-securitygroup 보안 그룹의 식별자(예: sg-9edd5cfb)입니다.

5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

DB 서브넷 그룹 만들기

DB 서브넷 그룹은 사용자가 VPC에서 만든 다음 DB 클러스터에 대해 지정하는 서브넷의 모음입니다. DB 서브넷 그룹에서 DB 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다.

DB 서브넷 그룹을 만들려면

1. VPC에서 데이터베이스의 프라이빗 서브넷을 식별합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 서브넷을 선택합니다.
   c. tutorial-subnet-private1-us-west-2a 및 tutorial-subnet-private2-us-west-2b라는 서브넷의 서브넷 ID를 기록해 둡니다.

   DB 서브넷 그룹을 생성할 때 서브넷 ID가 필요합니다.

2. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

   Amazon VPC 콘솔이 아닌 Amazon RDS 콘솔에 연결해야 합니다.

3. 탐색 창에서 [Subnet groups]를 선택합니다.
4. DB 서브넷 그룹 생성을 선택합니다.
5. DB 서브넷 그룹 생성 페이지에서 서브넷 그룹 세부 정보에 이들 값을 설정합니다.

   - 이름: **`tutorial-db-subnet-group`**
   - 설명: **`Tutorial DB Subnet Group`**
   - VPC: tutorial-vpc(vpc-***`identifier`***)

6. 서브넷 추가 섹션에서 가용 영역 및 서브넷을 선택합니다.

이 자습서에서는 us-west-2a와 us-west-2b를 가용 영역으로 선택합니다. 서브넷에서 이전 단계에서 식별한 프라이빗 서브넷을 선택합니다.

7. 생성(Create)을 선택합니다.

   새 DB 서브넷 그룹은 RDS 콘솔의 DB 서브넷 그룹 목록에 나타납니다. DB 서브넷 그룹을 선택하여 창 하단의 세부 정보 창에서 세부 정보를 볼 수 있습니다. 이러한 세부 정보에는 그룹과 연결된 모든 서브넷이 포함됩니다.

> **Note**
>
> 자습서: 웹 서버 및 Amazon Aurora DB 클러스터 생성을 완료하기 위해 이 VPC를 생성한 경우 Amazon Aurora DB 클러스터 생성의 지침에 따라 DB 클러스터를 생성합니다.

VPC 삭제

이 자습서에 대한 VPC 및 기타 리소스를 생성한 후, 더 이상 필요하지 않은 경우 삭제할 수 있습니다.

> **Note**
>
> 이 자습서를 위해 생성한 VPC에 리소스를 추가한 경우 VPC를 삭제하기 전에 이러한 리소스를 삭제해야 할 수 있습니다. 예를 들어 이러한 리소스에는 Amazon EC2 인스턴스 또는 Amazon RDS DB 클러스터가 포함될 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC의 삭제를 참조하세요.

VPC 및 관련 리소스 삭제하기

1. DB 서브넷 그룹을 삭제합니다.

   a. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
   b. 탐색 창에서 서브넷 그룹을 선택합니다.
   c. 삭제하려는 DB 서브넷 그룹을 선택합니다(예: tutorial-db-subnet-group).
   d. 삭제를 선택한 다음 확인 창에서 삭제를 선택합니다.

2. VPC ID를 기록합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.

   b. VPC 대시보드를 선택한 다음 VPC를 선택합니다.
   c. 목록에서 생성한 VPC를 식별합니다(예:tutorial-vpc).
   d. 생성한 VPC의 VPC ID를 기록합니다. 이후 단계에서 해당 VPC ID가 필요합니다.

3. 보안 그룹을 삭제합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 보안 그룹을 선택합니다.
   c. Amazon RDS DB 인스턴스에 대한 보안 그룹을 선택합니다(예:tutorial-db-securitygroup).
   d. 작업(Actions)에서 보안 그룹 삭제를 선택한 다음 확인 페이지에서 삭제(Delete)를 선택합니다.
   e. 보안 그룹 페이지에서 Amazon EC2 인스턴스에 대한 보안 그룹을 선택합니다(예:tutorial-securitygroup).
   f. 작업(Actions)에서 보안 그룹 삭제를 선택한 다음 확인 페이지에서 삭제(Delete)를 선택합니다.

4. VPC를 삭제합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 VPC를 선택합니다.
   c. 삭제하려는 VPC 선택합니다(예:tutorial-vpc).
   d. 작업에서 VPC 삭제를 선택합니다.

      확인 페이지에는 VPC와 연결된 서브넷을 포함하여 삭제될 VPC와 연결된 다른 리소스가 표시됩니다.
   e. 확인 페이지에서 **`delete`**를 입력하고 삭제를 선택합니다.

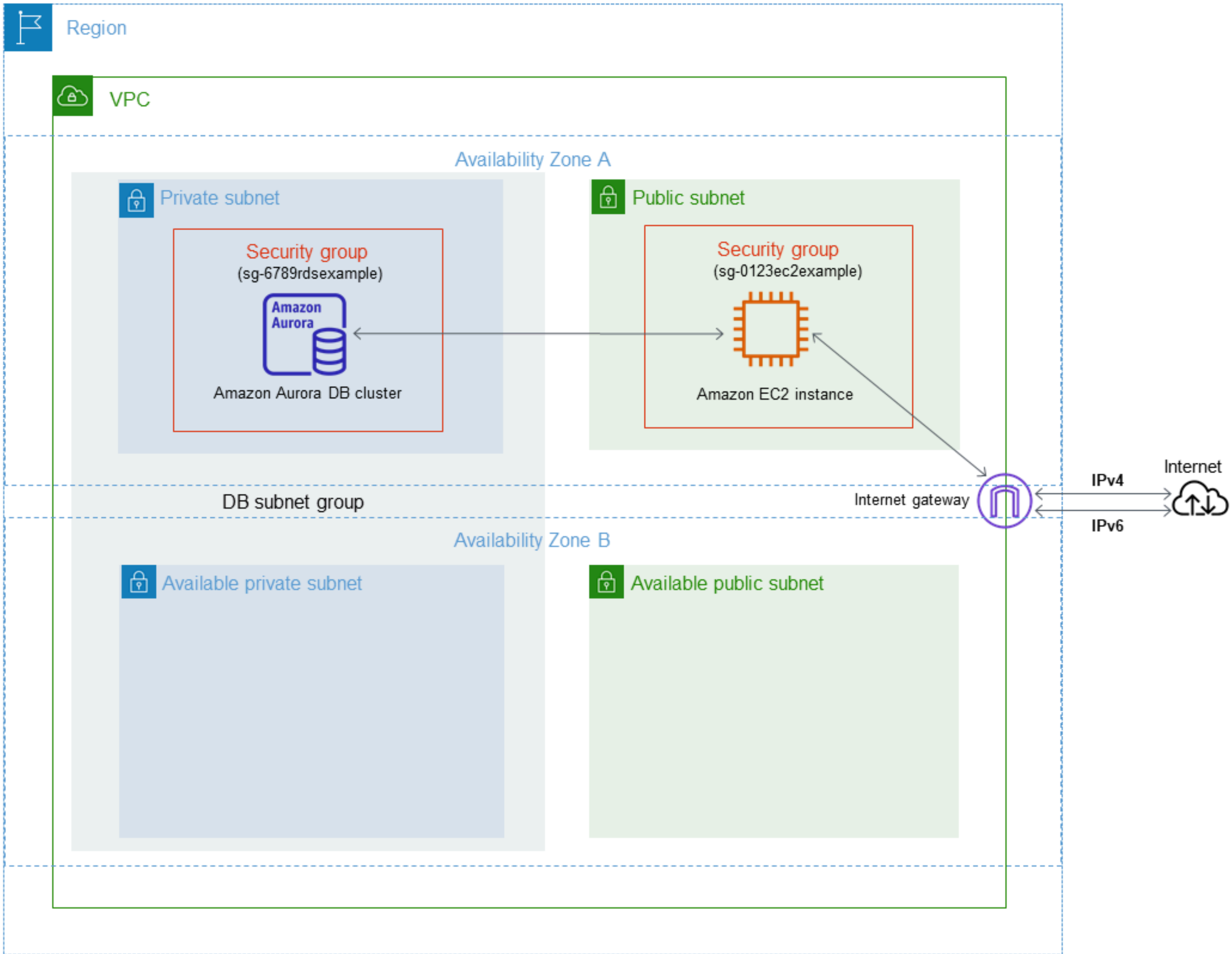
자습서: DB 클러스터(듀얼 스택 모드)에 사용할 VPC 생성

일반적인 시나리오에는 Amazon VPC 서비스를 기반으로 하는 Virtual Private Cloud(VPC)의 DB 클러스터가 포함됩니다. 이 VPC는 동일한 VPC에서 실행 중인 퍼블릭 Amazon EC2 인스턴스와 데이터를 공유합니다.

이 자습서에서는 듀얼 스택 모드에서 실행되는 데이터베이스와 함께 작동하는 이 시나리오의 VPC를 생성합니다. IPv6 주소 지정 프로토콜을 통한 연결을 가능하게 하는 듀얼 스택 모드입니다. IP 주소에 대한 자세한 내용은 Amazon Aurora IP 주소 지정 단원을 참조하십시오.

이중 스택 네트워크 클러스터는 대부분의 리전에서 지원됩니다. 자세한 정보는 듀얼 스택 네트워크 DB 클러스터의 가용성 섹션을 참조하세요. 이중 스택 모드의 제한 사항을 보려면 듀얼 스택 네트워크 DB 클러스터에 대한 제한 사항을 참조하세요.

다음 다이어그램은 이 시나리오를 보여 줍니다.



다른 시나리오에 대한 자세한 내용은 VPC에서 DB 클러스터에 액세스하는 시나리오을(를) 참조하세요.

퍼블릭 인터넷이 아닌 Amazon EC2 인스턴스에서만 DB 클러스터를 사용할 수 있어야 합니다. 따라서 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하여 VPC를 생성합니다. 퍼블릭 서브넷에서 Amazon EC2 인스턴스를 호스팅하므로 Amazon EC2 인스턴스에서 퍼블릭 인터넷에 액세스할 수 있습니다. DB 클러스터는 프라이빗 서브넷에서 호스팅됩니다. Amazon EC2 인스턴스는 동일한 VPC 내에서 호스팅되므로 DB 클러스터에 연결할 수 있습니다. 하지만 퍼블릭 인터넷에서는 DB 클러스터를 사용할 수 없으므로 보안이 강화됩니다.

이 자습서에서는 별도의 가용 영역에서 추가 퍼블릭 및 프라이빗 서브넷을 구성합니다. 이러한 서브넷은 자습서에서 사용하지 않습니다. RDS DB 서브넷 그룹은 최소 2개의 가용 영역에 한 개 이상의 서브넷이 필요합니다. 추가 서브넷을 사용하면 둘 이상의 Aurora DB 인스턴스를 쉽게 구성할 수 있습니다.

듀얼 스택 모드를 사용하는 DB 클러스터를 생성하려면 네트워크 유형 설정에 듀얼 스택 모드를 지정하세요. 동일한 설정으로 DB 클러스터를 수정할 수도 있습니다. DB 클러스터 생성에 대한 자세한 내용은 Amazon Aurora DB 클러스터 생성 단원을 참조하세요. DB 클러스터 수정에 대한 자세한 내용은 Amazon Aurora DB 클러스터 수정 단원을 참조하세요.

이 자습서에서는 Amazon Aurora DB 클러스터용 VPC 구성에 대해 설명합니다. Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성

다음은 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하는 VPC를 생성하는 절차입니다.

VPC 및 서브넷을 생성하는 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. AWS Management Console의 오른쪽 상단에서 VPC를 생성할 리전을 선택합니다. 이 예에서는 미국 동부(오하이오) 리전을 사용합니다.
3. 왼쪽 상단 모서리에서 VPC 대시보드를 선택합니다. VPC 생성을 시작하려면 VPC 생성을 선택합니다.
4. VPC 설정의 생성할 리소스에서 VPC 등을 선택합니다.
5. 남은 VPC 설정에서는 다음 값들을 설정합니다.

   - 네임 태그 자동 생성 - **`tutorial-dual-stack`**
   - IPv4 CIDR block: - **`10.0.0.0/16`**
   - IPv6 CIDR 블록(IPv6 CIDR block) - Amazon 제공 IPv6 CIDR 블록(Amazon-provided IPv6 CIDR block)
   - 테넌시 - 기본값
   - 가용 영역(AZ)의 수 - 2
   - Customize AZs(AZ 사용자 지정) - 기본값을 유지합니다.
   - 퍼블릭 서브넷 수 – 2
   - 프라이빗 서브넷 수 – 2
   - Customize subnets CIDR blocks(서브넷 CIDR 블록 사용자 지정) - 기본값을 유지합니다.
   - NAT 게이트웨이($) – 없음
   - 외부 전용 인터넷 게이트웨이 - 아니요
   - VPC 엔드포인트 – 없음

- DNS options(DNS 옵션) - 기본값을 유지합니다.

> **Note**
>
> Amazon RDS에서는 다중 AZ DB 인스턴스 배포를 지원하려면 서로 다른 두 가용 영역에 있는 Amazon RDS가 필요합니다. 이 자습서에서는 단일 AZ 배포를 생성하지만 요구 사항에 따라 향후 다중 AZ DB 인스턴스 배포로 쉽게 전환할 수 있습니다.

6. VPC 생성을 선택합니다.

퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성

이제 퍼블릭 액세스를 위한 보안 그룹을 생성합니다. VPC의 퍼블릭 EC2 인스턴스에 연결하려면 인터넷으로부터의 트래픽 연결을 허용하는 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다.

VPC 보안 그룹의 생성 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름: **`tutorial-dual-stack-securitygroup`**
   - 설명: **`Tutorial Dual-Stack Security Group`**
   - VPC: 이전에 생성한 VPC 선택(예: vpc-*###*(tutorial-dual-stack-vpc)
4. 인바운드 규칙을 보안 그룹에 추가합니다.
   a. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 지정합니다.

      IPv4(인터넷 프로토콜 버전 4) 주소의 예는 `203.0.113.25/32`입니다. IPv6(인터넷 프로토콜 버전 6) 주소 범위의 예는 `2001:db8:1234:1a00::/64`입니다.

      대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 찾습니다.

> **Warning**
>
> IPv4에 `0.0.0.0/0`, IPv6에 `::0`을 사용하면 SSH를 통해 모든 IP 주소가 퍼블릭 인스턴스에 액세스하도록 설정할 수 있습니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 특정 IP 주소나 주소 범위만 인스턴스에 액세스하도록 허용하세요.

b. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

c. 새로운 인바운드 규칙으로 다음 값을 설정하여 SSH(Secure Shell)에서 Amazon EC2 인스턴스에 액세스하도록 허용합니다. 이렇게 하면 EC2 인스턴스에 연결하여 SQL 클라이언트 및 다른 애플리케이션을 설치할 수 있습니다. EC2 인스턴스에 액세스할 수 있도록 IP 주소를 지정합니다.

- 유형: **`SSH`**
- 소스(Source): a단계의 IP 주소 또는 범위입니다. IPv4 IP 주소의 예는 **`203.0.113.25/32`**입니다. IPv6 IP 주소의 예는 **`2001:DB8::/32`**입니다.

5. 보안 그룹을 생성하려면 보안 그룹 생성(Create security group)을 선택합니다.

   이 자습서에서 나중에 필요하므로 보안 그룹 ID를 적어 둡니다.

프라이빗 DB 클러스터에 대한 VPC 보안 그룹 생성

DB 클러스터를 프라이빗으로 유지하려면 프라이빗 액세스를 위한 보조 보안 그룹을 생성합니다. VPC의 프라이빗 DB 클러스터에 연결하려면 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다. 이 규칙은 Amazon EC2 인스턴스로부터의 트래픽만 허용합니다.

VPC 보안 그룹의 생성 방법

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름: **`tutorial-dual-stack-db-securitygroup`**
   - 설명: **`Tutorial Dual-Stack DB Instance Security Group`**
   - VPC: 이전에 생성한 VPC 선택(예: vpc-***###***(tutorial-dual-stack-vpc)
4. 인바운드 규칙을 보안 그룹에 추가합니다.

a. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.
b. 새로운 인바운드 규칙으로 다음 값을 설정하여 Amazon EC2 인스턴스의 포트 3306에서 MySQL 트래픽을 허용합니다. 이렇게 하면 EC2 인스턴스에서 DB 클러스터에 연결할 수 있습니다. 이는 EC2 인스턴스의 데이터를 데이터베이스에 전송할 수 있음을 의미합니다.

   - 유형: MySQL/Aurora
   - 소스: 이 자습서에서 이전에 생성한 tutorial-dual-stack-securitygroup 보안 그룹의 식별자 (예: sg-9edd5cfb)입니다.

5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

DB 서브넷 그룹 만들기

DB 서브넷 그룹은 사용자가 VPC에서 만든 다음 DB 클러스터에 대해 지정하는 서브넷의 모음입니다. DB 서브넷 그룹을 사용하면 DB 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. DUAL과 호환 가능한 DB 서브넷 그룹을 생성하려면 모든 서브넷이 DUAL과 호환 가능해야 합니다. DUAL과 호환 가능하려면 서브넷에 IPv6 CIDR이 연결되어 있어야 합니다.

DB 서브넷 그룹을 만들려면

1. VPC에서 데이터베이스의 프라이빗 서브넷을 식별합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 서브넷을 선택합니다.
   c. tutorial-dual-stack-subnet-private1-us-west-2a 및 tutorial-dual-stack-subnet-private2-us-west-2b 서브넷의 서브넷 ID를 기록해 둡니다.

      DB 서브넷 그룹을 생성할 때 서브넷 ID가 필요합니다.

2. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.

   Amazon VPC 콘솔이 아닌 Amazon RDS 콘솔에 연결해야 합니다.

3. 탐색 창에서 [Subnet groups]를 선택합니다.
4. DB 서브넷 그룹 생성을 선택합니다.
5. DB 서브넷 그룹 생성 페이지에서 서브넷 그룹 세부 정보에 이들 값을 설정합니다.

   - 이름: **`tutorial-dual-stack-db-subnet-group`**
   - 설명: **`Tutorial Dual-Stack DB Subnet Group`**

- VPC: tutorial-dual-stack-vpc(vpc-***identifier***)

6. 서브넷 추가(Add subnets) 섹션에서 가용 영역(Availability Zones) 및 서브넷(Subnets)의 값을 선택합니다.

   이 자습서에서는 us-east-2a와 us-east-2b를 가용 영역으로 선택합니다. 서브넷에서 이전 단계에서 식별한 프라이빗 서브넷을 선택합니다.

7. 생성(Create)을 선택합니다.

새 DB 서브넷 그룹은 RDS 콘솔의 DB 서브넷 그룹 목록에 나타납니다. DB 서브넷 그룹을 선택하여 세부 정보를 확인할 수 있습니다. 여기에는 지원되는 주소 지정 프로토콜과 그룹과 연결된 모든 서브넷 및 DB 서브넷 그룹에서 지원하는 네트워크 유형이 포함됩니다.

듀얼 스택 모드로 Amazon EC2 인스턴스 생성

Amazon EC2 인스턴스를 생성하려면 Amazon EC2 사용 설명서의 새 시작 인스턴스 마법사를 사용하여 인스턴스 시작의 지침을 따르세요.

인스턴스 세부 정보 구성 페이지에서 이러한 값을 설정하고 다른 값은 기본값으로 유지합니다.

- 네트워크 - 프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성에서 생성한 tutorial-dual-stack-vpc(vpc-*identifier*)와 같이 퍼블릭 및 프라이빗 서브넷이 있는 기존 VPC를 선택합니다.
- 서브넷 - 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성에서 생성한 subnet-***identifier*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a와 같은 기존 퍼블릭 서브넷을 선택합니다.
- 퍼블릭 IP 자동 할당 - 활성화를 선택합니다.
- Auto-assign IPv6 IP - 활성화를 선택합니다.
- 방화벽(보안 그룹) - 기존 보안 그룹 선택을 선택합니다.
- 일반 보안 그룹 - 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성에서 생성한 `tutorial-securitygroup`과 같이 기존 보안 그룹을 선택합니다. 선택한 보안 그룹에 SSH(Secure Shell) 및 HTTP 액세스에 대한 인바운드 규칙이 포함되어 있는지 확인합니다.

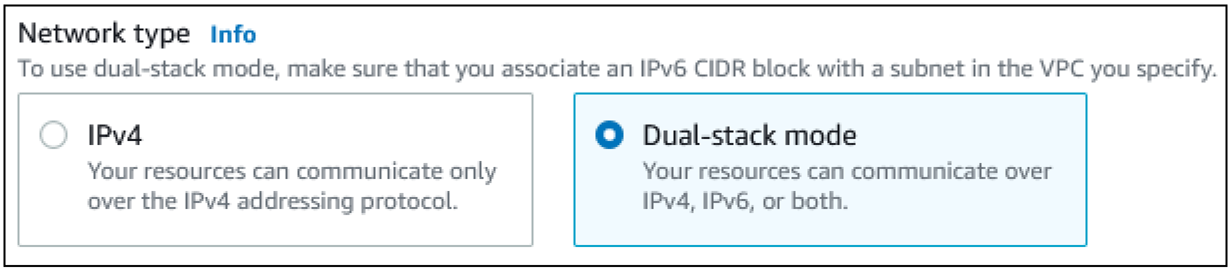
듀얼 스택 모드로 DB 클러스터 생성

이 단계에서는 듀얼 스택 모드로 실행되는 Amazon RDS DB 클러스터를 생성합니다.

DB 인스턴스를 생성하려면

1. https://console.aws.amazon.com/rds/에서 AWS Management Console에 로그인한 후 Amazon RDS 콘솔을 엽니다.
2. 콘솔의 오른쪽 상단에서 DB 인스턴스를 생성하려는 을 선택합니다. 이 예에서는 미국 동부(오하이오) 리전을 사용합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택합니다.
5. [데이터베이스 생성(Create database)] 페이지에서 다음과 같이 [표준 생성(Standard create)] 옵션이 선택되어 있는지 확인하고 [MySQL]을 선택합니다.
6. [연결(Connectivity)] 섹션에서 다음 값을 설정합니다.

    - 네트워크 유형(Network type) – 듀얼 스택 모드(Dual-stack mode) 선택

    - 가상 사설 클라우드(VPC) - 프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성에서 생성한 tutorial-dual-stack-vpc(vpc-*identifier*)와 같이 퍼블릭 및 프라이빗 서브넷이 있는 기존 VPC를 선택합니다.

      VPC는 서로 다른 가용 영역에 서브넷이 있어야 합니다.
    - 서브넷 그룹(Subnet Group) - DB 서브넷 그룹 만들기에서 생성한 tutorial-dual-stack-db-subnet-group과 같은 VPC용 DB 서브넷 그룹을 선택합니다.
    - 퍼블릭 액세스— 선택아니요.
    - VPC 보안 그룹 (방화벽)— 선택기존 항목 선택.
    - 기존 VPC 보안 그룹(Existing VPC security groups) – 프라이빗 DB 클러스터에 대한 VPC 보안 그룹 생성에서 생성한 tutorial-dual-stack-db-securitygroup과 같이 프라이빗 액세스에 맞게 구성된 기존 VPC 보안 그룹을 선택합니다.

      각각에 연결된 X를 선택해 기본 보안 그룹 같은 다른 보안 그룹을 제거합니다.
    - [Availability Zone]에서 [us-west-2a]를 선택합니다.

      AZ 간 트래픽을 피하려면 DB 인스턴스와 EC2 인스턴스가 동일한 가용 영역에 있어야 합니다.

7. 나머지 섹션에서 DB 클러스터 설정을 지정합니다. 각 설정에 대한 자세한 내용은 Aurora DB 클러스터 설정 단원을 참조하세요.

Amazon EC2 인스턴스 및 DB 인스턴스에 연결

Amazon EC2 인스턴스와 DB 인스턴스가 듀얼 스택 모드로 생성된 후에는 IPv6 프로토콜을 사용하여 각 인스턴스에 연결할 수 있습니다. IPv6 프로토콜을 사용하여 Amazon EC2 인스턴스에 연결하려면 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결에 있는 지침을 따르세요.

Amazon EC2 인스턴스에서 Aurora MySQL DB 클러스터에 연결하려면 Aurora MySQL DB 클러스터에 연결의 지침을 따르세요.

VPC 삭제

이 자습서에 대한 VPC 및 기타 리소스를 생성한 후, 더 이상 필요하지 않은 경우 삭제할 수 있습니다.

이 자습서를 위해 생성한 VPC에 리소스를 추가한 경우 VPC를 삭제하기 전에 이러한 리소스를 삭제해야 할 수 있습니다. 리소스의 예로는 Amazon EC2 인스턴스 또는 DB 클러스터가 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC의 삭제를 참조하세요.

VPC 및 관련 리소스 삭제하기

1. DB 서브넷 그룹을 삭제합니다.
   a. https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
   b. 탐색 창에서 서브넷 그룹을 선택합니다.
   c. 삭제할 DB 서브넷 그룹을 선택합니다(예: tutorial-db-subnet-group).
   d. 삭제를 선택한 다음 확인 창에서 삭제를 선택합니다.
2. VPC ID를 기록합니다.
   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 VPC를 선택합니다.
   c. 목록에서 생성한 VPC를 식별합니다(예:tutorial-dual-stack-vpc).
   d. 생성한 VPC의 VPC ID를 기록합니다. 이후 단계에서 해당 VPC ID가 필요합니다.
3. 보안 그룹을 삭제합니다.
   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 보안 그룹을 선택합니다.

c. Amazon RDS DB 인스턴스에 대한 보안 그룹을 선택합니다(예:utorial-dual-stack-db-securitygroup).
d. 작업(Actions)에서 보안 그룹 삭제(Delete security groups)를 선택한 다음 확인 페이지에서 삭제(Delete)를 선택합니다.
e. 보안 그룹(Security Groups) 페이지에서 Amazon EC2 인스턴스에 대한 보안 그룹을 선택합니다(예:tutorial-dual-stack-securitygroup).
f. 작업(Actions)에서 보안 그룹 삭제(Delete security groups)를 선택한 다음 확인 페이지에서 삭제(Delete)를 선택합니다.

4. NAT 게이트웨이를 삭제합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 NAT 게이트웨이를 선택합니다.
   c. 생성한 VPC의 NAT 게이트웨이를 선택합니다. VPC ID를 사용하여 올바른 NAT 게이트웨이를 식별합니다.
   d. 작업에서 NAT 게이트웨이 삭제(Delete NAT gateway)를 선택합니다.
   e. 확인 페이지에서 **`delete`**를 입력하고 삭제를 선택합니다.

5. VPC를 삭제합니다.

   a. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 VPC를 선택합니다.
   c. 삭제하려는 VPC를 선택합니다(예:tutorial-dual-stack-vpc).
   d. 작업에서 VPC 삭제를 선택합니다.

      확인 페이지에는 VPC와 연결된 서브넷을 포함하여 삭제될 VPC와 연결된 다른 리소스가 표시됩니다.
   e. 확인 페이지에서 **`delete`**를 입력하고 삭제를 선택합니다.

6. 탄력적 IP 주소를 릴리스합니다.

   a. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
   b. EC2 대시보드를 선택한 다음 탄력적 IP 주소를 선택합니다.
   c. 릴리스하려는 탄력적 IP 주소를 선택합니다.
   d. 작업(Actions)에서 탄력적 IP 주소 릴리스(Release Elastic IP addresses)를 선택합니다.
   e. 확인 페이지에서 릴리스를 선택합니다.

Amazon Aurora에 대한 할당량 및 제약 조건

다음에는 Amazon Aurora에 대한 리소스 할당량 및 명명 제약 조건에 대한 설명을 찾을 수 있습니다.

주제

Amazon Aurora의 할당량

각 AWS 계정에는 AWS 리전마다 생성할 수 있는 Amazon Aurora 리소스 수에 할당량이 있습니다. 리소스 할당량에 도달하면 해당 리소스 생성을 위한 추가 호출이 예외와 함께 실패합니다.

다음 표에는 AWS 리전별 리소스 및 그 할당량이 나열되어 있습니다.

이름	기본값	조정 가능	설명
DB 보안 그룹당 권한 부여	지원되는 각 리전: 20	아니요	DB 보안 그룹당 보안 그룹 인증 수
사용자 지정 엔진 버전	지원되는 각 리전: 40	예	현재 리전에서 이 계정에 허용된 커스텀 엔진 버전의 최대 개수
DB 클러스터 파라미터 그룹	지원되는 각 리전: 50	아니요	DB 클러스터 파라미터 그룹의 최대 개수
DB 클러스터	지원되는 각 리전: 40	예	현재 리전에서 이 계정에 허용된 클러스터의 최대 개수

이름	기본값	조정 가능	설명
DB 인스턴스	지원되는 각 리전: 40	예	현재 리전에서 이 계정에 허용된 DB 인스턴스의 최대 개수
DB 서브넷 그룹	지원되는 각 리전: 50	예	DB 서브넷 그룹의 최대 개수
데이터 API HTTP 요청 본문 크기	지원되는 각 리전: 4MB	아니요	HTTP 요청 본문에 허용된 최대 크기입니다.
데이터 API 최대 동시 클러스터-암호 페어 수	지원되는 각 리전: 30개	아니요	현재 AWS 리전의 이 계정에 대한 동시 데이터 API 요청에서 Aurora Serverless v1 DB 클러스터 및 암호의 최대 고유한 페어 개수입니다.
데이터 API 최대 동시 요청 수	지원되는 각 리전: 500	아니요	Aurora Serverless v1 DB 클러스터에 대해 동일한 암호를 사용하고 동시에 처리될 수 있는 최대 데이터 API 요청 개수입니다. 추가 요청은 대기열에 추가되고, 처리 중인 요청이 완료되면 처리됩니다.
데이터 API 최대 결과 집합 크기	지원되는 각 리전: 1MB	아니요	데이터 API에서 반환할 수 있는 데이터베이스 결과 세트의 최대 크기입니다.

이름	기본값	조정 가능	설명
JSON 응답 문자열의 데이터 API 최대 크기	지원되는 각 리전: 10MB	아니요	RDS 데이터 API에서 반환하는 단순화된 JSON 응답 문자열의 최대 크기입니다.
초당 데이터 API 요청 수	지원되는 각 리전: 초당 1,000개	아니요	현재 AWS 리전에서 이 계정에 허용되는 초당 데이터 API에 대한 최대 요청 수입니다. 이 할당량은 Amazon Aurora Serverless v1 클러스터에만 적용됩니다.
이벤트 구독	지원되는 각 리전: 20	예	이벤트 구독의 최대 개수
DB 클러스터당 IAM 역할	지원되는 각 리전: 5	예	DB 클러스터와 연결된 IAM 역할의 최대 개수
DB 인스턴스당 IAM 역할	지원되는 각 리전: 5	예	DB 인스턴스와 연결된 IAM 역할의 최대 개수
통합	지원되는 각 리전: 100	아니요	현재 AWS 리전에서 이 계정에 허용된 최대 통합 수
수동 DB 클러스터 스냅샷	지원되는 각 리전: 100	예	수동 DB 클러스터 스냅샷의 최대 수
수동 DB 인스턴스 스냅샷 수	지원되는 각 리전: 100	예	수동 DB 인스턴스 스냅샷의 최대 수
옵션 그룹 수	지원되는 각 리전: 20	예	옵션 그룹의 최대 개수

이름	기본값	조정 가능	설명
파라미터 그룹	지원되는 각 리전: 50	예	파라미터 그룹의 최대 개수
프록시	지원되는 각 리전: 20	예	현재 AWS 리전에서 이 계정에 허용된 프록시의 최대 개수
기본당 읽기 전용 복제본	지원되는 각 리전: 15개	예	프라이머리 DB 인스턴스당 읽기 전용 복제본의 최대 개수 Amazon Aurora에서는 이 할당량을 조정할 수 없습니다.
예약 DB 인스턴스	지원되는 각 리전: 40	예	현재 AWS 리전에서 이 계정에 허용된 예약 DB 인스턴스의 최대 개수
보안 그룹당 규칙	지원되는 각 리전: 20	아니요	DB 보안 그룹당 규칙의 최대 개수
보안 그룹	지원되는 각 리전: 25	예	DB 보안 그룹의 최대 개수
보안 그룹(VPC)	지원되는 각 리전: 5	아니요	Amazon VPC당 DB 보안 그룹의 최대 개수
DB 서브넷 그룹당 서브넷 수	지원되는 각 리전: 20	아니요	DB 서브넷 그룹당 서브넷의 최대 개수

이름	기본값	조정 가능	설명
리소스당 태그	지원되는 각 리전: 50	아니요	Amazon RDS 리소스당 태그의 최대 개수
모든 DB 인스턴스의 총 스토리지	지원되는 각 리전: 100,000기가바이트	예	함께 추가된 모든 Amazon RDS DB 인스턴스에 대한 EBS 볼륨의 최대 총 스토리지(GB)입니다. 이 할당량은 각 DB 클러스터에 대해 최대 클러스터 볼륨이 128TiB인 Amazon Aurora에는 적용되지 않습니다.

> **Note**
>
> 기본적으로 최대 총 40개의 DB 인스턴스를 실행할 수 있습니다. RDS DB 인스턴스, Aurora DB 인스턴스, Amazon Neptune 인스턴스 및 Amazon DocumentDB 인스턴스가 이 할당량에 적용됩니다.
>
> 애플리케이션에 DB 인스턴스가 더 필요한 경우 Service Quotas 콘솔을 열어 추가 DB 인스턴스를 요청할 수 있습니다. 탐색 창에서 AWS 서비스를 선택합니다. Amazon Relational Database Service(Amazon RDS)를 선택하고, 할당량을 선택한 다음, 지침에 따라 할당량 증가를 요청합니다. 자세한 내용은 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하십시오.
>
> AWS Backup에서 관리하는 백업은 수동 DB 클러스터 스냅샷으로 간주되지만 수동 클러스터 스냅샷 할당량에 포함되지는 않습니다. AWS Backup에 대한 자세한 내용은 AWS Backup 개발자 가이드를 참조하십시오.

RDS API 작업을 사용하고 초당 호출 수의 기본 할당량을 초과하는 경우, Amazon RDS API는 다음과 같이 오류를 생성합니다.

ClientError: *API_name* 작업을 호출하는 동안 오류 발생(ThrottlingException): 속도 초과됨.

이 경우 초당 호출 수를 줄입니다. 할당량은 대부분의 사용 사례를 다루기 위한 것입니다. 더 높은 할당량이 필요한 경우 다음 옵션 중 하나를 사용하여 할당량 증가를 요청할 수 있습니다.

- 콘솔에서 Service Quotas 콘솔을 엽니다.
- AWS CLI에서 request-service-quota-increase AWS CLI 명령을 사용합니다.

자세한 내용은 Service Quotas 사용 설명서를 참조하세요.

Amazon Aurora의 명명 제약 조건

다음은 Amazon Aurora의 명명 제약 조건을 설명한 것입니다.

- DB 클러스터 식별자:
  - 1–63자의 영숫자 또는 하이픈으로 구성되어야 합니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
  - 각 AWS 리전별로 AWS 계정 1개의 모든 DB 인스턴스는 고유해야 합니다.
- 초기 데이터베이스 이름 – Aurora MySQL과 Aurora PostgreSQL 간에 데이터베이스 이름 제약 조건이 다릅니다. 자세한 내용은 각 DB 클러스터를 생성할 때 사용 가능한 설정을 참조하세요.
- 마스터 사용자 이름 - 마스터 사용자 이름 제약 조건은 각 데이터베이스 엔진에 따라 다릅니다. 자세한 내용은 DB 클러스터를 생성할 때 사용 가능한 설정을 참조하세요.
- 마스터 암호:
  - 마스터 데이터베이스 사용자의 암호에는 /, ' , ", @ 또는 공백을 제외한 모든 인쇄 가능한 ASCII 문자가 포함될 수 있습니다.
  - 암호는 DB 엔진에 따라 다음과 같은 수의 인쇄 가능한 ASCII 문자를 포함할 수 있습니다.
    - Aurora MySQL: 8–41
    - Aurora PostgreSQL: 8–99
- DB 파라미터 그룹:
  - 1–255자의 영숫자로 구성되어야 합니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈은 허용되지만 이름은 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
- DB 서브넷 그룹:
  - 1–255자로 구성되어야 합니다.

- 영숫자, 스페이스, 하이픈, 밑줄, 마침표를 사용할 수 있습니다.

Amazon Aurora 크기 제한

스토리지 크기 제한

Aurora 클러스터 볼륨은 다음 엔진 버전의 경우 최대 128테비바이트(TiB) 크기로 증가할 수 있습니다.

- 사용 가능한 Aurora MySQL 버전 3 버전, Aurora MySQL 버전 2, 버진 2.09 이상
- 사용 가능한 모든 Aurora PostgreSQL 버전

더 낮은 엔진 버전의 경우, Aurora 클러스터 볼륨의 최대 크기는 64TiB입니다. 자세한 내용은 Aurora 스토리지 크기가 자동으로 조정되는 방법 단원을 참조하십시오.

남은 스토리지 공간을 모니터링하려는 경우 `AuroraVolumeBytesLeftTotal` 지표를 사용할 수 있습니다. 자세한 내용은 Amazon Aurora에 대한 클러스터 수준 지표 단원을 참조하십시오.

SQL 테이블 크기 제한

Aurora MySQL DB 클러스터의 경우 최대 테이블 크기는 64테비바이트(TiB)입니다. Aurora PostgreSQL DB 클러스터의 경우 최대 테이블 크기는 32테비바이트(TiB)입니다. 테이블 디자인 모범 사례(예: 대용량 테이블 분할)를 따르는 것이 좋습니다.

테이블스페이스 ID 제한

Aurora MySQL 최대 테이블스페이스 ID는 2147483647입니다. 테이블을 자주 만들고 삭제하는 경우 테이블스페이스 ID를 알고 있어야 하며 논리적 덤프를 사용할 계획을 세워야 합니다. 자세한 내용은 mysqldump를 사용하여 MySQL에서 Amazon Aurora MySQL로 논리적 마이그레이션 단원을 참조하십시오.

Amazon Aurora 문제 해결

다음 섹션을 바탕으로 Amazon RDS 및 Amazon Aurora의 DB 인스턴스와 관련하여 발생하는 문제를 해결할 수 있습니다.

주제

Amazon RDS API를 사용해 문제를 디버깅하는 방법에 대한 자세한 내용은 Aurora에서 애플리케이션 문제 해결 단원을 참조하세요.

Amazon RDS DB 인스턴스에 연결할 수 없음

DB 인스턴스에 연결할 수 없는 경우 공통적인 원인은 다음과 같습니다.

- 인바운드 규칙 – 로컬 방화벽에서 적용되는 액세스 규칙과 DB 인스턴스에 액세스할 수 있는 권한이 부여된 IP 주소가 일치하지 않을 수 있습니다. 보안 그룹의 인바운드 규칙에 문제가 있을 가능성이 매우 높습니다.

  기본적으로 DB 인스턴스는 액세스를 허용하지 않습니다. 액세스 권한은 VPC와 연결된 보안 그룹을 통해 부여되며, 이는 DB 인스턴스로 들어오고 나가는 트래픽을 허용합니다. 필요한 경우 특정 상황에 대한 인바운드 및 아웃바운드 규칙을 보안 그룹에 추가합니다. IP 주소, IP 주소의 범위 또는 다른 VPC 보안 그룹을 지정할 수 있습니다.

  > **Note**
  > 새 인바운드 규칙을 추가할 때 원본 의 내 IP를 선택하여 브라우저에서 감지된 IP 주소에서 DB 인스턴스에 액세스하도록 허용할 수 있습니다.

보안 그룹 설정에 대한 자세한 내용은 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다. 단원을 참조하십시오.

> **Note**
>
> 169.254.0.0/16 범위의 IP 주소에서 클라이언트 연결은 허용되지 않습니다. 이는 로컬 링크 주소 지정에 사용되는 APIPA(Automatic Private IP Addressing) 범위입니다.

- 퍼블릭 액세스 가능성 – 클라이언트 애플리케이션을 사용하는 등 VPC 외부에서 DB 인스턴스에 연결하려면 인스턴스에 퍼블릭 IP 주소가 할당되어 있어야 합니다.

  인스턴스에 공개적으로 액세스할 수 있도록 하려면 인스턴스를 수정하고 Public accessibility(퍼블릭 액세스 가능성)에서 예를 선택합니다. 자세한 내용은 VPC에 있는 DB 클러스터를 인터넷에서 숨기기 단원을 참조하십시오.
- 포트 – 로컬 방화벽 제한 때문에 DB 인스턴스를 만들 때 지정한 포트를 사용하여 통신을 주고받을 수 없습니다. 이 경우 네트워크에서 지정한 포트를 인바운드 및 아웃바운드 통신에 사용할 수 있는지 여부를 네트워크 관리자에게 확인하십시오.
- 가용성 – 새로 생성한 DB 인스턴스의 경우 DB 인스턴스를 사용할 준비가 될 때까지 DB 인스턴스의 상태는 `creating`입니다. 상태가 `available`로 변경되면 DB 인스턴스에 연결할 수 있습니다. DB 인스턴스의 크기에 따라, 인스턴스를 사용할 수 있을 때까지 최장 20분까지 걸릴 수 있습니다.
- 인터넷 게이트웨이 – DB 인스턴스에 공개적으로 액세스할 수 있도록 하려면 DB 서브넷 그룹의 서브넷에 인터넷 게이트웨이가 있어야 합니다.

  서브넷에 인터넷 게이트웨이를 구성하려면

  1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/rds/에서 Amazon RDS 콘솔을 엽니다.
  2. 탐색 창에서 데이터베이스를 선택한 다음 DB 인스턴스의 이름을 선택합니다.
  3. 연결&보안 탭에서, VPC에서의 VPC ID 값과 서브넷에서의 서브넷 ID 값을 적어둡니다.
  4. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
  5. 탐색 창에서 [Internet Gateways]를 선택합니다. VPC에 인터넷 게이트웨이가 연결되어 있는지 확인합니다. 또는 인터넷 게이트웨이 생성을 선택하여 인터넷 게이트웨이를 만듭니다. 인터넷 게이트웨이를 선택한 후 VPC에 연결을 선택하고 지침에 따라 VPC에 연결합니다.
  6. 탐색 창에서 서브넷을 선택한 후 해당 서브넷을 선택합니다.

7. [라우팅 테이블(Route table)] 탭에서 대상 위치로 `0.0.0.0/0` 경로가 있으며, VPC의 대상으로 해당 인터넷 게이트웨이가 있는지 확인합니다.

   IPv6 주소를 이용해 인스턴스에 연결하는 경우 인터넷 게이트웨이를 가리키는 모든 IPv6 트래픽(`::/0`)에 대한 경로가 있는지 확인합니다. 그렇지 않으면 다음을 수행하십시오.

   a. 라우팅 테이블의 ID(rtb-xxxxxxxx)를 선택해 해당 라우팅 테이블로 이동합니다.
   b. 라우팅 탭에서 라우팅 편집을 선택합니다. 라우팅 추가를 선택하고 대상 위치로 `0.0.0.0/0`을, 대상으로 인터넷 게이트웨이를 사용합니다.

      IPv6의 경우 라우팅 추가를 선택하고 대상 위치로 `::/0`을, 대상으로 인터넷 게이트웨이를 사용합니다.
   c. 라우팅 저장을 선택합니다.

   또한 IPv6 엔드포인트에 연결하려는 경우 클라이언트 IPv6 주소 범위가 DB 인스턴스에 연결할 수 있는 권한이 있는지 확인합니다.

자세한 내용은 VPC에서 DB 클러스터를 사용한 작업 단원을 참조하세요.

DB 인스턴스 연결 테스트

공통 Linux 또는 Microsoft Windows 도구를 사용하여 DB 인스턴스에 대한 연결을 테스트할 수 있습니다.

Linux 또는 Unix 터미널에서 다음을 입력하여 연결을 테스트할 수 있습니다. *`DB-instance-endpoint`*를 엔드포인트로 대체하고 *`port`*를 DB 인스턴스의 포트로 대체합니다.

```
nc -zv DB-instance-endpoint port
```

예를 들어, 다음은 샘플 명령과 반환 값을 보여 줍니다.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows 사용자는 Telnet을 사용하여 DB 인스턴스에 대한 연결을 테스트할 수 있습니다. Telnet 작업은 연결 테스트 이외의 목적으로는 지원되지 않습니다. 연결에 성공한 경우 이 작업을 수행할 때 아무런 메시지도 반환되지 않습니다. 연결에 실패한 경우 다음과 같은 오류 메시지가 수신됩니다.

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

Telnet 작업으로 성공 메시지가 반환되면 보안 그룹이 올바로 구성된 것입니다.

> **Note**
>
> Amazon RDS는 ping을 포함하여 ICMP(Internet Control Message Protocol) 트래픽을 수락하지 않습니다.

연결 인증 문제 해결

경우에 따라서는 DB 인스턴스에 연결할 수 있지만 인증 오류가 발생하는 경우가 있습니다. 이러한 경우 DB 인스턴스에 대한 마스터 사용자 암호를 재설정하는 것이 좋습니다. RDS 인스턴스를 수정하여 이 작업을 수행할 수 있습니다.

Amazon RDS 보안 문제

보안 문제를 방지하려면 사용자 계정에 AWS 계정 루트 사용자 이메일 주소와 암호를 사용하지 마세요. 모범 사례에 따라 루트 사용자로 사용자를 생성하고 이런 사용자를 DB 사용자 계정에 할당하는 것이 좋습니다. 필요한 경우 루트 사용자로 다른 사용자 계정을 만들 수도 있습니다.

사용자 생성에 대한 자세한 정보는 AWS 계정에서 IAM 사용자 생성을 참조하세요. AWS IAM Identity Center에서의 사용자 생성에 대한 자세한 정보는 IAM Identity Center에서 ID 관리 섹션을 참조하십시오.

오류 메시지 "계정 속성을 불러오지 못했습니다. 일부 콘솔 기능이 손상되었을 수 있습니다."

여러 가지 이유로 이 오류가 발생할 수 있습니다. 계정에 권한이 없거나 계정이 제대로 설정되지 않았기 때문일 수 있습니다. 새 계정인 경우 계정이 준비되기까지 충분한 시간이 지나지 않았을 수도 있습

니다. 기존 계정인 경우 DB 인스턴스 생성과 같은 특정 작업을 수행하기 위한 액세스 정책에 권한이 없을 수 있습니다. 이 문제를 해결하려면 관리자가 필요한 역할을 해당 계정에 제공해야 합니다. 자세한 내용은 IAM 설명서를 참조하세요.

DB 인스턴스 소유자 암호 재설정

DB 클러스터가 잠긴 잠긴 경우 마스터 사용자로 로그인할 수 있습니다. 그런 다음 다른 관리 사용자 또는 역할에 대한 자격 증명을 재설정할 수 있습니다. 마스터 사용자로 로그인할 수 없는 경우 AWS 계정 소유자가 마스터 사용자 암호를 재설정할 수 있습니다. 재설정해야 할 관리 계정 또는 역할에 대한 자세한 내용은 마스터 사용자 계정 권한 단원을 참조하십시오.

Amazon RDS 콘솔, AWS CLI 명령 modify-db-instance 또는 ModifyDBInstance API 작업을 사용하여 DB 인스턴스 암호를 변경할 수 있습니다. DB 클러스터에 있는 DB 인스턴스를 수정하는 것에 대한 자세한 내용은 DB 클러스터에서 DB 인스턴스 수정 단원을 참조하십시오.

Amazon RDS DB 인스턴스 중단 또는 재부팅

DB 인스턴스가 재부팅되면 DB 인스턴스가 중단될 수 있습니다. 이는 DB 인스턴스가 액세스할 수 없는 상태로 전환되거나 데이터베이스가 다시 시작될 때도 발생할 수 있습니다. DB 인스턴스를 수동으로 재부팅하면 재부팅이 수행될 수 있습니다. DB 인스턴스 설정을 변경하여 이 변경 사항을 적용하기 위해 재부팅해야 할 때 재부팅이 수행될 수 있습니다.

DB 인스턴스 재부팅은 설정을 변경하여 재부팅해야 할 때 또는 수동으로 재부팅할 때 이 발생합니다. 설정을 변경하고 변경 사항을 즉시 적용할 것을 요청하는 경우 재부팅이 수행될 수 있습니다. 또는 DB 인스턴스의 유지 관리 기간 중에 재부팅이 수행될 수 있습니다.

다음 중 한 가지가 발생할 때는 그 즉시 DB 인스턴스가 재부팅됩니다.

- DB 인스턴스에 대한 백업 보존 기간을 0에서 0이 아닌 값으로 변경하거나 0이 아닌 값에서 0으로 변경합니다. 그런 다음 즉시 적용을 `true`로 설정합니다.
- DB 인스턴스 클래스를 변경하고 즉시 적용이 `true`로 설정된 경우

유지 관리 기간 중에 다음 중 한 가지가 발생할 때 DB 인스턴스가 재부팅됩니다.

- DB 인스턴스에 대한 백업 보존 기간을 0에서 0이 아닌 값으로 변경하거나 0이 아닌 값에서 0으로 변경하고 즉시 적용이 `false`로 설정된 경우
- DB 인스턴스 클래스를 변경하고 즉시 적용이 `false`로 설정된 경우

DB 파라미터 그룹에서 정적 파라미터를 변경하면 해당 변경 사항은 파라미터 그룹과 연결된 DB 인스턴스를 재부팅해야 적용됩니다. 변경 작업을 수행하려면 수동 재부팅이 필요합니다. 유지 관리 기간 중에는 DB 인스턴스가 자동으로 재부팅되지 않습니다.

Amazon RDS DB 파라미터 변경 사항이 적용 안 됨

경우에 따라 DB 파라미터 그룹에서 파라미터를 변경할 수 있지만 해당 변경 사항이 적용되지 않을 수 있습니다. 이 경우 DB 파라미터 그룹과 연결된 DB 인스턴스를 재부팅해야 할 수 있습니다. 동적 파라미터를 변경하면 해당 변경 사항이 즉시 적용됩니다. 정적 파라미터를 변경하면 해당 변경 사항은 파라미터 그룹과 연결된 DB 인스턴스를 재부팅해야 적용됩니다.

RDS 콘솔을 사용하여 DB 인스턴스를 재부팅할 수 있습니다. 또는 `RebootDBInstance` API 작업을 명시적으로 호출할 수 있습니다. DB 인스턴스를 다중 AZ 배포로 생성한 경우에는 장애 조치 없이 재부팅할 수 있습니다. 정적 파라미터 변경 후 연결된 DB 인스턴스를 재부팅하도록 하면 잘못된 파라미터 구성이 API 호출에 영향을 주는 위험을 완화할 수 있습니다. 예를 들면 `ModifyDBInstance`를 호출하여 DB 인스턴스 클래스를 변경하는 경우입니다. 자세한 내용은 Amazon Aurora에서 DB 파라미터 그룹의 파라미터 수정 단원을 참조하십시오.

Amazon Aurora의 여유 메모리 부족

여유 메모리는 데이터베이스 엔진에서 사용할 수 있는 DB 인스턴스의 총 랜덤 액세스 메모리(RAM)입니다. 이는 여유 운영 체제(OS) 메모리와 사용 가능한 버퍼 및 페이지 캐시 메모리의 합계입니다. 데이터베이스 엔진이 호스트의 메모리 대부분을 사용하지만, OS 프로세스도 일부 RAM을 사용합니다. 현재 데이터베이스 엔진에 할당되거나 OS 프로세스에서 사용하는 메모리는 여유 메모리에 포함되지 않습니다. 데이터베이스 엔진의 메모리가 부족하면 DB 인스턴스는 버퍼링 및 캐싱에 일반적으로 사용되는 임시 공간을 이용하게 됩니다. 앞서 언급했듯이 이 임시 공간은 여유 메모리에 포함됩니다.

Amazon CloudWatch의 `FreeableMemory` 지표를 사용하여 여유 메모리를 모니터링할 수 있습니다. 자세한 내용은 Amazon Aurora의 모니터링 도구 단원을 참조하십시오.

DB 인스턴스에서 사용 가능한 메모리가 계속해서 부족하거나 스왑 공간을 사용하는 경우 더 큰 DB 인스턴스 클래스로 스케일 업하는 것을 고려하세요. 자세한 내용은 Amazon Aurora DB 인스턴스 클래스 단원을 참조하십시오.

메모리 설정을 변경할 수도 있습니다. 예를 들어, Aurora MySQL 에서는 `innodb_buffer_pool_size` 파라미터의 크기를 조정할 수 있습니다. 이 파라미터는 기본적으로 실제 메모리의 75%로 설정됩니다. MySQL 문제 해결 팁을 자세히 알아보려면 Amazon RDS for MySQL 데이터베이스에서 여유 메모리가 부족한 문제를 어떻게 해결할 수 있습니까?를 참조하세요.

Aurora Serverless v2의 경우 `FreeableMemory`는 Aurora Serverless v2 DB 인스턴스가 최대 용량으로 확장될 때 사용 가능한 미사용 메모리의 양을 나타냅니다. 인스턴스를 비교적 낮은 용량으로 스케일 다운할 수 있지만, 인스턴스가 스케일 업될 수 있기 때문에 계속해서 `FreeableMemory`에 대해 높은 값을 보고합니다. 이 메모리는 현재 사용할 수 없지만, 필요한 경우 가져올 수 있습니다.

현재 용량이 최대 용량 미만인 모든 Aurora 용량 단위(ACU)에 대해 `FreeableMemory`가 약 2GiB씩 증가합니다. 따라서 DB 인스턴스가 가능한 한 높게 확장될 때까지 이 지표는 0에 접근하지 않습니다.

이 지표가 `0` 값에 가까워지면 DB 인스턴스가 최대한 스케일 업된 것입니다. 사용 가능한 메모리 한계에 가까워지고 있음을 나타냅니다. 클러스터에 대한 최대 ACU 설정을 늘리는 것이 좋습니다. 이 지표가 리더 DB 인스턴스에서 `0` 값에 가까워지는 경우 클러스터에 리더 DB 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 DB 인스턴스에 분산하여 각 리더 DB 인스턴스의 메모리 사용량을 줄일 수 있습니다. 자세한 내용은 Aurora Serverless v2에 대한 중요 Amazon CloudWatch 지표 단원을 참조하세요.

Aurora Serverless v1의 경우 더 많은 ACU를 사용하도록 용량 범위를 변경할 수 있습니다. 자세한 내용은 Aurora Serverless v1 DB 클러스터 수정 단원을 참조하십시오.

Amazon Aurora MySQL 복제 문제

일부 MySQL 복제 문제도 Aurora MySQL에 적용됩니다. 이러한 문제를 진단하고 해결할 수 있습니다.

주제

읽기 전용 복제본 간 지연 문제 진단 및 해결

MySQL 읽기 전용 복제본을 생성하고 읽기 전용 복제본을 사용할 수 있게 된 후 Amazon RDS는 우선 읽기 전용 복제본 생성 작업이 시작된 시간부터 원본 DB 인스턴스에서 변경된 사항을 복제합니다. 이 단계에서 읽기 전용 복제본의 복제 지연 시간은 0보다 큽니다. Amazon RDS `AuroraBinlogReplicaLag` 지표를 보고 Amazon CloudWatch에서 이 지연 시간을 모니터링할 수 있습니다.

AuroraBinlogReplicaLag 메트릭은 MySQL Seconds_Behind_Master 명령의 SHOW REPLICA STATUS 필드의 값을 보고합니다. 자세한 내용은 MySQL 설명서의 SHOW REPLICA STATUS 문을 참조하세요.

AuroraBinlogReplicaLag 지표가 0에 도달하면 복제본이 원본 DB 인스턴스를 따라잡은 것입니다. AuroraBinlogReplicaLag 메트릭이 -1을 반환하는 경우에는 복제가 활성 상태가 아닐 수 있습니다. 복제 오류 문제를 해결하는 방법은 MySQL 읽기 복제 오류 진단 및 해결 단원을 참조하십시오. AuroraBinlogReplicaLag 값이 -1인 경우 Seconds_Behind_Master 값을 결정할 수 없거나 이 값이 NULL이라는 의미일 수도 있습니다.

> **Note**
>
> 이전 버전의 Aurora MySQL에는 SHOW REPLICA STATUS 대신 SHOW SLAVE STATUS가 사용되었습니다. Aurora MySQL 버전 1 또는 2을 사용하는 경우 SHOW SLAVE STATUS를 사용합니다. Aurora MySQL 버전 3 이상의 경우 SHOW REPLICA STATUS를 사용합니다.

네트워크가 중단된 기간 동안이나 유지 관리 기간 중에 패치가 적용될 때 AuroraBinlogReplicaLag 지표는 -1을 반환합니다. 이 경우에는 네트워크 연결이 복원되거나 유지 관리 기간이 종료되기를 기다린 후 AuroraBinlogReplicaLag 지표를 다시 확인합니다.

MySQL 읽기 전용 복제 기술은 비동기식입니다. 따라서 소스 DB 인스턴스의 BinLogDiskUsage 지표와 읽기 전용 복제본의 AuroraBinlogReplicaLag 지표가 가끔 증가할 수도 있습니다. 예를 들어, 원본 DB 인스턴스에 대량의 쓰기 작업이 병렬로 발생하는 경우를 생각해 보십시오. 동시에 읽기 전용 복제본에 대한 쓰기 작업은 단일 I/O 스레드를 사용하여 직렬화됩니다. 이러한 상황으로 인해 원본 인스턴스와 읽기 전용 복제본 사이에 지연 시간이 발생할 수 있습니다.

읽기 전용 복제본과 MySQL에 대한 자세한 내용은 MySQL 설명서의 복제 구현 세부 정보를 참조하십시오.

다음을 수행하여 원본 DB 인스턴스에 대한 업데이트와 읽기 전용 복제본에 대한 후속 업데이트 사이의 지연 시간을 줄일 수 있습니다.

- 읽기 전용 복제본의 DB 인스턴스 클래스를 원본 DB 인스턴스와 비슷한 스토리지 크기로 설정합니다.
- 원본 DB 인스턴스와 읽기 전용 복제본에 사용되는 DB 파라미터 그룹의 파라미터 설정이 호환되는지 확인합니다. 자세한 정보와 예는 다음 섹션에서 max_allowed_packet 파라미터에 대해 설명한 내용을 참조하십시오.

- 쿼리 캐시를 비활성화합니다. 자주 수정되는 테이블의 경우, 쿼리 캐시를 사용하면 캐시가 자주 잠기고 새로 고쳐지기 때문에 복제 지연이 늘어날 수 있습니다. 이럴 경우 쿼리 캐시를 비활성화하면 복제 지연이 줄어드는 효과를 볼 수도 있습니다. DB 인스턴스에 대한 DB 파라미터 그룹에서 `query_cache_type parameter`를 0으로 설정하여 쿼리 캐시를 비활성화할 수 있습니다. 쿼리 캐시에 대한 자세한 정보는 쿼리 캐시 구성을 참조하십시오.
- MySQL용 InnoDB의 읽기 전용 복제본에서 버퍼 풀을 워밍합니다. 예를 들어, 자주 업데이트되는 작은 테이블 집합이 있고 InnoDB 또는 XtraDB 테이블 스키마를 사용하고 있다고 가정해 보십시오. 이 경우 읽기 전용 복제본에 해당 테이블을 덤프합니다. 그러면 데이터베이스 엔진이 디스크에서 해당 테이블의 행을 전체적으로 검사한 다음 버퍼 풀에 캐시합니다. 이 방법을 사용하면 복제본 지연 시간을 줄일 수 있습니다. 다음은 그 한 예입니다.

  대상 LinuxmacOS, 또는Unix:

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

  Windows의 경우:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

MySQL 읽기 복제 오류 진단 및 해결

Amazon RDS는 읽기 전용 복제본의 복제 상태를 모니터링합니다. RDS는 어떤 이유로든 복제가 중지되는 경우 읽기 전용 복제본 인스턴스의 복제 상태 필드를 `Error`로 업데이트합니다. 복제 오류 필드를 확인하여 MySQL 엔진에서 발생한 관련 오류의 세부 정보를 검토할 수 있습니다. RDS-EVENT-0045, RDS-EVENT-0046 및 RDS-EVENT-0057을 포함하여 읽기 전용 복제본의 상태를 나타내는 이벤트도 생성됩니다. 이벤트와 이벤트 구독에 대한 자세한 내용은 Amazon RDS 이벤트 알림 작업 단원을 참조하십시오. MySQL 오류 메시지가 반환되는 경우 MySQL 오류 메시지 설명서에 있는 오류를 확인하세요.

복제 오류의 원인이 되는 공통적인 상황은 다음과 같습니다.

- 읽기 전용 복제본에 대한 max_allowed_packet 파라미터의 값은 원본 DB 인스턴스에 대한 max_allowed_packet 파라미터보다 작습니다.

  max_allowed_packet 파라미터는 DB 파라미터 그룹에서 설정할 수 있는 사용자 지정 파라미터입니다. max_allowed_packet 파라미터는 데이터베이스에서 실행할 수 있는 데이터 조작 언어(DML)의 최대 크기를 지정하는 데 사용됩니다. 경우에 따라서는 원본 DB 인스턴스의 max_allowed_packet 값이 읽기 전용 복제본에 대한 max_allowed_packet 값보다 클 수도 있습니다. 이러한 경우 복제 프로세스에서 오류가 발생하여 복제가 중지될 수도 있습니다. 가장 흔한 오류는 packet bigger than 'max_allowed_packet' bytes입니다. 원본 및 읽기 전용 복제본이 동일한 max_allowed_packet 파라미터 값을 가진 DB 파라미터 그룹을 사용하도록 하여 이 오류를 해결할 수 있습니다.
- 읽기 전용 복제본의 테이블에 쓰기 작업 중일 때. 읽기 전용 복제본에서 인덱스를 생성할 경우 read_only 파라미터를 0으로 설정하여 인덱스를 생성해야 합니다. 읽기 전용 복제본에 있는 테이블에 데이터를 쓰면 복제가 중단될 수 있습니다.
- MyISAM과 같은 비트랜잭션 스토리지 엔진 사용. 읽기 전용 복제본에는 트랜잭션 스토리지 엔진이 필요합니다. 복제는 MySQL 또는 MariaDB용 InnoDB에 대해서만 지원됩니다.

  다음 명령으로 MyISAM 테이블을 InnoDB로 변환할 수 있습니다.

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- SYSDATE()와 같이 안전하지 않은 비결정적 쿼리를 사용하는 경우. 자세한 내용은 MySQL 설명서의 이진 로깅에서 안전한 문과 안전하지 않은 문 결정을 참조하십시오.

다음 단계를 통해 복제 오류를 해결할 수 있습니다.

- 논리적 오류가 발생하고 이 오류를 건너뛰어도 안전할 경우 현재 복제 오류 건너뛰기에 설명된 단계를 따르십시오. Aurora MySQL DB 인스턴스에서는 mysql_rds_skip_repl_error 프로시저를 포함한 버전이 실행 중이어야 합니다. 자세한 내용은 mysql_rds_skip_repl_error 단원을 참조하세요.
- 이진 로그(binlog) 위치 문제가 발생하는 경우 복제본 재생 위치를 변경할 수 있습니다. 그러려면 Aurora MySQL 버전 1 및 2에서는 mysql.rds_next_master_log 명령을 사용합니다. Aurora MySQL 버전 3 이상에서는 mysql.rds_next_source_log 명령을 사용합니다. Aurora MySQL DB 인스턴스에서 복제본 재생 위치를 변경하려면 이 명령을 지원하는 버전을 실행해야 합니다. 버전 정보는 mysql_rds_next_master_log를 참조하세요.
- 높은 DML 로드 때문에 일시적인 성능 문제가 발생할 경우 읽기 전용 복제본의 DB 파라미터 그룹에서 innodb_flush_log_at_trx_commit 파라미터를 2로 설정할 수 있습니다. 그러면 일시적으

로 원자성, 일관성, 격리성 및 내구성(ACID)이 감소하지만 읽기 전용 복제본이 변화를 따라잡는 데 도움이 될 수 있습니다.

- 읽기 전용 복제본을 삭제하고 동일한 DB 인스턴스 식별자를 사용하여 인스턴스를 생성할 수 있습니다. 이렇게 하면 엔드포인트는 이전 읽기 전용 복제본의 엔드포인트와 동일하게 유지됩니다.

복제 오류가 해결되면 Replication State가 replicating으로 변경됩니다. 자세한 내용은 MySQL 읽기 전용 복제본의 문제 해결을 참조하십시오.

복제 중지 오류

`mysql.rds_skip_repl_error` 명령을 호출할 때 복제본이 중단되었거나 사용 중지되었다는 오류 메시지가 표시될 수 있습니다.

이 오류 메시지는 복제가 중지되었고 재시작할 수 없기 때문에 표시됩니다.

많은 수의 오류를 건너뛰어야 하는 경우, 복제 지연이 이진 로그 파일의 기본 보관 기간 이상으로 늘어날 수 있습니다. 이 경우 이진 로그 파일이 복제본에서 재생되기 전에 지워지기 때문에 치명적 오류가 발생할 수 있습니다. 이 제거는 복제를 중지시키며, 복제 오류를 건너뛰기 위해 더 이상 `mysql.rds_skip_repl_error` 명령을 호출할 수 없습니다.

이 문제는 복제 원본에서 이진 로그 파일이 보관되는 시간을 늘림으로써 완화할 수 있습니다. binlog 보관 시간을 늘린 후에 복제를 재시작하고 필요에 따라 `mysql.rds_skip_repl_error` 명령을 호출할 수 있습니다.

binlog 보관 시간을 설정하려면 mysql_rds_set_configuration 프로시저를 사용합니다. 'binlog 보관 시간' 구성 파라미터와 DB 클러스터에 binlog 파일을 보관할 시간(최대 2160시간(90일))을 함께 지정합니다. Aurora MySQL의 기본값은 24(1일)입니다. 다음 예제에서는 binlog 파일의 보관 기간을 48시간으로 설정합니다.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

읽기 전용 복제본 복제가 메타데이터 구조를 초기화하지 못함

복제를 시작하려고 할 때 다음과 같은 오류 메시지가 표시됩니다.

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

이 오류는 복제본의 메타데이터 구조에 문제가 있는 경우에 발생합니다. 메타데이터 구조를 수정하려면 새 복제본을 생성해야 합니다.

향후 이 문제가 발생하지 않도록 하려면 다음 작업 중 하나를 수행합니다.

- 가능하면 복제본에서 멀티스레딩을 비활성화합니다. MySQL 8.0.27부터 멀티스레딩이 기본적으로 활성화됩니다.
- 복제본에 멀티스레딩을 사용해야 하는 경우 GTID 기반 복제를 사용하는 것이 좋습니다. 자세한 내용은 GTID 기반 복제 사용 단원을 참조하십시오.

Amazon RDS API 참조

Amazon RDS는 AWS Management Console 및 AWS Command Line Interface(AWS CLI) 외에 API도 제공합니다. API를 사용하여 Amazon RDS의 DB 인스턴스 및 기타 객체 관리 작업을 자동화할 수 있습니다.

- API 작업의 알파벳순 목록은 작업을 참조하십시오.
- 데이터 형식에 대한 알파벳순 목록은 데이터 형식을 참조하십시오.
- 공통 쿼리 파라미터 목록은 공통 파라미터를 참조하십시오.
- 오류 코드에 대한 설명은 공통 오류를 참조하십시오.

AWS CLI에 대한 자세한 내용은 Amazon RDS용 AWS Command Line Interface 참조를 참조하세요.

주제

쿼리 API 사용

다음 단원에서는 쿼리 API와 함께 사용되는 파라미터 및 요청 인증에 대해 간략하게 설명합니다.

쿼리 API의 작동 방식에 대한 일반적인 내용은 Amazon EC2 API Reference의 쿼리 요청을 참조하십시오.

쿼리 파라미터

HTTP 쿼리 기반 요청은 GET 또는 POST와 같은 HTTP 동사와 `Action` 쿼리 매개 변수를 사용하는 HTTP 요청입니다.

각 쿼리 요청은 인증 및 작업을 처리할 수 있도록 일부 공통 파라미터를 포함해야 합니다.

일부 작업은 파라미터의 목록을 허용합니다. 이러한 목록은 `param.`*n* 표기법을 사용하여 지정됩니다. *n*의 값은 1부터 시작하는 정수입니다.

Amazon RDS 리전과 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조의 리전 및 엔드포인트 섹션에서 Amazon Relational Database Service(RDS)를 참조하세요.

쿼리 요청 인증

HTTPS를 통해서만 쿼리 요청을 보낼 수 있으며 모든 쿼리 요청에는 서명이 포함되어야 합니다. AWS 서명 버전 4 또는 서명 버전 2를 사용해야 합니다. 자세한 정보는 서명 버전 4 서명 프로세스 및 서명 버전 2 서명 프로세스를 참조하십시오.

Aurora에서 애플리케이션 문제 해결

Amazon RDS는 Amazon RDS API와 상호 작용하는 동안 발생하는 문제를 해결할 때 도움이 되도록 구체적이고 서술적인 오류를 제공합니다.

주제

- 오류 검색
- 문제 해결 팁

Amazon RDS DB 인스턴스 문제 해결에 대한 자세한 내용은 Amazon Aurora 문제 해결 단원을 참조하십시오.

오류 검색

일반적으로 사용자는 시간을 소비하여 결과를 처리하기 전에 애플리케이션이 먼저 해당 요청으로 오류가 발생되는지 여부를 확인하려고 합니다. 오류 발생 여부를 확인하는 가장 쉬운 방법은 Amazon RDS API의 응답에서 `Error` 노드를 찾는 것입니다.

XPath 구문은 `Error` 노드의 존재 여부를 검색하는 간단한 방법을 제공합니다. 또한 오류 코드와 메시지를 비교적 쉽게 검색할 수 있는 방법을 제공합니다. 다음 코드 조각에서는 요청 중에 오류가 발생했는지 여부를 파악하기 위해 Perl 및 XML::Xpath 모듈을 사용합니다. 오류가 발생되면 코드는 응답에 첫 번째 오류 코드와 메시지를 인쇄합니다.

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

문제 해결 팁

다음 절차를 통해 Amazon RDS API의 문제를 진단하고 해결하는 것이 좋습니다.

- 타겟팅하는 AWS 리전에서 Amazon RDS가 정상적으로 작동하는지 http://status.aws.amazon.com에서 확인합니다.
- 요청 구조 확인.

  각 Amazon RDS 작업에 대한 참조 페이지는 Amazon RDS API 참조에 있습니다. 파라미터를 올바르게 사용하고 있는지 여부를 다시 확인합니다. 어떤 문제가 발생할 수 있을지 알아보려면 샘플 요청이나 사용자 시나리오를 살펴보고 이러한 예시가 유사한 작업을 하는지 확인하세요.
- AWS re:Post 확인

  Amazon RDS와 관련하여 다른 사람들이 경험한 문제에 대한 해결책을 검색할 수 있는 개발 커뮤니티가 있습니다. 주제를 보려면 AWS re:Post로 이동하세요.

문서 기록

현재 API 버전: 2014-10-31

아래 표에 Amazon Aurora 사용 설명서의 주요 변경 사항이 설명되어 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다. Amazon Relational Database Service(Amazon RDS)에 대한 자세한 내용은 Amazon Relational Database Service 사용 설명서를 참조하십시오.

> **Note**
>
> 2018년 8월 31일 이전의 Amazon Aurora은 Amazon Relational Database Service 사용 설명서로 작성되었습니다. Aurora의 이전 문서 기록은 Amazon Relational Database Service 사용 설명서의 문서 기록을 참조하십시오.

데이터베이스 관련 새로운 소식 페이지에서 새로운 Amazon Aurora 기능을 필터링할 수 있습니다. [제품(Products)]에서 Amazon Aurora를 선택합니다. 그런 다음 **`global database`** 또는 **`Serverless`**와 같은 키워드를 사용하여 검색합니다.

변경 사항	설명	날짜
기존 정책 업데이트	Amazon RDS는 `AWSServiceRoleForRDSBeta` 서비스 연결 역할의 `AmazonRDSBetaServiceRolePolicy` 에서 `sns:Publish` 권한을 제거했습니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2024년 8월 7일
기존 정책 업데이트	Amazon RDS는 `AWSServiceRoleForRDSPreview` 서비스 연결 역할의 `AmazonRDSPreviewServiceRolePolicy` 에서 `sns:Publish` 권한을 제거	2024년 8월 7일

	했습니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	
MySQL용 AWS ODBC 드라이버 일반 공개	MySQL용 Amazon Web Services(AWS) ODBC 드라이버는 Aurora MySQL의 고가용성을 위해 설계된 클라이언트 드라이버입니다. 자세한 내용은 MySQL용 Amazon Web Services(AWS) ODBC 드라이버로 Aurora MySQL에 연결을 참조하세요.	2024년 7월 18일
더 많은 리전에서 Aurora Serverless v2용 RDS 데이터 API 사용 가능	이제 여러 추가 AWS 리전에서 Aurora PostgreSQL에 RDS 데이터 API를 사용할 수 있습니다. RDS 데이터 API에 대한 리전 지원에 대한 자세한 내용은 Aurora PostgreSQL Serverless v2 및 프로비저닝에 데이터 API 사용을 참조하세요.	2024년 7월 9일
AWS Python 드라이버 정식 출시	Amazon Web Services(AWS) Python 드라이버는 고급 Python 래퍼로 설계되었습니다. 이 래퍼는 오픈 소스 Psycopg 드라이버의 기능을 보완하고 확장합니다. 자세한 내용은 Connecting to Aurora DB clusters with the AWS drivers를 참조하세요.	2024년 5월 23일

중국 리전에서 제로 ETL 통합 사용 가능	이제 AWS 리전 중국(베이징)과 중국(닝샤)에서 제로 ETL 통합을 사용할 수 있습니다. 자세한 내용은 Amazon Redshift가 구성된 제로 ETL 통합을 참조하세요.	2024년 5월 21일
더 많은 리전에서 RDS 프록시 사용 가능	이제 RDS Proxy를 아시아 태평양(하이데라바드), 아시아 태평양(멜버른), 중동(UAE), 이스라엘(텔아비브), 캐나다 서부(캘거리), 유럽(취리히) 리전에서 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2024년 5월 21일
Amazon RDS 추가 지원	이제 Aurora MySQL 버전 2나 3 또는 Aurora PostgreSQL 버전 11 데이터베이스를 생성하거나 복원하면 해당 데이터베이스가 Amazon RDS 확장 지원에 자동으로 등록되므로 기존 애플리케이션이 그대로 계속 작동합니다. RDS 확장 지원을 옵트아웃하면 데이터베이스 엔진에 적용되는 Aurora 표준 지원 종료일이 지난 후 요금이 부과되는 것을 막을 수 있습니다. 자세한 내용은 Amazon RDS 추가 지원 사용을 참조하세요.	2024년 3월 21일

제로 ETL 통합의 데이터 필터링	Amazon RDS는 Amazon Redshift와 제로 ETL 통합에 데이터베이스 및 테이블 수준에서 데이터 필터링을 지원합니다. 자세한 내용은 Data filtering for Aurora zero-ETL integrations with Amazon Redshift를 참조하세요.	2024년 3월 20일
Aurora MySQL과 Amazon Bedrock 통합	이제 Amazon Aurora MySQL 데이터베이스를 Amazon Bedrock과 통합하여 생성형 AI 애플리케이션을 지원할 수 있습니다. 자세한 내용은 Aurora MySQL과 함께 Amazon Aurora 기계 학습 사용을 참조하세요.	2024년 3월 8일
새 AWS 관리형 정책	Amazon RDS는 RDS Custom이 EC2 인스턴스 프로파일을 통해 자동화 작업 및 데이터베이스 관리 작업을 수행할 수 있도록 `AmazonRDSCustomInstanceProfileRolePolicy` 로 이름이 지정된 새 관리형 정책을 추가했습니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2024년 2월 27일
이스라엘(텔아비브) 리전에서 AWS Secrets Manager에 대해 Amazon RDS 지원	Amazon RDS는 이스라엘(텔아비브) 리전에서 Secrets Manager를 지원합니다. 자세한 내용은 Amazon RDS와 AWS Secrets Manager를 사용한 암호 관리를 참조하세요.	2024년 2월 21일

Amazon RDS 추가 지원	이제 DB 클러스터 및 글로벌 클러스터의 Aurora MySQL 및 Aurora PostgreSQL 메이저 엔진 버전이 Aurora 표준 지원 종료일에 다다르면 Amazon RDS는 Amazon RDS 추가 지원을 자동으로 활성화합니다. 자세한 내용은 Amazon RDS 추가 지원 사용을 참조하세요.	2024년 2월 15일
Aurora PostgreSQL 16.1에서 Babelfish for Aurora PostgreSQL 4.0.0 지원	Aurora PostgreSQL 16.1에서 Babelfish 4.0.0을 지원합니다. 새 기능 목록은 16.1 섹션을 참조하세요. 각 Babelfish 릴리스에서 지원되는 기능 목록은 버전별 Babelfish에서 지원되는 기능 섹션을 참조하세요. 사용에 관한 자세한 내용은 Babelfish for Aurora PostgreSQL 작업을 참조하세요.	2024년 1월 31일
기본 CA 인증서로 업데이트	기본 CA 인증서는 `rds-ca-rsa2048-g1` 로 설정되어 있습니다. 자세한 내용은 SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화를 참조하세요.	2024년 1월 26일
유럽(스페인) 리전에서 RDS 프록시 지원	이제 유럽(스페인) 리전에서 RDS 프록시를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2024년 1월 8일

Aurora PostgreSQL Serverless v2를 사용하고 프로비저닝된 RDS 데이터 API	이제 Aurora PostgreSQL Serverless v2 및 프로비저닝된 DB 클러스터와 함께 RDS 데이터 API를 사용할 수 있습니다. RDS 데이터 API를 사용하면 데이터베이스 드라이버를 사용하거나 연결을 관리하지 않고도 보안 HTTP 엔드포인트를 통해 Aurora 클러스터에 액세스하고 SQL 문을 실행할 수 있습니다. 자세한 내용은 RDS 데이터 API 사용을 참조하세요.	2023년 12월 21일
Amazon Bedrock과 Aurora PostgreSQL 통합	이제 Amazon Aurora PostgreSQL 데이터베이스를 Amazon Bedrock과 통합하여 생성형 AI 애플리케이션을 지원할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에서 Amazon Aurora 기계 학습 사용을 참조하세요.	2023년 12월 21일
Amazon Aurora에서 캐나다 서부(캘거리) 리전 지원	이제 Amazon Aurora를 캐나다 서부(캘거리) 리전에서 사용할 수 있습니다 자세한 내용은 리전 및 가용 영역을 참조하세요.	2023년 12월 20일
Amazon RDS에서 권장 사항 확인 및 권장 사항에 대한 응답 지원	Amazon Aurora 권장 사항에는 이제 임계값 기반 사전 예방적 권장 사항과 기계 학습 기반 사후 대응적 권장 사항이 포함됩니다. 자세한 내용은 Amazon Aurora 권장 사항을 참조하세요.	2023년 12월 19일

[Amazon Redshift가 구성된 Aurora PostgreSQL 제로 ETL 통합(미리 보기)]()	이제 Aurora PostgreSQL 소스 DB 클러스터를 사용하여 Amazon Redshift가 구성된 제로 ETL 통합을 만들 수 있습니다. 미리 보기 릴리스의 경우 미국 동부(오하이오)(us-east-2) AWS 리전의 Amazon RDS 데이터베이스 미리 보기 환경에서 모든 통합을 생성해야 합니다. 자세한 내용은 [Amazon Redshift가 구성된 Amazon Aurora 제로 ETL 통합 작업]()을 참조하세요.	2023년 11월 28일
[Amazon Aurora PostgreSQL, 글로벌 데이터베이스 쓰기 전달 지원]()	이제 Aurora PostgreSQL 기반 글로벌 데이터베이스의 보조 클러스터에서 쓰기 전달을 활성화할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL 글로벌 데이터베이스에서 쓰기 전달 사용]()을 참조하세요.	2023년 11월 9일
[Aurora PostgreSQL, 최적화된 읽기 지원]()	Aurora 최적화된 읽기로 Aurora PostgreSQL의 쿼리 처리 속도를 높일 수 있습니다. 자세한 내용은 [Aurora 최적화된 읽기로 Aurora PostgreSQL 쿼리 성능 개선]()을 참조하세요.	2023년 11월 8일

Amazon RDS, 성능 개선 도우미 지표를 Amazon CloudWatch에 내보내기 가능	성능 개선 도우미를 사용하면 사전 구성된 지표 또는 사용자 지정 지표 대시보드를 Amazon CloudWatch로 내보낼 수 있습니다. 내보낸 지표 대시보드를 CloudWatch 콘솔에서 볼 수 있습니다. 선택한 성능 개선 도우미 지표 위젯을 내보내고 CloudWatch 콘솔에서 지표 데이터를 볼 수도 있습니다. 자세한 내용은 CloudWatch에 성능 개선 도우미 지표 내보내기를 참조하세요.	2023년 11월 8일
Amazon Redshift가 구성된 Aurora MySQL 제로 ETL 통합 정식 출시	Amazon Redshift가 구성된 Aurora MySQL 제로 ETL 통합이 정식 출시되었습니다. 자세한 내용은 Amazon Redshift가 구성된 Amazon Aurora 제로 ETL 통합 작업을 참조하세요.	2023년 11월 7일
Aurora for PostgreSQL, RDS 블루/그린 배포 지원	이제 Aurora PostgreSQL DB 클러스터에서 블루/그린 배포를 생성할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용을 참조하세요.	2023년 10월 26일

Aurora MySQL, AWS KMS keys(SSE-KMS)를 사용한 서버 측 암호화 지원	Aurora MySQL 버전 3.05 이상에서 Amazon S3에서 로드하거나 Amazon S3에 저장하는 데이터의 서버 측 암호화에 AWS 관리형 키 및 고객 관리형 키를 포함한 SSE-KMS를 사용할 수 있습니다. 자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 및 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로부터 데이터 저장을 참조하세요.	2023년 10월 25일
Aurora MySQL 최적화, 데이터베이스 재시작 시간 단축	Aurora MySQL 버전 3.05 이상에서 데이터베이스 재시작 시간을 단축하는 최적화 기능을 도입했습니다. 이러한 최적화를 통해 최적화를 사용하지 않을 때보다 가동 중지 시간이 최대 65% 줄어들고 재시작 후 데이터베이스 워크로드가 중단되는 일도 줄어듭니다. 자세한 내용은 데이터베이스 재시작 시간 단축을 위한 최적화를 참조하세요.	2023년 10월 25일

AWS 관리형 정책으로 업데이트	`AmazonRDSPerformanceInsightsReadOnly` 및 `AmazonRDSPerformanceInsightsFullAccess` 관리형 정책에서 이제 정책 문에 `Sid`(문 ID)를 식별자로 포함합니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2023년 10월 23일
Amazon RDS는 성능 개선 도우미 카운터 지표를 Amazon CloudWatch에 게시	CloudWatch 콘솔의 DB_PERF_INSIGHTS 지표 수학 함수를 사용하면 Amazon RDS에서 성능 개선 도우미 카운터 지표를 쿼리할 수 있습니다. 자세한 내용은 Amazon Aurora를 모니터링하여 CloudWatch 경보 생성을 참조하세요.	2023년 9월 20일
Amazon Aurora에서 AWS Backup를 사용해 시점 복구(PITR) 지원	이제 AWS Backup에서 Aurora 자동(연속) 백업을 관리하고 이를 통해 지정된 시간으로 복원할 수 있습니다. 자세한 내용은 AWS Backup로 DB 클러스터를 특정 시점으로 복원에서 참조하세요.	2023년 9월 7일

Amazon RDS 추가 지원	Amazon Aurora는 Aurora 표준 지원 종료일이 지난 후에도 DB 인스턴스에서 Aurora MySQL 및 Aurora PostgreSQL 메이저 엔진 버전을 계속 실행할 수 있는 기능을 곧 출시한다고 발표했습니다. 자세한 내용은 Amazon RDS 추가 지원 사용을 참조하세요.	2023년 9월 1일
Amazon Aurora MySQL로 Percona XtraBackup에 대해 추가 지원	이제 물리적으로 MySQL 8.0 데이터베이스를 Aurora MySQL 버전 3 DB 클러스터로 마이그레이션할 수 있습니다. 자세한 내용은 Percona XtraBackup 및 Amazon S3를 사용하여 MySQL에서 물리적으로 마이그레이션을 참조하세요.	2023년 8월 24일
Aurora 글로벌 데이터베이스에서 글로벌 데이터베이스 장애 조치 지원	이제 Aurora 글로벌 데이터베이스에서 관리형 글로벌 장애 조치를 지원하므로 실제 리전별 재해나 전체 서비스 수준 중단을 보다 쉽게 복구할 수 있습니다. 이 기능에 대한 자세한 내용은 Aurora 글로벌 데이터베이스에서 관리형 장애 조치 수행을 참조하세요. 이 기능의 이전 명칭은 '계획된 관리형 장애 조치'이며, 현재 명칭은 '전환'입니다. 전환에 대한 자세한 내용은 Amazon Aurora Global Database에서 전환 수행을 참조하세요.	2023년 8월 21일

AWS 관리형 정책 권한 업데이트	`AmazonRDSFullAccess` 관리형 정책에는 일정 기간 동안 성능 분석 보고서를 생성, 확인 및 삭제할 수 있는 새 권한이 있습니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2023년 8월 17일
AWS 관리형 정책 권한 업데이트	`AmazonRDSPerformanceInsightsReadOnly` 관리형 정책에 새 권한을 추가하고 새로운 관리형 정책인 `AmazonRDSPerformanceInsightsFullAccess` 를 추가하면 일정 기간 동안의 DB 로드 분석 보고서를 생성할 수 있습니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2023년 8월 16일
Amazon RDS가 성능 개선 도우미를 통해 DB 로드 기간 분석 지원	성능 개선 도우미를 사용하면 특정 기간 동안의 성과 분석 보고서를 만들 수 있습니다. 보고서는 식별된 인사이트와 성능 문제를 해결하기 위한 권장 사항을 제공합니다. 자세한 내용은 일정 기간 동안의 DB 로드 분석을 참조하세요.	2023년 8월 16일
Amazon Aurora가 DB 클러스터의 자동 백업 보존 지원	이제 삭제된 Aurora 클러스터의 자동 백업을 보존하고 지정된 시점으로 복원할 수 있습니다. 자세한 내용은 자동 백업 보존을 참조하세요.	2023년 8월 4일

이스라엘(텔아비브) 리전에서 Amazon Aurora 사용 가능	이제 이스라엘(텔아비브) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2023년 8월 1일
Amazon Aurora MySQL이 로컬(클러스터 내) 쓰기 전달 지원	이제 리더 DB 인스턴스의 쓰기 작업을 Aurora MySQL DB 클러스터 내의 라이터 DB 인스턴스로 전달할 수 있습니다. 자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 쓰기 전달 사용을 참조하세요.	2023년 7월 31일
Amazon Aurora가 추가 AWS 리전에서 Aurora Serverless v2 지원	이제 아시아 태평양(멜버른) AWS 리전에서 Aurora Serverless v2 DB 클러스터를 만들 수 있습니다. Aurora Serverless v2에 대한 자세한 내용은 Aurora Serverless v2 사용을 참조하세요.	2023년 6월 28일
Amazon Redshift가 구성된 제로 ETL 통합을 도입한 Amazon Aurora(미리 보기)	제로 ETL 통합은 트랜잭션 데이터가 Aurora MySQL DB 클러스터에 기록된 후 몇 초 안에 Amazon Redshift에서 사용할 수 있도록 하기 위한 완전관리형 솔루션을 제공합니다. 자세한 내용은 Amazon Redshift가 구성된 Amazon Aurora 제로 ETL 통합 작업을 참조하세요.	2023년 6월 28일

Amazon RDS가 성능 개선 도우미 대시보드에서 통합된 성능 개선 도우미 및 CloudWatch 지표 보기 제공	Amazon RDS는 이제 성능 개선 도우미 대시보드에서 성능 개선 도우미 및 CloudWatch 지표에 대한 통합 보기를 제공합니다. 자세한 내용은 성능 개선 도우미 대시보드와 결합된 지표 보기를 참조하세요.	2023년 5월 24일
Amazon Aurora에서 db.r7g 인스턴스 클래스 지원	이제 db.r7g 인스턴스 클래스를 사용하여 Aurora DB 클러스터를 생성할 수 있습니다. 자세한 내용은 Aurora DB 인스턴스 클래스를 참조하세요.	2023년 5월 11일
Amazon Aurora에서 새로운 DB 클러스터 스토리지 구성 지원	Aurora I/O-Optimized를 사용하면 읽기 및 쓰기 I/O 작업에 대한 추가 비용 없이 DB 클러스터의 사용량 및 스토리지에 대한 비용만 지불하면 됩니다. 자세한 내용은 Amazon Aurora DB 클러스터 스토리지 구성을 참조하세요.	2023년 5월 11일
Amazon Aurora가 추가 AWS 리전에서 Aurora Serverless v2 지원	이제 아시아 태평양(하이데라바드), 유럽(스페인), 유럽(취리히), 중동(UAE) AWS 리전에서 Aurora Serverless v2 DB 클러스터를 만들 수 있습니다. Aurora Serverless v2에 대한 자세한 내용은 Aurora Serverless v2 사용을 참조하세요.	2023년 5월 4일

Aurora Serverless v1에서 프로비저닝된 상태로 변환 지원	Aurora Serverless v1 DB 클러스터를 프로비저닝된 DB 클러스터로 직접 변환할 수 있습니다. 자세한 내용은 Aurora Serverless v1 DB 클러스터를 프로비저닝된 클러스터로 변환을 참조하세요.	2023년 4월 27일
Aurora Serverless v1에서 Amazon Aurora PostgreSQL 버전 13 지원	이제 Aurora PostgreSQL 버전 13을 실행하는 Aurora Serverless v1 DB 클러스터를 생성할 수 있습니다. 자세한 내용은 Aurora Serverless v1 단원을 참조하십시오.	2023년 4월 27일
중국 리전에서 AWS Secrets Manager에 대해 Amazon Aurora 지원	Amazon Aurora가 중국(베이징) 및 중국(닝샤) 리전에서 Secrets Manager를 지원합니다. 자세한 내용은 Amazon RDS와 AWS Secrets Manager을 사용한 암호 관리를 참조하세요.	2023년 4월 20일
Amazon Aurora에서 주제 구독자에게 태그가 포함된 이벤트 게시 지원	Amazon Aurora에서 Amazon Simple Notification Service(SNS) 또는 Amazon EventBridge로 이벤트 알림을 전송할 경우, 이제 메시지 본문에 이벤트 태그가 포함됩니다. 이러한 태그는 서비스 이벤트의 영향을 받은 리소스 데이터를 제공합니다. 자세한 내용은 Amazon RDS 이벤트 알림 태그 및 속성을 참조하세요.	2023년 4월 17일

IAM 서비스 연결 역할 권한 업데이트	`AmazonRDSFullAccess` 및 `AmazonRDSReadOnlyAccess` 정책은 이제 RDS 콘솔에서 Amazon DevOps Guru 결과를 표시할 수 있는 추가 권한을 부여합니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2023년 3월 30일
아시아 태평양(멜버른) 리전에서 Amazon Aurora 글로벌 데이터베이스 지원	이제 아시아 태평양(멜버른) 리전에서 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 사용을 참조하세요.	2023년 3월 22일
AWS 관리형 정책 권한 업데이트	`AmazonRDSFullAccess` 및 `AmazonRDSReadOnlyAccess` 정책은 이제 Amazon CloudWatch에 추가 권한을 부여합니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2023년 3월 16일
중국 리전에서 RDS 프록시 사용 가능	이제 중국(베이징) 및 중국(닝샤) 리전에서 RDS 프록시를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2023년 3월 15일

중국 리전에서 Amazon Aurora 가 지원Aurora Serverless v2	이제 중국(베이징) 및 중국(닝샤) 리전에서 Aurora Serverless v2를 사용할 수 있습니다. 자세한 내용은 Aurora Serverless v2 단원을 참조하십시오.	2023년 3월 15일
아시아 태평양(자카르타) 리전에서 RDS 프록시 사용 가능	이제 아시아 태평양(자카르타) 리전에서 RDS 프록시를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2023년 3월 8일
Amazon Aurora MySQL에서 Kerberos 인증 지원	이제 사용자가 Aurora PostgreSQL DB 클러스터에 연결할 때 Kerberos 인증을 사용하여 사용자를 인증할 수 있습니다. 자세한 내용은 Aurora MySQL에서 Kerberos 인증 사용을 참조하세요.	2023년 3월 8일
추가 AWS 리전에서 Amazon Aurora 글로벌 데이터베이스 지원	이제 아프리카(케이프타운), 아시아 태평양(홍콩), 아시아 태평양(하이데라바드), 아시아 태평양(자카르타), 유럽(밀라노), 유럽(스페인), 유럽(취리히), 중동(바레인), 중동(UAE) 리전에서 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 사용을 참조하세요.	2023년 3월 6일

추가 AWS 리전에서 Amazon Aurora가 DB 클러스터 스냅샷 복사 지원	이제 아프리카(케이프타운), 아시아 태평양(홍콩), 아시아 태평양(하이데라바드), 아시아 태평양(자카르타), 아시아 태평양(멜버른), 유럽(밀라노), 유럽(스페인), 유럽(취리히), 중동(바레인), 중동(UAE) 리전에서 DB 클러스터를 복사할 수 있습니다. DB 클러스터 스냅샷 복사에 대한 자세한 내용은 DB 스냅샷 복사를 참조하세요.	2023년 3월 6일
Amazon DevOps Guru for RDS에서 사전 예방 인사이트 지원	Amazon DevOps Guru for RDS는 Aurora 데이터베이스에서 문제가 발생할 것으로 예측되기 전에 문제를 해결하는 데 도움이 되는 권장 사항과 함께 사전 예방 인사이트를 게시합니다. 자세한 내용은 DevOps Guru for RDS 작동 방식을 참조하세요.	2023년 2월 28일
Amazon Aurora MySQL 버전 1이 더 이상 사용되지 않음	Aurora MySQL 버전 1(MySQL 5.6과 호환)은 더 이상 사용되지 않습니다. 자세한 내용은 Amazon Aurora 메이저 버전을 사용할 수 있는 기간을 참조하세요.	2023년 2월 28일
Aurora Serverless v1에서 DB 클러스터 유지 관리 기간 설정 지원	이제 Aurora Serverless v1 DB 클러스터의 유지 관리 기간을 설정할 수 있습니다. 자세한 내용은 기본 DB 클러스터 유지 관리 기간 조정을 참조하세요.	2023년 2월 27일

Amazon Aurora는 아시아 태평양(하이데라바드), 유럽(스페인), 중동(UAE) 리전에서 데이터베이스 활동 스트림을 지원합니다.	자세한 내용은 데이터베이스 활동 스트림을 참조하세요.	2023년 1월 27일
아시아 태평양(멜버른) 리전에서 Amazon Aurora 사용 가능	이제 아시아 태평양(멜버른) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2023년 1월 23일
DB 클러스터 생성 중 인증 기관(CA) 지정	이제 DB 클러스터 생성 중에 DB 클러스터의 서버 인증서에 사용할 CA를 지정할 수 있습니다. 자세한 내용은 인증 기관을 참조하세요.	2023년 1월 5일
Aurora MySQL 3.*에서 역추적 지원	이제 Aurora MySQL 3.* 버전에서 잘못된 테이블이나 잘못된 행을 삭제하는 등의 사용자 실수를 빠르게 복구하는 기능을 제공합니다. 역추적을 사용하면 백업에서 복구하지 않고도 데이터베이스를 이전 시점으로 되돌릴 수 있습니다. 대용량 데이터베이스도 몇 초 안에 복구를 완료할 수 있습니다. 자세한 내용은 Aurora DB 클러스터 역추적을 참조하십시오.	2023년 1월 4일
추가 AWS 리전에서 Amazon RDS 블루/그린 배포 사용 가능	이제 중국(베이징) 및 중국(닝샤) 리전에서 블루/그린 배포 기능을 사용할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용을 참조하세요.	2022년 12월 22일

IAM 서비스 연결 역할 권한 업데이트	AmazonRDSServiceRolePolicy 정책은 이제 AWS Secrets Manager에 추가 권한을 부여합니다. 자세한 내용은 AWS 관리형 정책에 대한 Amazon RDS 업데이트를 참조하세요.	2022년 12월 22일
Amazon Aurora, 암호 관리를 위해 AWS Secrets Manager과 통합	Aurora는 Secrets Manager에서 DB 클러스터용 마스터 사용자 암호를 관리할 수 있습니다. 자세한 내용은 Amazon RDS와 AWS Secrets Manager을 사용한 암호 관리를 참조하세요.	2022년 12월 22일
Amazon Aurora가 추가 AWS 리전에서 Aurora Serverless v2 지원	Aurora Serverless v2는 이제 아프리카(케이프타운) 및 유럽(밀라노) 리전에서 사용할 수 있습니다. 자세한 내용은 Aurora Serverless v2 단원을 참조하십시오.	2022년 12월 21일
Aurora PostgreSQL에서 PostgreSQL 14가 있는 RDS 프록시 지원	이제 Aurora PostgreSQL 14 DB 클러스터가 있는 RDS Proxy를 생성할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2022년 12월 13일
Amazon Aurora는 Amazon DevOps Guru가 최근에 감지한 이상 징후를 사용자에게 통보합니다.	콘솔의 데이터베이스 세부 정보 페이지에는 현재 및 지난 24시간 동안 발생한 이상 현상이 모두 표시됩니다. 자세한 내용은 DevOps Guru for RDS 작동 방식을 참조하세요.	2022년 12월 13일

Amazon RDS 프록시에서 글로벌 데이터베이스 지원	이제 Aurora 글로벌 데이터베이스가 있는 RDS 프록시를 사용할 수 있습니다. 자세한 내용은 Aurora 글로벌 데이터베이스가 있는 RDS 프록시 사용을 참조하세요.	2022년 12월 7일
Aurora PostgreSQL DB 클러스터에서 PostgreSQL용 신뢰할 수 있는 언어 확장 지원	PostgreSQL용 신뢰할 수 있는 언어 확장은 고성능 PostgreSQL 확장을 구축하고 Aurora PostgreSQL DB 클러스터에서 안전하게 실행할 수 있는 오픈 소스 개발 키트입니다. 자세한 내용은 PostgreSQL용 신뢰할 수 있는 언어 확장 작업을 참조하세요.	2022년 11월 30일
액세스 위협을 모니터링하는 Amazon GuardDuty RDS Protection	GuardDuty RDS Protection을 켜면 GuardDuty는 Aurora 데이터베이스의 RDS 로그인 이벤트를 소비하고, 이러한 이벤트를 모니터링하고, 잠재적인 내부자 위협 또는 외부 행위자를 프로파일링합니다. GuardDuty RDS Protection에서 잠재적 위협을 탐지하면, GuardDuty는 손상되었을지도 모르는 데이터베이스 관련 세부 정보가 포함된 새 탐지 결과를 생성합니다. 자세한 내용은 GuardDuty RDS Protection을 이용한 위협 모니터링을 참조하세요.	2022년 11월 30일

데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용	스테이징 환경에서 프로덕션 DB 클러스터를 변경하고, DB 클러스터에 영향을 주지 않고 변경 사항을 테스트할 수 있습니다. 준비가 되면 다운타임을 최소화하여 스테이징 환경을 새로운 프로덕션 데이터베이스 환경으로 승격할 수 있습니다. 자세한 내용은 데이터베이스 업데이트에 Amazon RDS 블루/그린 배포 사용을 참조하세요.	2022년 11월 27일
아시아 태평양(하이데라바드) 리전에서 Amazon Aurora 사용 가능	이제 아시아 태평양(하이데라바드) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2022년 11월 22일
유럽(스페인) 리전에서 Amazon Aurora 사용 가능	이제 유럽(스페인) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2022년 11월 16일
유럽(취리히) 리전에서 Amazon Aurora 사용 가능	이제 유럽(취리히) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2022년 11월 9일
Amazon Aurora가 DB 클러스터에서 Amazon S3으로의 데이터 내보내기를 지원	이제 먼저 스냅샷을 생성하지 않고도 Aurora 클러스터 데이터를 S3로 바로 내보낼 수 있습니다. 자세한 내용은 Amazon S3으로 DB 클러스터 데이터 내보내기를 참조하세요.	2022년 10월 27일

Amazon Aurora MySQL이 Amazon S3로의 더 빠른 내보내기를 지원	이제 DB 클러스터 스냅샷 데이터를 MySQL 5.7 및 8.0과 호환되는 Aurora MySQL 클러스터용 S3로 내보내는 성능이 최대 10배 빨라졌습니다. 자세한 내용은 Amazon S3로 DB 클러스터 스냅샷 데이터 내보내기를 참조하세요.	2022년 10월 20일
Amazon Aurora가 Aurora DB 클러스터와 EC2 인스턴스의 연결 자동 설정을 지원함	AWS Management Console을 사용하여 기존 Aurora DB 클러스터와 EC2 인스턴스 간에 연결을 설정할 수 있습니다. 자세한 내용은 EC2 인스턴스와 Aurora DB 클러스터를 자동으로 연결을 참조하세요.	2022년 10월 14일
AWS JDBC Driver for PostgreSQL을 일반적으로 사용할 수 있게 됨	AWS JDBC Driver for PostgreSQL은 Aurora PostgreSQL을 위해 설계된 클라이언트 드라이버입니다. AWS JDBC Driver for PostgreSQL은 현재 일반적으로 사용 가능합니다. 자세한 내용은 AWS JDBC Driver for PostgreSQL에 연결을 참조하세요.	2022년 10월 6일
Amazon Aurora가 MySQL 5.7과 호환되는 Aurora MySQL에 대한 현재 위치 업그레이드를 지원함	기존 MySQL 5.7 호환 Aurora MySQL 클러스터에 현재 위치 업그레이드를 수행하여 MySQL 8.0 호환 Aurora MySQL 클러스터로 변경할 수 있습니다. 자세한 내용은 Aurora MySQL 2.x에서 3.x로 업그레이드를 참조하세요.	2022년 9월 26일

성능 개선 도우미에서 상위 25개의 SQL 쿼리를 보여줌	성능 개선 도우미 대시보드에서 상위 SQL 탭은 DB 로드에 가장 많은 기여하는 SQL 쿼리를 보여줍니다. 자세한 내용은 상위 SQL(Top SQL) 탭 개요를 참조하세요.	2022년 9월 13일
Aurora MySQL이 새로운 DB 인스턴스 클래스를 지원함	이제 Aurora MySQL DB 클러스터에 db.r6i DB 인스턴스 클래스를 사용할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스를 참조하세요.	2022년 9월 13일
중동(UAE) 리전에서 Amazon Aurora 사용 가능	이제 중동(UAE) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2022년 8월 30일
Amazon Aurora는 EC2 인스턴스와의 연결 자동 설정 지원	Aurora DB 클러스터를 생성할 때 AWS Management Console에서 Amazon Elastic Compute Cloud 인스턴스와 새 DB 클러스터 간의 연결을 설정할 수 있습니다. 자세한 내용은 EC2 인스턴스로 자동 네트워크 연결 구성 섹션을 참조하세요.	2022년 8월 22일
듀얼 스택 모드를 지원하는 Amazon Aurora	이제 DB 인스턴스를 듀얼 스택 모드로 실행할 수 있습니다. 듀얼 스택 모드에서 리소스는 IPv4, IPv6 또는 모두를 통해 DB 클러스터와 통신할 수 있습니다. 자세한 내용은 Amazon Aurora IP 주소 지정을 참조하세요.	2022년 8월 17일

PostgreSQL 호환 Aurora Serverless v1에 대한 실행 중 업그레이드를 지원하는 Amazon Aurora	PostgreSQL 10 호환 Aurora Serverless v1 클러스터에 대해 인플레이스 업그레이드를 수행하여 기존 클러스터를 PostgreSQL 11 호환 Aurora Serverless v1 클러스터로 변경할 수 있습니다. 실행 중 업그레이드 절차는 Aurora Serverless v1 DB 클러스터 수정을 참조하세요.	2022년 8월 8일
아시아 태평양(자카르타) 리전을 지원하는 성능 개선 도우미	이전에는 아시아 태평양(자카르타) 리전에서 성능 개선 도우미를 사용할 수 없었습니다. 이 제한 사항은 제거되었습니다. 자세한 내용은 성능 개선 도우미에 대한 AWS 리전 지원을 참조하세요.	2022년 7월 21일
새로운 DB 인스턴스 클래스를 지원하는 Amazon Aurora	이제 Aurora PostgreSQL DB 클러스터에 db.r6i DB 인스턴스 클래스를 사용할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스를 참조하세요.	2022년 7월 14일
추가 보존 기간을 지원하는 RDS 성능 개선 도우미	이전에 성능 개선 도우미는 7일(기본값) 또는 2년(731일)의 두 가지 보존 기간만 제공했습니다. 이제 실적 데이터를 7일 이상 보존해야 하는 경우 기간을 1~24개월로 지정할 수 있습니다. 자세한 내용은 성능 개선 도우미 위한 가격 및 데이터 보존을 참조하세요.	2022년 7월 1일

MySQL 호환 Aurora Serverless v1에 대한 실행 중 업그레이드를 지원하는 Amazon Aurora	MySQL 5.6 호환 Aurora Serverless v1 클러스터에 대해 인플레이스 업그레이드를 수행하여 기존 클러스터를 MySQL 5.7 호환 Aurora Serverless v1 클러스터로 변경할 수 있습니다. 실행 중 업그레이드 절차는 Aurora Serverless v1 DB 클러스터 수정을 참조하세요.	2022년 6월 16일
Aurora는 RDS 콘솔에서 Amazon DevOps Guru 사용을 지원합니다.	RDS 콘솔 내에서 Aurora 데이터베이스에 대한 DevOps Guru 적용 범위를 설정할 수 있습니다. 자세한 내용은 DevOps Guru for RDS 설정을 참조하세요.	2022년 6월 9일
Amazon Aurora는 암호화된 Amazon SNS 주제에 이벤트 게시를 지원합니다.	Amazon Aurora는 이제 서버 측 암호화(SSE)가 활성화된 Amazon Simple Notification Service(SNS) 주제에 이벤트를 게시하여 민감한 데이터를 전달하는 이벤트를 추가로 보호할 수 있습니다. 자세한 내용은 Amazon RDS 이벤트 알림 구독을 참조하세요.	2022년 6월 1일
Amazon RDS에서 Amazon CloudWatch에 사용 지표 게시	Amazon CloudWatch의 `AWS/Usage` 네임스페이스에는 Amazon RDS 서비스 할당량에 대한 계정 수준 사용 지표가 포함됩니다. 자세한 내용은 Amazon Aurora에 대한 Amazon CloudWatch 사용 지표 섹션을 참조하세요.	2022년 4월 28일

JSON 형식의 데이터 API 결과 세트	`ExecuteStatement` 함수에 대한 선택적 파라미터는 쿼리 결과 세트가 JSON 형식의 문자열로 반환되도록 합니다. JSON 결과 세트는 애플리케이션의 언어 데이터 구조로 변환하기가 쉽고 편리합니다. 자세한 내용은 JASON 형식으로 쿼리 결과 처리를 참조하세요.	2022년 4월 27일
Amazon Aurora Serverless v2가 이제 일반 공개되었습니다.	Amazon Aurora Serverless v2는 모든 사용자에게 제공됩니다. 자세한 내용은 Aurora Serverless v2 사용 섹션을 참조하세요.	2022년 4월 21일
Aurora MySQL에서 구성 가능한 암호 그룹 지원	Aurora MySQL을 사용하면 이제 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. 자세한 내용은 Aurora MySQL DB 클러스터 연결을 위한 암호 그룹 구성 섹션을 참조하세요.	2022년 4월 15일
Aurora PostgreSQL은 PostgreSQL 13으로 RDS Proxy 지원	이제 Aurora PostgreSQL 13 DB 클러스터를 사용하여 RDS Proxy를 생성할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2022년 4월 4일

Aurora PostgreSQL 릴리스 정보	이제 Amazon Aurora PostgreSQL 릴리스 정보에 대한 별도의 가이드가 제공됩니다. 자세한 내용은 Aurora PostgreSQL 릴리스 정보를 참조하세요.	2022년 3월 22일
Aurora MySQL 릴리스 정보	이제 Amazon Aurora MySQL 릴리스 정보에 대한 별도의 가이드가 제공됩니다. 자세한 내용은 Aurora MySQL 릴리스 정보를 참조하세요.	2022년 3월 22일
Aurora PostgreSQL은 다중 메이저 버전 업그레이드를 지원	이제 여러 주요 버전에서 Aurora PostgreSQL DB 클러스터의 버전 업그레이드를 수행할 수 있습니다. 자세한 내용은 메이저 버전 업그레이드 방법을 참조하세요.	2022년 3월 4일
Aurora PostgreSQL에서 구성 가능한 암호 그룹 지원	Aurora PostgreSQL 버전 11.8 이상에서는 구성 가능한 암호 그룹을 사용하여 데이터베이스 서버가 허용하는 연결 암호화를 제어할 수 있습니다. Aurora PostgreSQL에서 구성 가능한 암호 그룹 사용에 대한 자세한 내용은 Aurora PostgreSQL DB 클러스터에 대한 연결 암호 그룹 구성을 참조하세요.	2022년 3월 4일

일반적으로 사용 가능한 MySQL용 AWS JDBC 드라이버	AWS JDBC Driver for MySQL은 Aurora MySQL의 고가용성을 위해 설계된 클라이언트 드라이버입니다. MySQL용 AWS JDBC 드라이버는 현재 일반적으로 사용 가능합니다. 자세한 내용은 MySQL용 Amazon Web Services JDBC 드라이버로 연결을 참조하세요.	2022년 3월 2일
Aurora PostgreSQL 13.5에서 Aurora PostgreSQL 1.1.0을 위한 Babelfish 지원	Aurora PostgreSQL 13.5에서 Babelfish 1.1.0 지원 새 기능 목록은 13.5 섹션을 참조하세요. 각 Babelfish 릴리스에서 지원되는 기능 목록은 버전별 Babelfish에서 지원되는 기능 섹션을 참조하세요. 사용에 관한 자세한 내용은 Babelfish for Aurora PostgreSQL 작업을 참조하세요.	2022년 2월 28일
Amazon Aurora에서 아시아 태평양(자카르타) 리전의 데이터베이스 활동 스트림 지원	자세한 내용은 데이터베이스 활동 스트림에 대한 AWS 리전 지원을 참조하세요.	2022년 2월 16일
성능 개선 도우미에서 새로운 API 작업 지원	이제 성능 개선 도우미에서 `GetResourceMetadata` , `ListAvailableResourceDimensions` 및 `ListAvailableResourceMetrics` API 작업을 지원합니다. 자세한 내용은 이 설명서의 성능 개선 도우미 API를 사용하여 지표 검색과 Amazon RDS 성능 개선 도우미 API 레퍼런스를 참조하세요.	2022년 1월 12일

Amazon RDS 프록시에서 이벤트 지원	이제 RDS 프록시가 CloudWatch Events에서 구독하고 보거나 Amazon EventBridge로 전송하도록 구성할 수 있는 이벤트를 생성합니다. 자세한 내용은 RDS 프록시 이벤트 작업을 참조하세요.	2022년 1월 11일
추가 AWS 리전에서 RDS 프록시 사용 가능	이제 아프리카(케이프타운), 아시아 태평양(홍콩), 아시아 태평양(홍콩), 아시아 태평양(오사카), 유럽(밀라노), 유럽(파리), 유럽(파리), 유럽(스톡홀름), 중동(바레인) 및 남아메리카(상파울루) 리전에서 RDS 프록시를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 프록시 사용을 참조하세요.	2022년 1월 5일
아시아 태평양(자카르타) 리전에서 Amazon Aurora 사용 가능	이제 아시아 태평양(자카르타) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하세요.	2021년 12월 13일

DevOps Guru for Amazon RDS는 Amazon Aurora에 대한 자세한 인사이트 및 권장 사항 제공	DevOps Guru for RDS는 성능 관련 데이터에 대한 성능 개선 도우미를 추출합니다. 이 데이터를 사용하면 서비스는 Amazon Aurora DB 인스턴스의 성능을 분석하고 성능 문제를 해결하는 데 도움이 될 수 있습니다. 자세한 내용은 이 가이드의 DevOps Guru for RDS를 사용하여 성능 이상 분석 및 Amazon DevOps Guru 사용 설명서의 DevOps Guru for RDS 개요를 참조하세요.	2021년 12월 1일
Aurora PostgreSQL은 PostgreSQL 12로 RDS 프록시 지원	이제 Aurora PostgreSQL 12 데이터베이스 클러스터를 사용하여 RDS Proxy를 생성할 수 있습니다. RDS Proxy에 대한 자세한 내용은 Amazon RDS Proxy 사용을 참조하세요.	2021년 11월 22일
Aurora는 데이터베이스 활동 스트림에 대한 AWS Graviton2 인스턴스 클래스 지원	Aurora MySQL 및 Aurora PostgreSQL용 db.r6g 인스턴스 클래스를 통해 데이터베이스 활동 스트림을 사용할 수 있습니다. 자세한 내용은 지원되는 DB 인스턴스 클래스를 참조하세요.	2021년 11월 3일
Amazon Aurora는 계정 간 AWS KMS keys 지원	Amazon S3로 DB 스냅샷을 내보내는 경우 다른 AWS 계정의 KMS 키를 사용하여 암호화할 수 있습니다. 자세한 내용은 Amazon S3로 DB 스냅샷 데이터 내보내기를 참조하세요.	2021년 11월 3일

Amazon Aurora는 Babelfish for Aurora PostgreSQL 지원	Babelfish for Aurora PostgreSQL은 Microsoft SQL Server 클라이언트에서 데이터베이스 연결을 수락할 수 있는 기능을 사용하여 Amazon Aurora PostgreSQL 호환 버전 데이터베이스를 확장합니다. 자세한 내용은 Babelfish for Aurora PostgreSQL 작업을 참조하세요.	2021년 10월 28일
Aurora Serverless v1은 연결에 SSL을 요구할 수 있음	Aurora 클러스터 파라미터(PostgreSQL의 경우 `force_ssl` 이고 MySQL의 경우 `require_secure_transport`)는 Aurora Serverless v1에 대해 지원됩니다. 자세한 내용은 Aurora Serverless v1으로 TLS/SSL을 사용하는 방법을 참조하세요.	2021년 10월 26일
Amazon Aurora에서 추가 AWS 리전의 성능 개선 도우미 지원	성능 개선 도우미는 중동(바레인), 아프리카(케이프타운), 유럽(밀라노) 및 아시아 태평양(오사카) 리전에서 사용할 수 있습니다. 자세한 내용은 성능 개선 도우미에 대한 AWS 리전 지원을 참조하세요.	2021년 10월 5일

Aurora Serverless v1에 대한 구성 가능한 자동 크기 조정 시간 초과	Aurora Serverless v1이 자동 크기 조정 지점을 찾기 위해 얼마나 오래 대기할지 선택할 수 있습니다. 해당 기간 동안 자동 크기 조정 지점을 찾을 수 없는 경우 Aurora Serverless v1은 선택한 시간 초과 작업에 따라 확장 이벤트를 취소하거나 용량 변경을 강제로 수행합니다. 자세한 내용은 Aurora Serverless v1에 대한 자동 크기 조정을 참조하세요.	2021년 9월 10일
Aurora에서 X2g 및 T4g 인스턴스 클래스 지원	이제 Aurora MySQL과 Aurora PostgreSQL은 모두 X2g 및 T4g 인스턴스 클래스를 사용할 수 있습니다. 사용할 수 있는 인스턴스 클래스는 Aurora MySQL 또는 Aurora PostgreSQL SQL의 버전에 따라 다릅니다. 지원되는 인스턴스 유형에 대한 자세한 내용은 DB 인스턴스 클래스를 참조하세요.	2021년 9월 10일
Amazon RDS, 공유 VPC에서 RDS 프록시 지원	이제 공유 Virtual Private Cloud(VPC)에 RDS 프록시를 만들 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 사용 설명서 또는 Aurora 사용 설명서에서 'Amazon RDS 프록시를 사용한 연결 관리'를 참조하세요.	2021년 8월 6일

Aurora 버전 정책 페이지	Amazon Aurora 사용 설명서에는 Aurora 버전 및 관련 정책에 대한 일반 정보를 담은 섹션이 포함되어 있습니다. 자세한 내용은 Amazon Aurora 버전을 참조하세요.	2021년 7월 14일
AWS CloudTrail 추적에서 데이터 API 이벤트 제외	CloudTrail 추적에서 데이터 API 이벤트를 제외할 수 있습니다. 자세한 내용은 AWS CloudTrail 추적에서 데이터 API 이벤트 제외를 참조하세요.	2021년 7월 2일
Amazon Aurora PostgreSQL 호환 버전에서 추가 확장 지원	새로 지원되는 확장에는 pg_bigm, pg_cron, pg_partman 및 pg_proctab이 포함됩니다. 자세한 내용은 Amazon Aurora PostgreSQL 호환 버전에 대한 확장 버전을 참조하세요.	2021년 6월 17일
Aurora Serverless 클러스터에 대한 복제	이제 Aurora Serverless인 복제된 클러스터를 생성할 수 있습니다. 복제에 대한 자세한 내용은 Aurora DB 클러스터 볼륨 복제를 참조하세요.	2021년 6월 16일
중국(베이징) 및 중국(닝샤) 리전에서 Aurora 글로벌 데이터베이스 사용 가능	이제 중국(베이징) 및 중국(닝샤) 리전에서 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 작업을 참조하십시오.	2021년 5월 19일

데이터 API에 대한 FIPS 140-2 지원	Data API는 SSL/TLS 연결에 대한 Federal Information Processing Standard Publication 140-2(FIPS 140-2)를 지원합니다. 자세한 내용은 Data API 가용성을 참조하세요.	2021년 5월 14일
AWS JDBC Driver for PostgreSQL(미리 보기)	현재 미리 보기로 제공되는 AWS JDBC Driver for PostgreSQL은 Aurora PostgreSQL의 고가용성을 위해 설계된 클라이언트 드라이버입니다. 자세한 내용은 PostgreSQL용 Amazon Web Services JDBC 드라이버 연결(평가판)을 참조하세요.	2021년 4월 27일
추가 AWS 리전에서 데이터 API 사용 가능	이제 아시아 태평양(서울) 및 캐나다(중부) 리전에서 데이터 API를 사용할 수 있습니다. 자세한 내용은 데이터 API 가용성을 참조하십시오.	2021년 4월 9일
Amazon Aurora에서 Graviton2 DB 인스턴스 클래스 지원	이제 Graviton2 DB 인스턴스 클래스 db.r6g.x를 사용하여 MySQL 또는 PostgreSQL을 실행하는 DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 유형 단원을 참조하십시오.	2021년 3월 12일

RDS 프록시 엔드포인트 개선 사항	각 RDS 프록시와 연결된 추가 엔드포인트를 생성할 수 있습니다. 다른 VPC에 엔드포인트를 생성하면 프록시에 대한 VPC 간 액세스가 가능합니다. Aurora MySQL 클러스터용 프록시에는 읽기 전용 엔드포인트가 있을 수도 있습니다. 이러한 리더 엔드포인트는 클러스터의 읽기 DB 인스턴스에 연결되며 쿼리 집약적 애플리케이션의 읽기 확장성과 가용성을 높일 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 사용 설명서 또는 Aurora 사용 설명서의 "Amazon RDS 프록시를 사용한 연결 관리"를 참조하세요.	2021년 3월 8일
아시아 태평양(오사카) 리전에서 Amazon Aurora 사용 가능	이제 아시아 태평양(오사카) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하십시오.	2021년 3월 1일
Aurora PostgreSQL, 동일한 DB 클러스터에 IAM 인증과 Kerberos 인증을 모두 사용하도록 지원	이제 Aurora PostgreSQL가 동일한 DB 클러스터에 IAM 인증과 Kerberos 인증을 모두 사용하도록 지원합니다. 자세한 내용은 Amazon Aurora를 사용한 데이터베이스 인증을 참조하세요.	2021년 2월 24일

Aurora 글로벌 데이터베이스에서 이제 계획된 관리형 장애 조치 지원	Aurora 글로벌 데이터베이스가 이제 계획된 관리형 장애 조치를 지원하므로 Aurora 글로벌 데이터베이스의 기본 AWS 리전을 보다 쉽게 변경할 수 있습니다. 정상적인 Aurora 전역 데이터베이스에서만 계획된 관리형 장애 조치를 사용할 수 있습니다. 자세한 내용은 재해 복구 및 Amazon Aurora 전역 데이터베이스를 참조하세요. 참조 정보는 Amazon RDS API Reference의 `FailoverGlobalCluster` 단원을 참조하세요.	2021년 2월 11일
이제 Aurora Serverless용 Data API에서 더 많은 데이터 형식 지원	Aurora Serverless용 Data API를 사용하면 `UUID` 및 `JSON` 데이터 형식을 데이터베이스에 대한 입력으로 사용할 수 있습니다. 또한 Aurora Serverless용 Data API를 사용하면 데이터베이스에서 `LONG` 유형 값을 `STRING` 값으로 반환할 수 있습니다. 자세한 내용은 Data API 호출을 참조하세요. 지원되는 데이터 형식에 대한 참조 정보는 Amazon RDS Data Service API 참조의 `SqlParameter` 단원을 참조하세요.	2021년 2월 2일

Aurora PostgreSQL에서 PostgreSQL 버전 12로의 메이저 버전 업그레이드 지원	Aurora PostgreSQL을 사용하면 이제 DB 엔진을 메이저 버전 12로 업그레이드할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에 대한 PostgreSQL DB 엔진 업그레이드를 참조하십시오.	2021년 1월 28일
Aurora MySQL에서 인플레이스 업그레이드 지원	기존 클러스터의 DB 인스턴스, 엔드포인트 등을 보존하여 Aurora MySQL 1.x 클러스터를 Aurora MySQL 2.x로 업그레이드할 수 있습니다. 이 현재 위치 업그레이드 기술을 사용하면 스냅샷을 복원하여 완전히 새로운 클러스터를 설정하는 불편함을 피할 수 있습니다. 또한 모든 테이블 데이터를 새 클러스터로 복사하는 오버헤드를 피할 수 있습니다. 자세한 내용은 Aurora MySQLDB 클러스터의 주 버전을 1.x에서 2.x로 업그레이드를참조하십시오.	2021년 1월 11일
AWS JDBC Driver for MySQL(미리 보기)	현재 미리 보기로 제공되는 MySQL용 AWS JDBC 드라이버는 Aurora MySQL의 고가용성을 위해 설계된 클라이언트 드라이버입니다. 자세한 내용은 MySQL용 Amazon Web Services JDBC 드라이버로 연결(평가판)을 참조하세요.	2021년 1월 7일

Aurora, 전역 데이터베이스의 보조 클러스터에서 데이터베이스 활동 스트림 지원	Aurora PostgreSQL 또는 Aurora MySQL의 주 또는 보조 클러스터에서 데이터베이스 활동 스트림으로 데이터베이스를 시작할 수 있습니다. 지원되는 엔진 버전은 Aurora 전역 데이터베이스의 제한 사항을 참조하십시오.	2020년 12월 22일
DB 인스턴스가 4개 있는 멀티 마스터 클러스터	Aurora MySQL 멀티 마스터 클러스터의 최대 DB 인스턴스 수는 이제 4개입니다. 이전에는 최대 DB 인스턴스 수가 두 개였습니다. 자세한 내용은 Aurora 멀티 마스터 클러스터 작업을 참조하십시오.	2020년 12월 17일
Aurora PostgreSQL에서 AWS Lambda 함수 지원	이제 Aurora PostgreSQL DB 클러스터에 대해 AWS Lambda 함수를 호출할 수 있습니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터에서 Lambda 함수 호출을 참조하세요.	2020년 12월 11일
Amazon Aurora에서 Graviton2 DB 인스턴스 클래스를 미리 보기로 지원	이제 미리 보기에서 Graviton2 DB 인스턴스 클래스 db.r6g.x를 사용하여 MySQL 또는 PostgreSQL을 실행하는 DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 유형 단원을 참조하십시오.	2020년 12월 11일

이제 미리 보기에서 Amazon Aurora Serverless v2를 사용할 수 있습니다.	미리 보기에서 Amazon Aurora Serverless v2을 사용할 수 있습니다. Amazon Aurora Serverless v2로 작업하려면 액세스를 신청해야 합니다. 자세한 내용은 Aurora Serverless v2 페이지를 참조하세요.	2020년 12월 1일

이제 Aurora Serverless에 대해 Aurora PostgreSQL을 더 많은 AWS 리전에서 사용할 수 있습니다.	이제 Aurora Serverless에 대해 Aurora PostgreSQL을 더 많은 AWS 리전에서 사용할 수 있습니다. 이제 Aurora MySQL Serverless v1을 제공하는 동일한 AWS 리전에서 Aurora PostgreSQL Serverless v1을 실행하도록 선택할 수 있습니다. Aurora Serverless를 지원하는 추가 AWS 리전은 미국 서부(캘리포니아 북부), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(서울), 아시아 태평양(뭄바이), 캐나다(중부), 유럽(런던) 및 유럽(파리)입니다. 모든 리전 및 Aurora Serverless를 지원하는 Aurora DB 엔진 목록은 Aurora Serverless v1을 지원하는 리전 및 Aurora DB 엔진을 참조하세요. Aurora Serverless용 Amazon RDS 데이터 API는 다음과 같은 동일한 AWS 리전에서도 사용할 수 있습니다. Aurora Serverless에 대한 데이터 API를 지원하는 모든 리전 목록은 Aurora MySQL Serverless v1을 사용하는 Data API를 참조하세요.	2020년 11월 24일

Amazon RDS 성능 개선 도우미, 새로운 차원 도입	데이터베이스, 애플리케이션(PostgreSQL) 및 세션 유형(PostgreSQL)에 대한 차원 그룹에 따라 데이터베이스 로드를 그룹화할 수 있습니다. Amazon RDS는 또한 차원 db.name, db.application.name(PostgreSQL) 및 db.session_type.name(PostgreSQL)을 지원합니다. 자세한 내용은 상위 부하 테이블을 참조하세요.	2020년 11월 24일
Aurora Serverless은 Aurora PostgreSQL 버전 10.12 지원	Aurora Serverless용 Aurora PostgreSQL은 AWS 리전에 걸쳐 Aurora PostgreSQL 버전 10.12로 업그레이드되었습니다. 리전에서는 Aurora Serverless용 Aurora PostgreSQL이 지원됩니다. 자세한 내용은 Aurora Serverless v1을 지원하는 리전 및 Aurora DB 엔진을 참조하세요.	2020년 11월 4일
이제 데이터 API에서 태그 기반 권한 부여 지원	데이터 API는 태그 기반 권한 부여를 지원합니다. RDS 클러스터 리소스에 태그를 레이블을 지정한 경우 정책 문에서 이러한 태그를 사용하여 데이터 API를 통해 액세스를 제어할 수 있습니다. 자세한 내용은 데이터 API에 대한 액세스 권한 부여를 참조하십시오.	2020년 10월 27일

Amazon Aurora에서 스냅샷 내보내기에 대한 지원을 Amzon S3로 확장	이제 모든 상용 AWS 리전에서 DB 스냅샷 데이터를 Amazon S3로 내보낼 수 있습니다. 자세한 내용은 Amazon S3으로 DB 스냅샷 데이터 내보내기를 참조하세요.	2020년 10월 22일
Aurora 글로벌 데이터베이스에서 복제 지원	이제 Aurora 글로벌 데이터베이스의 기본 및 보조 DB 클러스터에 대한 복제본을 생성할 수 있습니다. 이 작업을 수행하려면 AWS Management Console을 사용하여 복제본 생성(Create clone) 메뉴 옵션을 선택합니다. 또는 AWS CLI를 사용하여 `restore-db-cluster-to-point-in-time` 옵션과 함께 `--restore-type copy-on-write` 명령을 실행해도 됩니다. AWS Management Console 또는 AWS CLI를 사용하여 전체 AWS 계정의 Aurora 글로벌 데이터베이스에서 DB 클러스터를 복제할 수도 있습니다. 복제에 대한 자세한 내용은 Aurora DB 클러스터 볼륨 복제를 참조하세요.	2020년 10월 19일

Amazon Aurora에서 클러스터 볼륨의 동적 크기 조정 지원	Aurora MySQL 1.23 및 2.09, Aurora PostgreSQL 3.3.0 및 Aurora PostgreSQL 2.6.0부터는 `DROP TABLE`과 같은 작업을 통해 데이터를 제거하면 Aurora가 클러스터 볼륨의 크기를 줄입니다. 이 향상된 기능을 활용하려면 클러스터에서 사용하는 데이터베이스 엔진에 따라 적절한 버전 중 하나로 업그레이드합니다. 이 기능과 Aurora 클러스터에 사용된 스토리지 공간 및 사용 가능한 스토리지 공간을 확인하는 방법에 대한 자세한 내용은 Aurora DB 클러스터의 성능 및 조정 관리를 참조하세요.	2020년 10월 13일
Amazon Aurora에서 최대 128TiB까지 볼륨 크기 지원	이제 신규 및 기존 Aurora 클러스터 볼륨이 최대 128 tebibytes (TiB) 크기까지 확장될 수 있습니다. 자세한 내용은 Aurora 스토리지 증가 방법을 참조하세요.	2020년 9월 22일
Aurora PostgreSQL, 중국(닝샤) 리전에서 db.r5 및 db.t3 DB 인스턴스 클래스 지원	이제 db.r5 및 db.t3 DB 인스턴스 클래스를 사용하는 중국(닝샤) 리전에서 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스를 참조하십시오.	2020년 9월 3일

Aurora 병렬 쿼리 향상	Aurora MySQL 2.09 및 1.23부터는 병렬 쿼리 기능의 개선 사항을 활용할 수 있습니다. 병렬 쿼리 클러스터를 생성하기 위해 더 이상 특수 엔진 모드가 필요하지 않습니다. 이제 호환 가능한 Aurora MySQL 버전을 실행 중인 프로비저닝된 클러스터에 대해 `aurora_parallel_query` 구성 옵션을 사용하여 병렬 쿼리를 설정하거나 해제할 수 있습니다. 새 클러스터를 생성하고 데이터를 해당 클러스터로 가져오는 대신 기존 클러스터를 호환 가능한 Aurora MySQL 버전으로 업그레이드하고 병렬 쿼리를 사용할 수 있습니다. 병렬 쿼리 클러스터에 성능 개선 도우미를 사용할 수 있습니다. 병렬 쿼리 클러스터를 중지하고 시작할 수 있습니다. MySQL 5.7과 호환되는 Aurora 병렬 쿼리 클러스터를 생성할 수 있습니다. 병렬 쿼리는 `DYNAMIC` 행 형식을 사용하는 테이블에 대해 작동합니다. 병렬 쿼리 클러스터는 AWS Identity and Access Management(IAM) 인증을 사용할 수 있습니다. 병렬 쿼리 클러스터의 리더 DB 인스턴스는 `READ COMMITTED` 격리 수준을 활용할 수 있습니다. 이제 추가 AWS 리전에서 병렬 쿼리 클러스터를 생성할 수도 있습니다. 병렬 쿼리 기능과 이러한 개	2020년 9월 2일

	선 사항에 대한 자세한 내용은 Aurora MySQL용 병렬 쿼리를 참조하세요.	
Aurora MySQL 파라미터 binlog_rows_query_log_events 변경 사항	이제 Aurora MySQL 구성 파라미터 `binlog_rows_query_log_events` 의 값을 변경할 수 있습니다. 이전에는 이 파라미터를 수정할 수 없었습니다.	2020년 8월 26일
Aurora MySQL의 마이너 버전 자동 업그레이드 지원	이제 Aurora MySQL에서 Aurora MySQL DB 클러스터에 대해 마이너 버전 자동 업그레이드 사용(Enable auto minor version upgrade) 설정을 지정하면 설정이 적용됩니다. 마이너 버전 자동 업그레이드를 활성화하면 새 마이너 버전이 릴리스될 때 Aurora가 자동으로 새 마이너 버전으로 업그레이드됩니다. 자동 업그레이드는 데이터베이스의 유지 관리 기간 동안 발생합니다. Aurora MySQL의 경우 이 기능은 MySQL 5.7과 호환되는 Aurora MySQL 버전 2에만 적용됩니다. 처음에는 자동 업그레이드 절차를 통해 Aurora MySQL DB 클러스터가 버전 2.07.2로 업그레이드됩니다. Aurora MySQL에서 이 기능을 사용하는 방식에 대한 자세한 내용은 Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치를 참조하세요.	2020년 8월 3일

Aurora PostgreSQL에서 PostgreSQL 버전 11로의 메이저 버전 업그레이드 지원	Aurora PostgreSQL을 사용하면 이제 DB 엔진을 메이저 버전 11로 업그레이드할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에 대한 PostgreSQL DB 엔진 업그레이드를 참조하십시오.	2020년 7월 28일
Amazon Aurora에서 AWS PrivateLink 지원	이제 Amazon Aurora는 AWS 네트워크의 Aurora와 애플리케이션 간의 트래픽을 유지하기 위해 Amazon RDS API 호출을 위한 Amazon VPC 엔드포인트 생성을 지원합니다. 자세한 내용은 Amazon Aurora 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.	2020년 7월 9일
RDS 프록시 정식으로 사용 가능	이제 RDS 프록시를 정식으로 사용할 수 있습니다. RDS 프록시는 프로덕션 워크로드용 RDS for MySQL, Aurora MySQL, RDS for PostgreSQL 및 Aurora PostgreSQL에 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS 사용 설명서 또는 Aurora 사용 설명서의 “Amazon RDS 프록시를 사용한 연결 관리”를 참조하세요.	2020년 6월 30일

Aurora 글로벌 데이터베이스 쓰기 전달	이제 글로벌 데이터베이스의 보조 클러스터에서 쓰기 기능을 활성화할 수 있습니다. 쓰기 전달을 사용하여 보조 클러스터에서 DML 문을 실행하고, Aurora가 기본 클러스터에 쓰기를 전달하며, 업데이트된 데이터가 모든 보조 클러스터에 복제됩니다. 자세한 내용은 Aurora 글로벌 데이터베이스를 사용한 보조 AWS 리전의 쓰기 전달을 참조하세요.	2020년 6월 18일
Aurora에서 AWS Backup와의 통합 지원	AWS Backup을 사용하여 Aurora DB 클러스터의 백업을 관리할 수 있습니다. 자세한 내용은 Aurora DB 클러스터 백업 및 복원에 대한 개요를 참조하십시오.	2020년 6월 10일
Aurora PostgreSQL에서 db.t3.large DB 인스턴스 클래스 지원	이제 db.t3.large DB 인스턴스 클래스를 사용하는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스를 참조하십시오.	2020년 6월 5일

Aurora 글로벌 데이터베이스, PostgreSQL 버전 11.7 및 관리형 복구 시점 목표(RPO) 지원	이제 PostgreSQL 데이터베이스 엔진 버전 11.7에 대한 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. 또한 RPO(복구 지점 목표)를 사용하여 PostgreSQL 글로벌 데이터베이스가 실패로부터 복구하는 방법을 관리할 수 있습니다. 자세한 내용은 Aurora 글로벌 데이터베이스에 대한 리전 간 재해 복구를 참조하십시오.	2020년 6월 4일
Aurora MySQL에서 데이터베이스 활동 스트림으로 데이터베이스 모니터링 지원	이제 Aurora MySQL에 데이터베이스 활동 스트림이 포함되어, 관계형 데이터베이스에서 데이터베이스 활동의 데이터 스트림을 거의 실시간으로 제공하게 됩니다. 자세한 내용은 데이터베이스 활동 스트림 사용을 참조하십시오.	2020년 6월 2일
추가 AWS 리전에서 쿼리 편집기 사용 가능	이제 추가 AWS 리전에서 Aurora Serverless 쿼리 편집기를 사용할 수 있습니다. 자세한 내용은 쿼리 편집기 가용성을 참조하세요.	2020년 5월 28일
추가 AWS 리전에서 데이터 API 사용 가능	이제 추가 AWS 리전에서 데이터 API를 사용할 수 있습니다. 자세한 내용은 데이터 API 가용성을 참조하세요.	2020년 5월 28일

캐나다(중부) 리전에서 RDS 프록시 사용 가능	이제 캐나다(중부) 리전에서 RDS 프록시 미리 보기를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy(미리 보기)를 사용한 연결 관리를 참조하십시오.	2020년 5월 28일
Aurora 글로벌 데이터베이스 및 리전 간 읽기 전용 복제본	Aurora 글로벌 데이터베이스의 경우 이제 보조 클러스터와 동일한 리전에는 기본 클러스터의 Aurora MySQL 리전 간 읽기 전용 복제본을 생성할 수 있습니다. Aurora 글로벌 데이터베이스 및 리전 간 읽기 전용 복제본에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 작업 및 Amazon Aurora MySQL DB 복제를 참조하십시오.	2020년 5월 18일
더 많은 AWS 리전에서 RDS 프록시 사용 가능	이제 미국 서부(캘리포니아 북부) 리전, 유럽(런던) 리전, 유럽(프랑크푸르트) 리전, 아시아 태평양(서울) 리전, 아시아 태평양(뭄바이) 리전, 아시아 태평양(싱가포르) 리전 및 아시아 태평양(시드니) 리전에서 RDS 프록시 미리 보기를 사용할 수 있습니다. RDS 프록시에 대한 자세한 내용은 Amazon RDS Proxy(미리 보기)를 사용한 연결 관리를 참조하십시오.	2020년 5월 13일

Aurora PostgreSQL 호환 버전, 온프레미스 또는 자체 호스팅된 Microsoft Active Directory 지원	이제 사용자가 Aurora PostgreSQL DB 클러스터에 연결할 때 사용자의 Kerberos 인증을 위해 온프레미스 또는 셀프 호스팅된 Active Directory를 사용할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에서 Kerberos 인증 사용을 참조하십시오.	2020년 5월 7일
더 많은 AWS 리전에서 Aurora MySQL 멀티 마스터 클러스터 사용 가능	이제 아시아 태평양(서울) 리전, 아시아 태평양(도쿄) 리전, 아시아 태평양(뭄바이) 리전 및 유럽(프랑크푸르트) 리전에서 Aurora 멀티 마스터 클러스터를 생성할 수 있습니다. 멀티 마스터 클러스터에 대한 자세한 내용은 Aurora 멀티 마스터 클러스터 작업을 참조하십시오.	2020년 5월 7일
성능 개선 도우미에서 실행 중인 Aurora MySQL 쿼리에 대한 통계 분석 지원	이제 Aurora MySQL DB 인스턴스용 성능 개선 도우미를 사용하여 실행 중인 쿼리의 통계를 분석할 수 있습니다. 자세한 내용은 실행 중인 쿼리에 대한 통계 분석을 참조하십시오.	2020년 5월 5일

데이터 API용 Java 클라이언트 라이브러리 정식으로 사용 가능	이제 데이터 API용 Java 클라이언트 라이브러리를 정식으로 사용할 수 있습니다. 데이터 API용 Java 클라이언트 라이브러리를 다운로드하여 사용할 수 있습니다. 이를 통해 클라이언트 측 클래스를 데이터 API의 요청 및 응답에 매핑할 수 있습니다. 자세한 내용은 데이터 API용 Java 클라이언트 라이브러리 사용을 참조하십시오.	2020년 4월 30일
유럽(밀라노) 리전에서 Amazon Aurora 사용 가능	이제 유럽(밀라노) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하십시오.	2020년 4월 28일
유럽(밀라노) 리전에서 Amazon Aurora 사용 가능	이제 유럽(밀라노) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하십시오.	2020년 4월 27일
아프리카(케이프타운) 리전에서 Amazon Aurora 사용 가능	이제 아프리카(케이프타운) 리전에서 Amazon Aurora를 사용할 수 있습니다. 자세한 내용은 리전 및 가용 영역을 참조하십시오.	2020년 4월 22일
이제 Aurora PostgreSQL에서 db.r5.16xlarge 및 db.r5.8xlarge DB 인스턴스 클래스 지원	이제 db.r5.16xlarge 및 db.r5.8xlarge DB 인스턴스 클래스를 사용하는 PostgreSQL을 실행하는 Aurora PostgreSQL DB 인스턴스를 생성할 수 있습니다. 자세한 내용은 Aurora에 대한 DB 인스턴스 클래스의 하드웨어 사양을 참조하십시오.	2020년 4월 8일

Amazon RDS for PostgreSQL 프록시	이제 PostgreSQL에서 Amazon RDS 프록시를 사용할 수 있습니다. RDS Proxy를 사용하면 클러스터에서 연결 관리의 오버헤드를 줄이고 "연결이 너무 많음" 오류의 가능성을 줄일 수 있습니다. RDS Proxy는 현재 PostgreSQL에 대한 공개 미리 보기 상태입니다. 자세한 내용은 Amazon RDS Proxy(미리 보기)를 사용한 연결 관리를 참조하십시오.	2020년 4월 8일
이제 Aurora 글로벌 데이터베이스에서 Aurora PostgreSQL 지원	이제 PostgreSQL 데이터베이스 엔진에 대한 Aurora Global Database를 생성할 수 있습니다. Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있어, 대기 시간이 짧은 글로벌 읽기와 리전 전체의 중단으로부터 재해 복구를 지원합니다. 자세한 내용은 Amazon Aurora Global Database 작업을 참조하세요.	2020년 3월 10일
Aurora PostgreSQL에 대한 메이저 버전 업그레이드 지원	Aurora PostgreSQL을 사용하면 이제 DB 엔진을 메이저 버전으로 업그레이드할 수 있습니다. 이렇게 하면 PostgreSQL 엔진 버전 선택을 업그레이드할 때 최신 메이저 버전으로 건너뛸 수 있습니다. 자세한 내용은 Aurora PostgreSQL에 대한 PostgreSQL DB 엔진 업그레이드를 참조하십시오.	2020년 3월 4일

Aurora PostgreSQL에서 Kerberos 인증 지원	이제 사용자가 Aurora PostgreSQL DB 클러스터에 연결할 때 Kerberos 인증을 사용하여 사용자를 인증할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에서 Kerberos 인증 사용을 참조하십시오.	2020년 2월 28일
데이터 API에서 AWS PrivateLink 지원	이제 데이터 API는 AWS 네트워크의 데이터 API와 애플리케이션 간의 트래픽을 유지하기 위해 데이터 API 호출을 위한 Amazon VPC 엔드포인트 생성을 지원합니다. 자세한 내용은 데이터 API에 대한 Amazon VPC 엔드포인트(AWS PrivateLink) 생성을 참조하세요.	2020년 2월 6일
Aurora PostgreSQL에서 Aurora 기계 학습 지원	`aws_ml` Aurora PostgreSQL 확장은 데이터베이스 쿼리에서 감성 분석을 위해 Amazon Comprehend를 호출하고 자체 기계 학습 모델을 실행하기 위해 SageMaker를 호출하는 데 사용하는 함수를 제공합니다. 자세한 내용은 Aurora에서 기계 학습(ML) 기능 사용을 참조하십시오.	2020년 2월 5일

Aurora PostgreSQL에서 Amazon S3로의 데이터 내보내기 지원	Aurora PostgreSQL DB 클러스터에서 데이터를 쿼리하여 Amazon S3 버킷에 저장된 파일로 직접 내보낼 수 있습니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터에서 Amazon S3로 데이터 내보내기를 참조하십시오.	2020년 2월 5일
DB 스냅샷 데이터를 Amazon S3에 내보내기 지원	Amazon Aurora는 MySQL 및 PostgreSQL에서 Amazon S3로 기존의 DB 스냅샷 데이터를 내보낼 수 있도록 지원합니다. 자세한 내용은 Amazon S3로 DB 스냅샷 데이터 내보내기를 참조하십시오.	2020년 1월 9일
문서 이력의 Aurora MySQL 릴리스 정보	이 단원에는 2018년 8월 31일 이후 출시된 버전의 Aurora MySQL 호환 버전 릴리스 정보에 대한 기록 항목이 포함되어 있습니다. 특정 버전의 전체 릴리스 정보를 보려면 기록 항목의 첫 번째 열에 있는 링크를 선택합니다.	2019년 12월 13일

Amazon RDS 프록시	Amazon RDS Proxy를 사용하면 클러스터에서 연결 관리의 오버헤드를 줄이고 "연결이 너무 많음" 오류의 가능성을 줄일 수 있습니다. 각 프록시를 RDS DB 인스턴스 또는 Aurora DB 클러스터와 연결합니다. 그런 다음 애플리케이션의 연결 문자열에 프록시 엔드포인트를 사용합니다. Amazon RDS Proxy는 현재 공개 미리 보기 상태입니다. 이것은 Aurora MySQL 데이터베이스 엔진을 지원합니다. 자세한 내용은 Amazon RDS Proxy(미리 보기)를 사용한 연결 관리를 참조하십시오.	2019년 12월 3일
Aurora Serverless v1용 데이터 API에서 데이터 형식 매핑 힌트 지원	이제 힌트를 사용하여 Aurora Serverless v1 값을 다른 형식으로 데이터베이스에 전송하도록 `String`용 데이터 API에 지시할 수 있습니다. 자세한 내용은 데이터 API 호출을 참조하십시오.	2019년 11월 26일
Aurora Serverless v1용 데이터 API에서 Java 클라이언트 라이브러리(평가판) 지원	데이터 API용 Java 클라이언트 라이브러리를 다운로드하여 사용할 수 있습니다. 이를 통해 클라이언트 측 클래스를 데이터 API의 요청 및 응답에 매핑할 수 있습니다. 자세한 내용은 데이터 API용 Java 클라이언트 라이브러리 사용을 참조하십시오.	2019년 11월 26일

Aurora PostgreSQL에서 FedRAMP HIGH 사용 가능	Aurora PostgreSQL은 FedRAMP HIGH 사용 자격이 있습니다. AWS 및 규정 준수 활동에 대한 자세한 내용은 규정 준수 프로그램 제공 AWS 범위 내 서비스 섹션을 참조하세요.	2019년 11월 26일
Amazon Aurora MySQL 복제본에 대해 READ COMMITTED 격리 수준 활성화됨	이제 Aurora MySQL 복제본에서 `READ COMMITTED` 격리 수준을 활성화할 수 있습니다. 이렇게 하려면 세션 수준에서 `aurora_read_replica_read_committed_isolation_enabled` 구성 설정을 활성화해야 합니다. OLTP 클러스터의 장기 실행 쿼리에 `READ COMMITTED` 격리 수준을 사용하면 기록 목록 길이와 관련된 문제를 해결하는 데 도움이 됩니다. 이 설정을 활성화하기 전에 Aurora 복제본의 격리 동작이 `READ COMMITTED` 의 평상시 MySQL 구현과 어떻게 다른지 이해해야 합니다. 자세한 내용은 Aurora MySQL 격리 수준을 참조하십시오.	2019년 11월 25일
성능 개선 도우미에서 실행 중인 Aurora PostgreSQL 쿼리에 대한 통계 분석 지원	이제 Aurora PostgreSQL DB 인스턴스용 성능 개선 도우미를 사용하여 실행 중인 쿼리의 통계를 분석할 수 있습니다. 자세한 내용은 실행 중인 쿼리에 대한 통계 분석을 참조하십시오.	2019년 11월 25일

Aurora 글로벌 데이터베이스에 더 많은 클러스터 지원	이제 여러 개의 보조 리전을 Aurora 글로벌 데이터베이스에 추가할 수 있습니다. 더 넓은 지리 영역에 걸쳐 짧은 지연 시간 전역 읽기 및 재해 복구를 이용할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 작업을 참조하십시오.	2019년 11월 25일
Aurora MySQL에서 Aurora 기계 학습 지원	Aurora MySQL 2.07 이상에서 감성 분석에 대해 Amazon Comprehend를, 광범위한 기계 학습 알고리즘에 대해 SageMaker를 호출할 수 있습니다. 쿼리의 저장 함수에 대한 호출을 포함하여 데이터베이스 애플리케이션에서 직접 결과를 사용하십시오. 자세한 내용은 Aurora에서 기계 학습(ML) 기능 사용을 참조하십시오.	2019년 11월 25일

[Aurora 글로벌 데이터베이스에 엔진 모드 설정이 필요하지 않음]()	Aurora 글로벌 데이터베이스의 일부가 되게 하려는 클러스터를 생성할 때 `--engine-mode=global` 을 지정할 필요가 없습니다. 호환성 요구 사항을 충족하는 모든 Aurora 클러스터는 글로벌 데이터베이스의 일부가 될 자격이 있습니다. 예를 들어 클러스터에서는 현재 MySQL 5.6 호환성을 지닌 Aurora MySQL 버전 1을 사용해야 합니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스 작업]()을 참조하십시오.	2019년 11월 25일
[Aurora 글로벌 데이터베이스를 Aurora MySQL 버전 2에서 사용 가능]()	Aurora MySQL 2.07부터 MySQL 5.7 호환성을 지닌 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. 기본 또는 보조 클러스터에 `global` 엔진 모드를 지정할 필요가 없습니다. 새롭게 프로비저닝된 클러스터를 Aurora MySQL 2.07 이상을 이용해 Aurora 글로벌 데이터베이스에 추가할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스 작업]()을 참조하십시오.	2019년 11월 25일

랩 모드 없이도 Aurora MySQL 핫 행 경합 최적화 사용 가능	이제 핫 행 경합 최적화는 Aurora MySQL에 일반적으로 사용할 수 있으며 Aurora 랩 모드 설정이 ON 상태일 필요가 없습니다. 이 기능은 동일한 페이지의 행에 대해 경합하는 트랜잭션이 많은 워크로드의 처리량을 크게 향상시킵니다. 이러한 향상에는 Aurora MySQL에서 사용하는 잠금 해제 알고리즘의 변경이 수반됩니다.	2019년 11월 19일
랩 모드 없이도 Aurora MySQL 해시 조인 사용 가능	이제 해시 조인 기능은 Aurora MySQL에 일반적으로 사용할 수 있으며 Aurora 랩 모드 설정이 ON 상태일 필요가 없습니다. 이 기능은 동등 조인을 사용하여 많은 양의 데이터를 조인해야 하는 경우 쿼리 성능을 향상시킬 수 있습니다. 이 기능 사용에 대한 자세한 내용은 Aurora MySQL의 해시 조인 작업을 참조하세요.	2019년 11월 19일
Aurora MySQL 2.*에서 더 많은 db.r5 인스턴스 클래스 지원	Aurora MySQL 클러스터는 이제 db.r5.8xlarge, db.r5.16xlarge, db.r5.24xlarge 인스턴스 유형을 지원합니다. Aurora MySQL 클러스트의 인스턴스 유형에 대한 자세한 내용은 DB 인스턴스 클래스 선택을 참조하십시오.	2019년 11월 19일

Aurora MySQL 2.*에서 역추적 지원	이제 Aurora MySQL 2.* 버전에서 잘못된 테이블이나 잘못된 행을 삭제한 경우와 같이 사용자가 실수를 저지른 경우 빠르게 복구할 수 있는 기능을 제공합니다. 역추적을 사용하면 백업에서 복구할 필요 없이 데이터베이스를 이전 시점으로 되돌릴 수 있습니다. 또한 대용량 데이터베이스도 몇 초 안에 복구를 완료할 수 있습니다. 자세한 내용은 Aurora DB 클러스터 역추적을 참조하십시오.	2019년 11월 19일
Aurora에 대한 청구 태그 지원	이제 태그를 사용하여 Aurora 클러스터, Aurora 클러스터 내의 DB 인스턴스, I/O, 백업, 스냅샷 등과 같은 리소스에 대한 비용 할당을 추적할 수 있습니다. AWS Cost Explorer를 사용하여 각 태그와 관련된 비용을 볼 수 있습니다. Aurora에서 태그를 사용하는 방법에 대한 자세한 내용은 Amazon RDS 리소스 태깅을 참조하세요. 비용 분석에 태그를 사용하는 방법과 태그에 대한 일반적인 내용은 비용 할당 태그 사용 및 사용자 정의 비용 할당 태그를 참조하세요.	2019년 10월 23일

Aurora PostgreSQL용 데이터 API	이제 Aurora PostgreSQL에서는 Amazon Aurora Serverless v1 DB 클러스터에서 데이터 API를 사용할 수 있도록 지원합니다. 자세한 내용은 Aurora Serverless v1용 데이터 API 사용 단원을 참조하십시오.	2019년 9월 23일
Aurora PostgreSQL에서 CloudWatch 로그에 데이터베이스 로그 업로드 지원	Amazon CloudWatch Logs의 로그 그룹에 로그 데이터를 게시하도록 Aurora PostgreSQL DB 클러스터를 구성할 수 있습니다. CloudWatch Logs를 통해 로그 데이터에 대한 실시간 분석을 수행할 수 있고, CloudWatch를 사용하여 경보를 만들고 지표를 볼 수 있습니다. CloudWatch Logs를 사용하여 내구성이 뛰어난 스토리지에 로그 레코드를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시를 참조하십시오.	2019년 8월 9일
Aurora MySQL를 위한 멀티 마스터 클러스터	Aurora MySQL 멀티 마스터 클러스터를 설정할 수 있습니다. 이러한 클러스터에서는 각 DB 인스턴스에 읽기/쓰기 기능이 있습니다. 자세한 내용은 Aurora 멀티 마스터 클러스터 작업을 참조하십시오.	2019년 8월 8일

[Aurora PostgreSQL에서 Aurora Serverless v1 지원]	이제 Aurora PostgreSQL에서 Amazon Aurora Serverless v1를 사용할 수 있습니다. Aurora Serverless DB 클러스터는 애플리케이션 요구 사항을 기반으로 컴퓨팅 용량을 자동으로 시작, 종료, 확장 또는 축소합니다. 자세한 내용은 [Amazon Aurora Serverless v1 사용] 섹션을 참조하세요.	2019년 7월 9일
[Aurora MySQL의 계정 간 복제]	이제 AWS 계정 간에 Aurora MySQL DB에 대한 클러스터 볼륨을 복제할 수 있습니다. 공유는 AWS Resource Access Manager(AWS RAM)에서 승인합니다. 복제된 클러스터 볼륨은 기록 중 복사 메커니즘을 사용하기 때문에 새롭거나 변경된 데이터가 있을 경우 스토리지만 추가하면 됩니다. Aurora 복제에 대한 자세한 내용은 [Aurora DB 클러스터의 데이터베이스 복제]를 참조하십시오.	2019년 7월 2일
[Aurora PostgreSQL에서 db.t3 DB 인스턴스 클래스 지원]	이제 db.t3 DB 인스턴스 클래스를 사용하는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 [DB 인스턴스 클래스] 단원을 참조하십시오.	2019년 6월 20일

Amazon S3에서 Aurora PostgreSQL에 필요한 데이터 가져오기 지원	이제 Amazon S3 파일의 데이터를 Aurora PostgreSQL DB 클러스터의 테이블로 가져올 수 있습니다. 자세한 내용은 Amazon S3 데이터를 Aurora PostgreSQL DB 클러스터로 가져오기를 참조하십시오.	2019년 6월 19일
이제 Aurora PostgreSQL에서 클러스터 캐시 관리를 통해 장애를 신속하게 복구	이제 Aurora PostgreSQL에서는 장애 조치가 이뤄지는 경우 클러스터 캐시 관리를 제공하여 기본 DB 인스턴스의 신속한 복구를 보장합니다. 자세한 내용은 장애 조치 후 클러스터 캐시 관리를 통한 신속한 복구를 참조하십시오.	2019년 6월 11일
Aurora Serverless v1용 데이터 API 정식 출시	데이터 API를 사용하여 웹 서비스 기반 애플리케이션으로 Aurora Serverless v1 클러스터에 액세스할 수 있습니다. 자세한 내용은 Aurora Serverless v1용 데이터 API 사용 단원을 참조하십시오.	2019년 5월 30일
Aurora PostgreSQL에서 데이터베이스 활동 스트림으로 데이터베이스 모니터링 지원	이제 Aurora PostgreSQL에 데이터베이스 활동 스트림이 포함되어, 관계형 데이터베이스에서 데이터베이스 활동의 데이터 스트림을 거의 실시간으로 제공하게 됩니다. 자세한 내용은 데이터베이스 활동 스트림 사용을 참조하세요.	2019년 5월 30일

Amazon Aurora 권장 사항	Amazon Aurora에서 이제 Aurora 리소스에 대한 자동 권장 사항을 제공합니다. 자세한 내용은 Amazon Aurora 권장 사항 사용을 참조하십시오.	2019년 5월 22일
Aurora 글로벌 데이터베이스에 대한 성능 개선 도우미 지원	이제 Aurora Global Database에서 성능 개선 도우미를 사용할 수 있습니다. Aurora용 성능 개선 도우미에 대한 자세한 내용은 Amazon RDS 성능 개선 도우미 사용을 참조하세요. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Aurora 글로벌 데이터베이스 작업을 참조하십시오.	2019년 5월 13일
Aurora MySQL 5.7에 성능 개선 도우미 사용 가능	이제 MySQL 5.7과 호환되는 Aurora MySQL 2.x 버전에 Amazon RDS 성능 개선 도우미를 사용할 수 있습니다. 자세한 내용은 Amazon RDS 성능 개선 도우미 사용을 참조하십시오.	2019년 5월 3일
Aurora 글로벌 데이터베이스를 더 많은 AWS 리전에서 사용 가능	이제 Aurora를 사용할 수 있는 대부분의 AWS 리전에서 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. Amazon Aurora Global Database에 대한 자세한 내용은 Amazon Aurora Global Database 작업을 참조하세요.	2019년 4월 30일

Aurora Serverless v1에 최소 용량 1 적용	Aurora Serverless v1 클러스터에 사용할 수 있는 최소 용량 설정 값은 1입니다. 전에는 최소 용량이 2였습니다. Aurora 서버리스 용량 값 지정에 대한 자세한 내용은 Aurora Serverless v1 DB 클러스터의 용량 설정을 참조하십시오.	2019년 4월 29일
Aurora Serverless v1 제한 시간 조치	이제 Aurora Serverless v1 용량 변경 제한 시간이 초과되었을 때 취할 조치를 지정할 수 있습니다. 자세한 내용은 용량 변경에 대한 제한 시간 조치를 참조하십시오.	2019년 4월 29일
초당 청구	Amazon RDS는 이제 온디맨드 인스턴스의 경우 AWS GovCloud(미국)를 제외한 모든 AWS 리전에서 1초 증분 단위로 청구됩니다. 자세한 내용은 Aurora에 대한 DB 인스턴스 청구를 참조하세요.	2019년 4월 25일
AWS 리전 간 Aurora Serverless v1 스냅샷 공유	Aurora Serverless v1에서 스냅샷은 항상 암호화됩니다. 사용자 고유의 AWS KMS key로 스냅샷을 암호화하는 경우 이제는 AWS 리전 간에 스냅샷을 복사하거나 공유할 수 있습니다. Aurora Serverless v1 DB 클러스터의 스냅샷에 대한 자세한 내용은 Aurora Serverless v1 및 스냅샷을 참조하세요.	2019년 4월 17일

Amazon S3에서 MySQL 5.7 백업 복원	이제 MySQL 버전 5.7 데이터베이스의 백업을 생성하여 Amazon S3에 저장한 다음 새로운 Aurora MySQL DB 클러스터에 백업 파일을 복원할 수 있습니다. 자세한 내용은 외부 MySQL 데이터베이스에서 Aurora MySQL DB 클러스터로 데이터 마이그레이션을 참조하십시오.	2019년 4월 17일
리전 간 Aurora Serverless v1 스냅샷 공유	Aurora Serverless v1에서 스냅샷은 항상 암호화됩니다. 사용자 고유의 AWS KMS key(으)로 스냅샷을 암호화하는 경우 이제는 리전 간에 스냅샷을 복사하거나 공유할 수 있습니다. Aurora Serverless v1 DB 클러스터의 스냅샷에 대한 자세한 내용은 Aurora 서버리스 및 스냅샷을 참조하세요.	2019년 4월 16일
Aurora 개념 증명 자습서	개념 증명을 수행하여 Aurora에서 애플리케이션 및 워크로드를 시험하는 방법을 배울 수 있습니다. 전체 자습서는 Aurora 개념 증명 수행을 참조하십시오.	2019년 4월 16일
Aurora Serverless v1에서 Amazon S3 백업으로부터 복구 지원	이제 Amazon S3에서 Aurora Serverless 클러스터로 백업을 가져올 수 있습니다. 이 절차에 관한 자세한 내용은 Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션을 참조하십시오.	2019년 4월 16일

Aurora Serverless v1을 위한 새로운 수정 가능 파라미터	이제 다음과 같은 DB 파라미터를 수정하여 Aurora Serverless v1 클러스터에 사용할 수 있습니다. `innodb_file_format` , `innodb_file_per_table` , `innodb_large_prefix` , `innodb_lock_wait_timeout` , `innodb_monitor_disable` , `innodb_monitor_enable` , `innodb_monitor_reset` , `innodb_monitor_reset_all` , `innodb_print_all_deadlocks` , `log_warnings` , `net_read_timeout` , `net_retry_count` , `net_write_timeout` , `sql_mode`, `tx_isolation` . Aurora Serverless v1 클러스터용 구성 파라미터에 대한 자세한 내용은 Aurora Serverless v1 및 파라미터 그룹을 참조하십시오.	2019년 4월 4일
Aurora PostgreSQL에서 db.r5 DB 인스턴스 클래스 지원	이제 db.r5 DB 인스턴스 클래스를 사용하는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원을 참조하십시오.	2019년 4월 4일

Aurora PostgreSQL 논리적 복제	이제 PostgreSQL 논리적 복제를 사용해 Aurora PostgreSQL DB 클러스터용 데이터베이스의 일부를 복제할 수 있습니다. 자세한 내용은 PostgreSQL 논리적 복제 사용을 참조하십시오.	2019년 3월 28일
Aurora MySQL 2.04에 대한 GTID 지원	이제 MySQL 5.7의 전역 트랜잭션 ID(GTID) 기능을 통해 복제를 사용할 수 있습니다. 이 기능을 통해 Aurora MySQL과 외부 MySQL 5.7 호환 데이터베이스 간의 이진 로그(binlog) 복제 작업이 간소화됩니다. 복제에서는 Aurora MySQL 클러스터를 원본 또는 대상으로 사용할 수 있습니다. 이 기능은 Aurora MySQL 2.04 이상에 사용할 수 있습니다. GTID 기반 복제 및 Aurora MySQL에 대한 자세한 내용은 Aurora MySQL에 대한 GTID 기반 복제 사용을 참조하십시오.	2019년 3월 25일

Aurora Serverless v1 로그를 Amazon CloudWatch에 업로드	이제 Aurora에서 데이터베이스 로그를 Aurora Serverless v1 클러스터용 CloudWatch에 업로드하게 할 수 있습니다. 자세한 내용은 Aurora 서버리스 DB 클러스터 보기를 참조하세요. 이러한 개선 사항 중 일부로 이제 DB 클러스터 파라미터 그룹에 인스턴스 수준 파라미터의 값을 정의할 수 있으며, 이 값은 DB 파라미터 그룹에서 재정의하지 않는 한 클러스터 내 모든 DB 인스턴스에 적용됩니다. 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업을 참조하십시오.	2019년 2월 25일
Aurora MySQL에서 db.t3 DB 인스턴스 클래스 지원	이제 db.t3 DB 인스턴스 클래스를 사용하는 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원을 참조하십시오.	2019년 2월 25일
Aurora MySQL에서 db.r5 DB 인스턴스 클래스 지원	이제 db.r5 DB 인스턴스 클래스를 사용하는 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원을 참조하십시오.	2019년 2월 25일
Aurora MySQL용 성능 개선 도우미 카운터	이제 Aurora MySQL DB 인스턴스용 성능 개선 도우미 차트에 성능 카운터를 추가할 수 있습니다. 자세한 내용은 성능 개선 도우미 대시보드 구성 요소를 참조하십시오.	2019년 2월 19일

Amazon RDS 성능 개선 도우미에서 Aurora MySQL에 대해 더 많은 SQL 텍스트 보기 지원	Amazon RDS 성능 개선 도우미는 이제 Aurora MySQL DB 인스턴스에 대해 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트를 볼 수 있도록 지원합니다. 자세한 내용은 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트 보기 단원을 참조하십시오.	2019년 2월 6일
Amazon RDS 성능 개선 도우미에서 Aurora PostgreSQL에 대해 더 많은 SQL 텍스트 보기 지원	Amazon RDS 성능 개선 도우미는 이제 Aurora PostgreSQL DB 인스턴스에 대해 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트를 볼 수 있도록 지원합니다. 자세한 내용은 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트 보기 단원을 참조하십시오.	2019년 1월 24일
Aurora 백업 결제	Amazon CloudWatch 지표 `TotalBackupStorageBilled` , `SnapshotStorageUsed` 및 `BackupRetentionPeriodStorageUsed` 를 사용하여 Aurora 백업의 공간 사용량을 모니터링할 수 있습니다. CloudWatch 지표를 사용하는 방법에 대한 자세한 내용은 모니터링 개요를 참조하세요. 백업 데이터용 스토리지를 관리하는 방법에 대한 자세한 내용은 Aurora 백업 스토리지 사용량 파악을 참조하십시오.	2019년 1월 3일

성능 개선 도우미 카운터	이제 성능 개선 도우미 차트에 성능 카운터를 추가할 수 있습니다. 자세한 내용은 성능 개선 도우미 대시보드 구성 요소를 참조하십시오.	2018년 12월 6일
Aurora 글로벌 데이터베이스	이제 Aurora 글로벌 데이터베이스를 만들 수 있습니다. Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있어, 대기 시간이 짧은 글로벌 읽기와 리전 전체의 중단으로부터 재해 복구를 지원합니다. 자세한 내용은 Amazon Aurora Global Database 작업을 참조하세요.	2018년 11월 28일
Aurora PostgreSQL의 쿼리 계획 관리	Aurora PostgreSQL이 이제 PostgreSQL 쿼리 실행 계획을 관리하는 데 사용할 수 있는 쿼리 계획 관리를 제공합니다. 자세한 내용은 Aurora PostgreSQL용 쿼리 실행 계획 관리를 참조하십시오.	2018년 11월 20일
Aurora Serverless v1용 쿼리 편집기(베타)	Aurora Serverless v1 클러스터의 Amazon RDS 콘솔에서 SQL 문을 실행할 수 있습니다. 자세한 내용은 Aurora Serverless v1용 쿼리 편집기 사용 단원을 참조하십시오.	2018년 11월 20일

Aurora Serverless v1용 데이터 API(베타)	데이터 API를 사용하여 웹 서비스 기반 애플리케이션으로 Aurora Serverless v1 클러스터에 액세스할 수 있습니다. 자세한 내용은 Using the Data API for Aurora Serverless 단원을 참조하십시오.	2018년 11월 20일
Aurora Serverless v1에 대한 TLS 지원	Aurora Serverless v1 클러스터에서는 TLS/SSL 암호화를 지원합니다. 자세한 내용은 TLS/SSL for Aurora Serverless 단원을 참조하십시오.	2018년 11월 19일
사용자 지정 엔드포인트	이제 임의의 DB 인스턴스 세트와 연결된 엔드포인트를 만들 수 있습니다. 이 기능은 일부 DB 인스턴스의 용량 또는 구성이 다른 인스턴스와 다른 경우 Aurora 클러스터의 로드 밸런싱과 고가용성에 도움이 됩니다. 인스턴스 엔드포인트를 통해 특정 DB 인스턴스에 연결하는 대신에 사용자 지정 엔드포인트를 사용할 수 있습니다. 자세한 내용은 Amazon Aurora 엔드포인트를 참조하십시오.	2018년 11월 12일
Aurora PostgreSQL의 IAM 인증 지원	Aurora PostgreSQL이 이제 IAM 인증을 지원합니다. 자세한 내용은 IAM 데이터베이스 인증을 참조하십시오.	2018년 11월 8일

복원 및 특정 시점으로 복구를 위한 사용자 지정 파라미터 그룹	이제 스냅샷을 복원하거나 특정 시점으로 복구 작업을 수행할 때 사용자 지정 파라미터 그룹을 지정할 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 및 DB 클러스터를 지정된 시간으로 복원을 참조하십시오.	2018년 10월 15일
Aurora DB 클러스터에 대한 삭제 방지	DB 클러스터에 대해 삭제 방지를 활성화하면 모든 사용자가 데이터베이스를 삭제할 수 없습니다. 자세한 내용은 DB 인스턴스 삭제를 참조하십시오.	2018년 9월 26일
Aurora 중지/시작 기능	이제 단일 작업으로 전체 Aurora 클러스터를 중지하거나 시작할 수 있습니다. 자세한 내용은 Aurora 클러스터 중단 및 시작을 참조하십시오.	2018년 9월 24일
Aurora MySQL용 병렬 쿼리 기능	Aurora MySQL은 이제 Aurora 스토리지 인프라에서 쿼리에 대한 I/O 작업을 병렬화하는 옵션을 제공합니다. 이 기능은 데이터 집약적인 분석 쿼리의 속도를 높이는데 이는 워크로드에서 가장 시간이 많이 걸리는 작업입니다. 자세한 내용은 Aurora MySQL용 병렬 쿼리를 참조하세요.	2018년 9월 20일
새 설명서	이 설명서는 Amazon Aurora 사용 설명서의 첫 번째 릴리스입니다.	2018년 8월 31일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하십시오.